ESCUELA SUPERIOR POLICTÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ANÁLISIS DE PYTHON CON DJANGO FRENTE A RUBY ON RAILS PARA DESARROLLO ÁGIL DE APLICACIONES WEB. CASO PRÁCTICO: DECH."

TESIS DE GRADO

Previa obtención del título de:

INGENIERO EN SISTEMAS INFORMÁTICOS

Presentado por:

PAUL DAVID CUMBA ARMIJOS

BYRON AUGUSTO BARRENO PILCO

RIOBAMBA – ECUADOR

2012

AGRADECIMIENTO

La vida es una carrera incesante por alcanzar metas y objetivos, los mismo que son la recompensa del esfuerzo y dedicación que con el apoyo de las personas que quieres y estimas pueden llegar hacerse una realidad, es por eso que agradezco principalmente a Dios, a mi mami Susana, a mi familia, a mi director Danilo, mi amigo Byron y demás maestros, compañeros y amigos que de una u otra manera me supieron brindar el apoyo y el incentivo para culminar con éxito esta etapa de mi vida.

Paul Cumba Armijos

Dar gracias a Dios por lo que se tiene, allí comienza el arte de vivir, a mis padres Miriam y Augusto, que siempre estuvieron conmigo en mis éxitos y fracasos apoyándome en todo momento, a mi familia que me han ayudado en mucho con sus valiosos consejos, a mis amigos en especial mi gran amigo Paul, al Ing. Danilo Pastor por brindarme la oportunidad de recurrir a su capacidad y experiencia, al Ing. Geovanny Aucancela por confiar en mí y brindarme la oportunidad de incursionar en el mundo laboral. Gracias a todos Uds. que de una y otra manera han aportado y son parte de este nuevo logro en mi vida.

Byron Augusto Barreno Pilco.

DEDICATORIA

Dedico este trabajo a mi Mamita Susana Armijos que ha sido el pilar fundamental en todo el desarrollo de mi vida, que siempre supo extenderme su abrigo cuando más lo necesite, a mis hermanos Andrea, Rogelio y Anthony y mi sobrino Sebitas, que gracias a su apoyo, comprensión y constancia me hace poder ser una persona de bien para la sociedad.

Paul Cumba Armijos.

Se lo dedico a mi madre Miriam Pilco y a mi padre Augusto Barreno, quienes me trajeron a la vida que siempre han luchado incansablemente para que sus hijos podamos alcanzar nuestras metas, a mi madre por siempre estar a mi lado brindándome su gran amor, formándome en una persona de buenos valores, a mi padre por el gran apoyo demostrándome lo que es un hombre responsable y trabajador.

Byron Augusto Barreno Pilco.

FIRMAS RESPONSABILIDAD Y NOTA

NOMBRE	**FIRMA**	**FECHA**
Ing. Iván Menes Camejo **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRONICA**	……..	……
Ing. Raúl Rosero **DIRECTOR DE LA ESCUELA DE INGENIERIA EN SISTEMAS**	……..	……
Ing. Danilo Pastor **DIRECTOR DE TESIS**	……..	……
Ing. Gloria Arcos **MIEMBRO DEL TRIBUNAL**	……..	……
Tlgo. Carlos Rodríguez **DIRECTOR DEL CENTRO DE DOCUMENTACIÓN**	……..	……

RESPONSABILIDAD DEL AUTOR

"Nosotros Paul David Cumba Armijos y Byron Augusto Barreno Pilco, somos los responsables de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenece a la Escuela Superior Politécnica de Chimborazo."

FIRMAS:

_____ _____

Paul David Cumba Armijos Byron Augusto Barreno Pilco

ABREVIATURAS

A

ABE Ancho de Banda de Entrada

ABS Ancho de Banda de Salida

ACL Access Control List

ACO Access Control Object **-** Objetos de control de acceso

AM Agile Modeling – Modelado Agile

AJAX Asynchronus Javascript and XML – Javascript y XML Asíncrono

API Aplication Programming Interface. Interfaz de Programación de Aplicaciones
ARO Access Request Object Objetos que desean acceder a recursos (usuario o grupos)

B

BD Base de Datos

BSD Berkeley Software Distribution – Distribución de Software Berkeley

C

CLR. Common Language Runtime – Lenguaje Común de Ejecución

CRC Clase Responsabilidad y Colaboración

CSS Cascading Style Sheets – Hoja de Estilo en Cascada

CUxP Consumo de Usuarios por Prueba

CUxT Consumo de Usuarios por Transacción

D

DECH Dirección de Estudios de Chimborazo

DJ Django

DLL Dynamically Linked Library - Librería de Enlace Dinámico

F

FAQ Frencuently Asked Question - Preguntas y Respuestas Frecuentes

G

GPL General Public License - Licencia Pública General **GUI** Graphical User Interface- Interfaz gráfica de usuario **H**

HTML Hypertext Markup Language - Lenguaje de marcado de hipertexto

HTTP Hypertext Transfer Protocol – Protocolo de transferencia de Hipertexto

HW Hardware

I

IBM International Business Machines – Maquinas de Negocios Internacional **IDE** Integrated Development Environment - Entorno de Desarrollo Integrado **IIS** Internet Information Server – Servidor de información de Internet

IDE: (Integrated Development environment – Entorno de desarrollo integrado)

IMAP: (Internet Message Access protocol – Protocolo de acceso a mensajes de internet)

ISAPI Internet Server (**API**) Application Programming Interface - Interfaz de programación de aplicaciones para Internet Information Server

L

LOC Line of Code

M

MABR Mínimo Ancho de Banda Requerido

MD5 Message-Digest Algorithm 5

MPL Mozilla Public License

MVC Modelo Vista Controlador

N

NAT Network AddressTranslation

O

ODBC Open DataBase Connectivity

OO Orientados a objetos

ORM Object Relational Mapping

P

P Promedio máximo

PHP Personal Home Page

PHTML PHP Hypertext Markup Language

POO Programación Orientada a Objetos

POP3: (Post Office Protocol – protocolo de oficina de correos)

R

RoR Ruby on Rails

RAD: (Rapid Application Development – Desarrollo Rápido de Aplicaciones)

S

SOAP Simple Object Access Protocol – Protocolo de Acceso a objetos simples

SQL Structure Query Language - Lenguaje de Consulta Estructurado

U

URI: (Uniform Resource Identifier – Identificador uniforme del recurso)

URL Uniform Resource Locator - Localizador Uniforme de Recursos

UML Unified Modeling Language - Lenguaje de modelado unificado

V

VR Velocidad de la Red

VRTU Velocidad de Red Para el Total de Usuarios

X

XML Extended Markup Language - Lenguaje de marcado extendido

XP Extreme Programming – Programación extrema

INDICE GENERAL

INDICE DE FIGURAS

CAPÍTULO II

CAPÍTULO III

CAPITULO IV

INDICE DE TABLAS

CAPÍTULO II

CAPÍTULO III

CAPITULO IV

INTRODUCCIÓN

El presente trabajo de investigación de tesis previo a la obtención del título de Ingeniería en Sistemas Informáticos, trata sobre el "ANÁLISIS DE PYTHON CON DJANGO FRENTE A RUBY ON RAILS PARA DESARROLLO ÁGIL DE APLICACIONES WEB. CASO PRÁCTICO: DECH.".

Es un estudio y análisis investigativo de los Frameworks de desarrollo web Python con Django y Ruby on Rails que se ubican como los más populares en la actualidad a fin de obtener como resultado el Frameworks mas optimo para el desarrollo ágil de aplicaciones web.

Como precedente al momento de desarrollar aplicaciones resulta muy factible la utilización de un framework que facilite el desarrollo ágil pero esto puede resultar un poco confuso a la hora de decidirse por una determinada herramienta antes de empezar a desarrollar una aplicación, ya que nos enfrentamos a dos tipos de tecnologías de muy similares características, basados en los mismos conceptos, y lo más importante garantizando, tanto Ruby on Rails, así como también Django (Python), la rapidez y la simplicidad en el desarrollo de aplicaciones web, que es el objetivo primordial para los desarrolladores en el mundo.

En la actualidad el desarrollo de aplicaciones web requiere la aplicación de tecnologías las cuales nos permitan obtener la mayor optimización de recursos tanto de hardware como de software y es aquí que la aplicación de ingeniería en el desarrollo de software permite establecer procesos correctos de desarrollo.

El presente trabajo de investigación contiene los siguientes capítulos:

El en capítulo 1 se presenta el planteamiento de la investigación, antecedentes, hipótesis, métodos y técnicas, es todo el marco referencial para el desarrollo de la tesis.

En el capítulo 2 se detalla los aspectos teóricos motivo de la investigación, conceptos, terminologías, referenciadas al objeto de estudio.

Continuando con el capítulo 3 se establecen los parámetros de productividad en el desarrollo de software.

El capitulo 3 trata de el desarrollo del análisis de los Frameworks Python con Django y Ruby on Rails, estableciendo parámetros de productividad en el desarrollo de software.

Y así determinar cual brinda una mayor productividad al momento de realizar un desarrollo ágil, sometiendo a cada prototipo a diferentes escenarios de pruebas, finalizando con la demostración de la hipótesis.

En el capítulo 4 se detalla la parte aplicativa de la tesis, contiene los requisitos de ingeniería de software, desarrollo ágil de aplicaciones web, estándares de desarrollo, todo referente al sistema web para la contratación docente de la Dirección de Estudios de Chimborazo.

Tanto Django así como Ruby on Rails presentan similares características las cuales nos brindan un optimo beneficio al momento de desarrollar ágilmente aplicaciones web diferenciadas entre sí mas no por su conceptualidad sino por su implementación y estructura. Lo cual permite obtener una clara visión en el enfoque de un desarrollo más productivo de aplicaciones web.

En síntesis este trabajo de análisis e investigación ha permitido describir las tecnologías más eficientes en el ámbito del desarrollo de aplicaciones web. Teniendo en consideración los resultados obtenidos en base a la comparación y de acuerdo a los parámetros de productividad establecidos, se puede concluir que la aplicación de la tecnología Python con Django mejora la productividad en un 91,82% frente al 84,61% de Ruby on Rails para un desarrollo ágil de aplicaciones web.

CAPÍTULO I:

MARCO REFERENCIAL

1.1. Antecedentes

Debido a la gran exigencia y demanda en la utilización de sistemas enfocados a la web y al alto incremento de usuarios del internet, es importante enfocarse al poder desarrollar este tipo aplicaciones web de de manera más ágil y eficiente, sin descuidar ningún aspecto en la calidad de software a desarrollar. Lo cual representa desafíos para nosotros como desarrolladores el poder crear aplicaciones más rápidas, ligeras y robustas que permitan utilizar la web.

Se debe tomar en cuenta la versatilidad que debe poseer una interfaz Web, la cual debe cumplir con la filosofía web (libertad de acceso, universalidad, y rapidez), siendo capaz de ser utilizada en cualquier tipo de browser (internet explorer, mozilla firefox, etc.), en cualquier plataforma (Windows, Linux, Mac), con cualquier conexión a internet, y proporcionando rapidez en el manejo del sitio. Para el desarrollo de este tipo de aplicaciones existen diferentes lenguajes para poder crear Aplicaciones Web entre los cuales se destacan PHP, JSP, ASP, RUBY, PYTHON, entre otros, en los cuales se realizaba un proceso de desarrollo línea por línea, teniendo que programar todo lo que sea necesario imposibilitando mejorar los tiempos de desarrollo, es por ello que

aparecieron los conocidos Frameworks, los cuales interactúan directamente con los lenguajes de programación.

Los frameworks no son lenguajes de programación y se pueden definir como una arquitectura de software, que permite una colaboración directa en el desarrollo, el mismo que utiliza módulos de software que cuentan con procedimientos, librerías, clases y de mas funciones de un lenguaje en concreto, organizadas para ser reutilizadas en el desarrollo, que facilitan la construcción rápida de aplicaciones web.

En la actualidad con el gran crecimiento del internet como una herramienta necesaria en el acceso y compartición de la información, han venido apareciendo tecnologías y herramientas que permiten crear de manera más fácil un sitio para internet, y es así que en justamente a finales del año 2004 y principios del 2005 aparece Ruby on Rails como un completo entorno para desarrollar aplicaciones web, basado en el patrón de diseño MVC (Modelo Vista Controlador), de código abierto escrito en el lenguaje de programación Ruby. Este framework trata de simplificar el proceso de desarrollar aplicaciones para el mundo real, escribiendo menos código y realizando menos configuraciones, basado en dos principios fundamentales “No lo vuelvas a repetir” y “convención antes que configuración”. Esto ha permitido lograr desarrollar aplicaciones diez veces más deprisa con Rails que lo que tardarías con un framework típico de Java y esto es posible gracias a el lenguaje de programación Ruby ya que hay muchas cosas que son sencillas de hacer en Ruby pero que ni tan siquiera pueden hacerse en otros lenguajes, y Rails saca provecho de esto.

Por otro lado en el mismo año siendo casi contemporáneos y basados en las mismas ideas de Ruby on Rails, en julio del 2005 es liberado Django, un framework el cual es muy popular para el desarrollo de aplicaciones web. Está basado en el lenguaje de programación Python y de la misma manera sigue el patrón de diseño MVC, pero este framework incluye y aporta soluciones propias, las mismas que garantizan un desarrollo ágil, sencillo y sobre todo rápido de aplicaciones web. El objetivo fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis la reutilización de código, la conectividad y extensibilidad de componentes, el desarrollo

rápido y el principio de DRY (Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Esto puede resultar un poco confuso a la hora de decidirse por una determinada herramienta antes de empezar a desarrollar una aplicación, ya que nos enfrentamos a dos tipos de tecnologías de muy similares características, basados en los mismos conceptos, y lo más importante garantizando, tanto Ruby on Rails, así como también Django (Python), la rapidez y la simplicidad en el desarrollo de aplicaciones web, que es de mucho interés para nosotros los programadores.

Entonces como poder determinar que framework permite aprovechar de mejor manera las características que ofrece Ruby on Rails, o Django, con el único objetivo de poder enfocarnos a desarrollar aplicaciones web de una manera mucho más optima, priorizando el tiempo de desarrollo. Y es cuando nos encontramos con el gran problema de poder decidir que herramienta utilizar en la construcción de este tipo de aplicaciones, ya que ambos frameworks son de código abierto, establecen sus ventajas y desventajas de manera muy similar estableciendo primordialmente la facilidad de en la construcción y sobre todo la velocidad en el tiempo de desarrollo de aplicaciones web agiles, sin descuidar la funcionalidad de las aplicaciones.

La Dirección de Educación de Chimborazo, a través de la División de Recursos Humanos, se encuentra en proceso de contratación docente para cubrir los distintos requerimientos de las instituciones educativas de la provincia. El procesamiento de la información de los docentes e instituciones educativas se lo realiza de forma manual, mediante la recopilación de documentos escritos y el manejo de tablas de Excel, enfrentando problemas de dificultad, lentitud, desorganización e inconsistencias de datos, los cuales imposibilitan la optimización del tiempo al momento de realizar la contratación de los docentes.

1.2. Justificación del Proyecto de Tesis

1.2.1. Justificación Teórica

El presente análisis pretende estudiar las tecnologías más populares existentes en el mercado en la actualidad, como lo es Ruby on Rails y Django (Python) para el desarrollo de aplicaciones web, enfocándonos principalmente en la agilidad que pueden prestar estos lenguajes y sus respectivos frameworks al momento de construir este tipo de aplicaciones.

Está claro que ambas tecnologías, además de ser muy similares presentan un gran crecimiento y acogida entre los desarrolladores de aplicaciones web, por lo cual se considera relevante el poder determinar por medio de esta investigación el comportamiento de estas tecnologías que permitan describir con claridad la mejor opción para un desarrollo ágil y rápido de una aplicación web.

La investigación está dirigida a desarrolladores que desean elegir uno de estos dos frameworks de referencia, para su propio desarrollo y así poder determinar los efectos de esta comparación.

1.2.2. Justificación Práctica

Con el propósito de determinar la tecnología más adecuada se construirán prototipos de desarrollo, en los cuales permitan evaluar mediante parámetros la agilidad en el desarrollo de aplicaciones web. Los prototipos consistirán en pequeñas aplicaciones web, las mismas que serán desarrolladas usando Ruby on Rails y Python con Django.
Con la situación actual en la que se encuentra la División de Recursos Humanos de la DECH es necesario implementar una aplicación web, la misma que permitirá automatizar el proceso de contratación docente, el cual contará con una interfaz amigable con el usuario, estableciendo políticas de acceso para el ingreso al sistema, además posibilitará tener un control de la información de los docentes los cuales están y serán contratados por la Dirección. La aplicación web facilitará la realización de reportes, los mismos que detallaran información de los docentes, las escuelas y la ubicación geográfica donde se encuentran laborando. Esto podrá facilitar a los analistas de recursos humanos tener un control más óptimo al momento de ejecutar nuevos contratos, renovaciones, terminaciones y renuncias de los maestros lo cual establecerá

una mejor administración del personal docente, y la distribución hacia las distintas instituciones educativas.

1.3. Objetivos

1.3.1. Objetivo General:

Realizar un análisis de las tecnologías Django sobre Python frente a Ruby on Rails para determinar cuál es más adecuada para el desarrollo ágil de Aplicaciones Web.

1.3.2. Objetivos Específicos.

- Estudiar los beneficios y características que presentan los frameworks RUBY ON RAILS y PYTHON CON DJANGO para el desarrollo de aplicaciones web.

- Establecer parámetros de comparación los cuales nos permitan determinar con claridad que lenguaje y framework se adapta mejor a un desarrollo ágil de aplicaciones web.

- Analizar mediante métodos formales los frameworks, al utilizar RUBY ON RAILS y PYTHON CON DJANGO basados en la construcción de prototipos.

- Realizar una aplicación web que cumpla con los requerimientos establecidos por la División de Recursos Humanos de la Dirección de Estudios de Chimborazo, para automatizar el proceso de contratación docente.

1.4. Hipótesis

La aplicación de la tecnología Python con Django mejora la productividad en el desarrollo ágil de aplicaciones web que al usar la tecnología Ruby on Rails.

1.5. Métodos y Técnicas

1.5.1. Métodos

Para la comprobación de nuestra hipótesis será necesaria la aplicación de un método científico, experimental que permitirá establecer una secuencia ordenada de acciones que nos llevarán a establecer las conclusiones sobre el uso eficiente de las tecnologías de moda en el mercado para el desarrollo ágil de aplicaciones web.

Para el desarrollo de la aplicación web y la implementación de la aplicación web para la automatización del proceso de contratación docente de la DECH, se empleará la metodología ágil SCRUM.

1.5.2. Técnicas

Para lo que tiene que ver en cuanto a fuentes de información se utilizarán principalmente lo que se refieran al tema de investigación como la observación, libros, revistas, páginas web, etc. además, se contactará a especialistas en los temas mediante video-llamadas, foros en línea, etc. También se empleará la observación y análisis por parte de los investigadores para hacer deducciones e inducciones sobre el tema de tesis.

Para la aplicación se construirán prototipos y módulos de pruebas los mismos que por medio pruebas experimentales (medición del tiempo, tamaños de aplicación, utilización) y la observación directa, se obtendrá los resultados lo cual determinará las necesidades que se debe cubrir.

CAPÍTULO II:

MARCO TEÓRICO

2.1. Definiciones

2.1.1. Lenguajes de Programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente. Aunque muchas veces se usa lenguaje de programación y lenguaje informático como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el HTML. [22]

Figura II. 1: Lenguajes de Programación

2.1.2. Lenguaje de Programación Python

Python, según Raúl González Duque, es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses "Monty Python". Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.

Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos. [2].

2.1.2.1. Historia

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

El nombre del lenguaje proviene de la afición de su creador original, Guido van Rossum, por los humoristas británicos Monty Python.

Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como *Benevolente Dictador Vitalicio*(en inglés: *Benevolent Dictator for Life*, BDFL).

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: `str`, `list`, `dict`, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado deModula-3; van Rossum describe el módulo como "una de las mayores unidades de programación de Python". El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula `else`. En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje. [29]

```
def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
```

Figura II. 2: Código Python con coloreado de sintaxis

2.1.2.2. Características y Paradigmas

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y conteo de referencias para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse en algunas situaciones hostiles a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme. [29]

Algunas Características de Python [26]

- Simple
- Sencillo de Aprender
- Libre y Fuente Abierta

- Lenguaje de Alto Nivel
- Portable
- Interpretado
- Orientado a Objetos
- Ampliable
- Incrustable
- Librerías Extendidas

2.1.2.3. Filosofía

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" ("unpythonic" en inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en El Zen de Python. [29].

Principios de la Filosofía [9]

- Hermoso es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Ralo es mejor que denso.
- La legibilidad importa.
- Los casos especiales no son tan especiales como para romper las reglas.
- Aunque lo práctico vence a lo puritano.
- Los errores nunca deben pasar desapercibidos.
- A menos que sean explícitamente silenciados.
- Ante la ambigüedad, rechaza la tentación de adivinar.
- Debe haber una forma (y preferentemente sólo una forma) obvia de hacerlo.

- Aunque esa forma no sea obvia al principio a menos que seas holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede ser una buena idea.
- Los espacios de nombre son geniales; hagamos más de ellos

2.1.2.4. Modo Interactivo

El intérprete de Python estándar incluye un modo interactivo, en el cual se escriben las instrucciones en una especie de Shell: las expresiones pueden ser introducidas una por una, pudiendo verse el resultado de su evaluación inmediatamente. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como también para los programadores más avanzados: se pueden probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa.

Existen otros programas, tales como IDLE e IPython, que añaden funcionalidades extra al modo interactivo, como el auto-completar código y el coloreado de la sintaxis del lenguaje.

Ejemplo del modo interactivo:

```
>>> 1 + 1
2
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Figura II. 3: Ejemplo del modo interactivo de Python

2.1.2.5. Elementos del Lenguaje [29]

Python fue diseñado para ser leído con facilidad. Entre otras cosas se utilizan palabras en inglés donde otros lenguajes utilizarían símbolos. En vez de delimitar los bloques de código mediante el uso de llaves ({}), Python utiliza la Indentación. Esto hace que la misma sea obligatoria, ayudando a la claridad y consistencia del código escrito (incluso entre varios desarrolladores.

Comentarios

Los comentarios se inician con el símbolo #, y se extienden hasta el final de la línea. Al comentar una línea, esta no es tenida en cuenta por el interprete lo cual es útil si deseamos poner información adicional en nuestro código como por ejemplo un explicativo que comente que hacen dichas líneas, que falta o haría falta hacer (muy útil para los programadores al momento de leer un código hecho por otro programador).

```
# -*- coding: utf-8 -*-
# Comentario en una línea en Python.
print('Hola mundo') # También es posible añadir un comentario después de una línea de código
```

Variables

Las variables se definen de forma dinámica. El signo igual (=) se usa para asignar valores a las variables:

```
x = 1
x = 'texto' # esto es posible porque los tipos son asignados dinámicamente
```

- **Tipos de datos**

Python soporta implícitamente una gran variedad de tipos de datos.

Tabla II. I: Tipos de Datos de Python

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	'Cadena'
unicode	Cadena	Versión Unicode de str	u'Cadena'
list	Secuencia	Mutable, puede contener objetos de diversos tipos	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	(4.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, no contiene duplicados	set([4.0, 'Cadena', True])
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	frozenset([4.0, 'Cadena', True])
dict	Mapping	Grupo de pares clave:valor	{'key1': 1.0, 'key2': False}
int	Número entero	Precisión fija, convertido en *long* en caso de overflow.	42
long	Número entero	Precisión arbitraria	42L ó 456966786151987643L
float	Número decimal	Coma flotante de doble precisión	3.1415927
bool	Booleano	Valor booleano verdadero o falso	True o False

Funciones

Las funciones se definen con la palabra clave def, seguida del nombre de la función y sus parámetros. Otra forma de escribir funciones, aunque menos utilizada, es con la palabra clave lambda (que aparece en lenguajes funcionales como Lisp).

```
>>> # Ejemplo de función con def. En este ejemplo salvo que se indique la variable 'y' valdrá 2.
>>> def multiplicacion(x, y):
...     return x * y
...
>>> multiplicacion(10, 3)
30
```

Módulos

Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son mini códigos (la mayoría escritos también en Python) que llaman a los recursos del sistema. Un ejemplo es el módulo Tkinter, que permite crear interfaces gráficas que incluyan botones, cajas de texto, y muchas cosas que vemos habitualmente en el sistema operativo. Otro ejemplo es el módulo que permite manejar el sistema operativo desde el lenguaje. Los módulos se agregan a los códigos escribiendo import seguida del nombre del módulo que queramos usar. En este código se muestra como apagar el ordenador desde Windows.

```
apagar = "shutdown /s"
import os
os.system(apagar)
```

2.1.2.6. Frameworks Web para Python

- **Django**. Es un framework web de código abierto escrito en Pythonque permite construir aplicaciones web más rápido y con menos código.

  Django fue inicialmente desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de World Online, más tarde se liberó bajo licencia BSD.

Django se centra en automatizar todo lo posible y se adhiere al principio DRY (Don't Repeat Yourself). [11].

- **Grok**. Es un framework de aplicaciones web para desarrolladores de Python. Está dirigido a los principiantes y desarrolladores web con mucha experiencia. Grok tiene un énfasis en el desarrollo ágil. Grok es fácil y de gran alcance. [14].

- **Pylons**. Es un framework de aplicaciones web open source que implementa el patrón modelo-vista-controlador. Pylons usa intensivamente el estándar Web Server Gateway Interface (WSGI). [10].

- **TurboGears** . Es un megaframework para desarrollo web, basado en el modelo MVC.

  TurboGears fue creado en el año 2005 por Kevin Dangoor. Es un stack web completo, que abarca desde el Javascript del cliente hasta un mapper relacional-objetos para la base de datos. [34].

- **Web2py.** Es un framework empresarial completo libre y de código abierto para desarrollo ágil de aplicaciones web rápidas, escalables, seguras y portables basadas en bases de datos. Escrito y programable en Python. [27].

2.1.3. Lenguaje de Programación Ruby

Ruby, según Huw Collingbourne en su libro “The book of Ruby” menciona que Ruby es un lenguaje multiplataforma interpretado que tiene muchas características en común con otros lenguajes de “scripting” como Perl y Python. A primera vista cuenta con la sintaxis en “Idioma Inglés" un estilo que se parece un poco al estilo de Pascal. Es totalmente orientado a objetos, y tiene mucho en común con el bisabuelo de 'puro' lenguaje orientado a objetos, Smalltalk. Se ha dicho que las lenguas que más influyeron en el desarrollo de Ruby fueron: Perl, Smalltalk, Eiffel, Ada y Lisp. El lenguaje Ruby

fue creado por Yukihiro Matsumoto (comúnmente conocido como "Matz") y fue lanzado por primera vez en 1995. [1]

2.1.3.1. Historia

Ruby fue concebido el 24 de febrero de 1993 por Yukihiro Matsumoto, quien deseaba crear un nuevo leguaje que sea balanceado tanto en programación funcional con la programación imperativa. Matsumoto decía, “Esperaba un lenguaje de scripting que sea más poderoso que Perl, y más orientado a objetos que Python. Por esto decidí diseñar my propio lenguaje” [24].

El nombre “Ruby” fue decidido durante una sesión online de chat entre Matsumoto y Keiu Ishitsuka en febrero de 1993 [25], antes de que el código haya sido escrito para el lenguaje. Inicialmente se tenían dos nombres propuestos: “Coral” y “Ruby”, el último fue elegido por Matsumoto en el último email a Ishitsuka. Matsumoto comentó que el factor para que escogiera el nombre de “Ruby” fue porque esta era la piedra de nacimiento de uno de sus colegas.

Figura II. 4: Logo del lenguaje de programación Ruby.

Primera publicación

La primera versión pública de Ruby 0.95 se anunció por grupos de noticias nacionales de Japón el 21 de diciembre de 1995. Posteriormente, tres versiones más de Ruby fueron lanzados en dos días. El lanzamiento coincidió con el lanzamiento del Leguaje-Japonés ruby-list lista de correo, que fue la primera lista de correo para el nuevo lenguaje.

Ya presente en esta etapa de desarrollo eran muchas de las características conocidas en las últimas versiones de Ruby, incluyendo diseño orientado a objetos, las clases con herencia, mixins, colección de iteradores, cierres, manejo de excepciones, y basura.[21]

2.1.3.2. Filosofía

Matsumoto ha dicho que Ruby que ha sido diseñado para la productividad y diversión del programador, siguiendo los principios del buen diseño de interfaz de usuario. El hace hincapié en que el diseño de sistemas debe ser enfatizado para los humanos, en lugar de las computadoras: "A menudo la gente, sobre todo ingenieros informáticos, se centran en las máquinas. Ellos piensan, "De esta manera, la máquina funcionará más rápido. De esta manera, la máquina funcionará con mayor eficacia". Ellos se centran en las máquinas. Pero, de hecho, tenemos que centrarnos en los seres humanos, sobre cómo los humanos se preocupan por hacer la programación o explotación de la aplicación de las máquinas. Nosotros somos los amos. Ellos son los esclavos. "

Se dice que Ruby sigue el principio de la mínima sorpresa (principle of least astonishment, POLA), lo que significa que el lenguaje debe comportarse de tal manera que para minimizar la confusión para los usuarios experimentados. Matsumoto dice que su primer objetivo era hacer un lenguaje en el que utilizando sea divertido, reduciendo al mínimo el trabajo del programador y la posible confusión. El no había aplicado el principio de mínima sorpresa para el diseño de Ruby, pero sin embargo, la frase ha pasado a estar estrechamente relacionada con el lenguaje de programación Ruby. La frase ha sido una fuente de sorpresa, ya que los usuarios novatos pueden tomar en el sentido de que las conductas de Ruby traten de asemejarse a los comportamientos familiares de otros idiomas. En mayo de 2005, el debate del grupo de noticias comp.lang.ruby, Matsumoto intentó distanciarse Ruby de Pola, y explicó que debido a que cualquier opción de diseño será extraña a alguien, que utiliza un estándar de personal en la evaluación de sorpresa. Si ese estándar personal se mantiene constante, no habría sorpresas para aquellos familiarizados con la norma.

Matsumoto lo definió de esta manera en una entrevista:

"Todo el mundo tiene un fondo individual. Alguien puede venir de Python, alguien más puede venir de Perl, y que puede ser sorprendido por los diferentes aspectos del lenguaje. Entonces se me acercan y dicen: "Me sorprendió por esta característica del lenguaje, por lo que Ruby viola el principio de mínima sorpresa. Espera. Espera. El principio de mínima sorpresa no es solo para usted. El principio de mínima sorpresa

significa principio por lo menos de mi sorpresa. Y esto significa que el principio de mínima sorpresa después de aprender Ruby muy bien. Por ejemplo, yo era un programador C++ antes de empezar el diseño de Ruby. He programado en C++ en exclusiva para dos o tres años. Y después de dos años de programación C++, todavía me sorprende. [21].

Inicialmente, Matz buscó en otros lenguajes para encontrar la sintaxis ideal. Recordando su búsqueda, dijo, "quería un lenguaje que fuera más poderoso que Perl, y más orientado a objetos que Python" y todo ser visto como un objeto.

En Ruby, todo es un objeto. Se le puede asignar propiedades y acciones a toda información y código. La programación orientada a objetos llama a las propiedades *variables de instancia* y las acciones son conocidas como *métodos*.

En muchos lenguajes, los números y otros tipos primitivos no son objetos. Ruby sigue la influencia del lenguaje Smalltalk pudiendo poner métodos y variables de instancia a todos sus tipos de datos. Esto facilita el uso de Ruby, porque las reglas que se aplican a los objetos son aplicables a todo Ruby. [31]

2.1.3.3. Características

- Completamente orientado a objetos Con herencia, y mixins metaclases
- Tipado dinámico y Duck typing (tipado pato)
- Todo es una expresión (incluso las declaraciones) y todo lo que se ejecuta obligatoriamente (incluso las declaraciones)
- Sucinta y la sintaxis flexible que minimiza el ruido sintáctico y sirve como base para lenguajes específicos de dominio
- La reflexión dinámica y la alteración de los objetos para facilitar la meta programación
- Cierres léxicos, iteradores y generadores, con una sintaxis bloque único
- Notación literal de arrays, hashes, expresiones regulares y los símbolos
- Código de incrustación en las cadenas (interpolación)
- Argumentos por defecto

- Cuatro niveles de ámbito de variable (clase mundial, ejemplo y local), denotado por sellos o sin los mismos
- Recolección de basura
- Primera clase continuaciones
- Estrictas reglas de coerción de booleanos (todo es verdad, excepto false y nil).
- Manejo de excepciones
- Sobrecarga de operadores
- Soporte integrado para los números racionales, números complejos y de la aritmética de precisión arbitraria
- Comportamiento personalizado de despacho (a través de method_missing y const_missing)
- Hilos nativos y las fibras de cooperación
- El apoyo inicial para la codificación de caracteres Unicode y múltiple (aún con errores desde la versión 1.9)
- Plug-in API nativa en C Interactivo de Ruby Shell (un REPL)
- Paquetes centralizados de gestión a través de RubyGems
- Se implementa en todas las principales plataformas
- Gran biblioteca estándar. [21]

2.1.3.4. Semántica

Ruby es orientado a objetos, todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas, (como enteros, booleanos, y "nil"). Toda función es un método. Las variables siempre son referencias a objetos, no los objetos mismos. Ruby soporta herencia con enlace dinámico, mixins y métodos singleton (pertenecientes y definidos por un sola instancia más que definidos por la clase). A pesar de que Ruby no soporta herencia múltiple, la clases pueden importar módulos como mixins. La sintaxis procedural está soportada, pero todos los métodos definidos fuera del ámbito de un objeto son realmente métodos de la clase Object. Como esta clase es padre de todas las demás, los cambios son visibles para todas las clases y objetos.

Ruby ha sido descrito como un lenguaje de programación multiparadigma: permite programación procedural (definiendo funciones y variables fuera de las clases haciéndolas parte del objeto raíz Object), con orientación a objetos, (todo es un objeto) o funcionalmente (tiene funciones anónimas, clausuras o closures, y continuations; todas las sentencias tiene valores, y las funciones devuelven la última evaluación). Soporta introspección, reflexión y meta programación, además de soporte para hilos de ejecución gestionados por el intérprete. Ruby tiene tipado dinámico, y soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre). Ruby no requiere de polimorfismo de funciones al no ser fuertemente tipado (los parámetros pasados a un método pueden ser de distinta clase en cada llamada a dicho método).

De acuerdo con las preguntas frecuentes de Ruby,5 "Si te gusta Perl, te gustará Ruby y su sintaxis. Si te gusta Smalltalk, te gustará Ruby y su semántica. Si te gusta Python, la enorme diferencia de diseño entre Python y Ruby/Perl puede que te convenza o puede que no. [32]

2.1.3.5. Sintaxis

La sintaxis de Ruby es similar a la de Perl o Python. La definición de clases y métodos está definida por palabras clave. Sin embargo, en Perl, las variables no llevan prefijos. Cuando se usa, un prefijo indica el ámbito de las variables. La mayor diferencia con C y Perl es que las palabras clave son usadas para definir bloques de código sin llaves. Los saltos de línea son significativos y son interpretados como el final de una sentencia; el punto y coma tiene el mismo uso. De forma diferente que Python, la Indentación no es significativa.

Una de las diferencias entre Ruby y Python y Perl es que Ruby mantiene todas sus variables de instancia privadas dentro de las clases y solo la expone a través de métodos de acceso (attr_writer, attr_reader, etc). A diferencia de los métodos "getter" y "setter" de otros lenguajes como C++ o Java, los métodos de acceso en Ruby pueden ser escritos con una sola línea de código. Como la invocación de estos métodos no requiere el uso de paréntesis, es trivial cambiar una variable de instancia en una función sin tocar una sola línea de código o refactorizar dicho código. Los descriptores de propiedades de

Python son similares pero tienen una desventaja en el proceso de desarrollo. Si uno comienza en Python usando una instancia de variable expuesta públicamente y después cambia la implementación para usar una instancia de variable privada expuesta a través de un descriptor de propiedades, el código interno de la clase necesitará ser ajustado para usar la variable privada en vez de la propiedad pública. Ruby elimina está decisión de diseño obligando a todas las variables de instancia a ser privadas, pero también proporciona una manera sencilla de declarar métodos set y get. Esto mantiene el principio de que en Ruby no se puede acceder a los miembros internos de una clase desde fuera de esta; en lugar de esto se pasa un mensaje (se invoca un método) a la clase y recibe una respuesta.. [32]

2.1.3.6. La Flexibilidad de Ruby

Ruby es considerado un lenguaje flexible, ya que permite a sus usuarios alterarlo libremente. Las partes esenciales de Ruby pueden ser quitadas o redefinidas a placer. Se puede agregar funcionalidad a partes ya existentes. Ruby intenta no restringir al desarrollador.

Por ejemplo, la suma se realiza con el operador suma (+). Pero si prefieres usar la palabra *sumar*, puedes agregar un método llamado sumar a la clase Numeric que viene incorporada.

```
class Numeric
def sumar(x)
   self.+(x)
end
end
y = 5.sumar 6
# ahora y vale 11
```

Los operadores de Ruby son simples conveniencias sintácticas para los métodos. Los puedes redefinir cómo y cuando quieras. [31]

2.1.3.7. La apariencia visual de Ruby

A pesar de que Ruby utiliza la puntuación muy limitadamente y se prefieren las palabras clave en inglés, se utiliza algo de puntuación para decorar el código. Ruby no necesita declaraciones de variables. Se utilizan convenciones simples para nombrar y determinar el alcance de las mismas.

- var puede ser una variable local.
- @var es una variable de instancia.
- $var es una variable global.

Estos detalles mejoran la legibilidad permitiendo que el desarrollador identifique fácilmente los roles de las variables. También se hace innecesario el uso del molesto self. como prefijo de todos los miembros de instancia.

2.1.4. Frameworks web para Ruby.

- **Ruby on Rails**

Es un entorno de desarrollo web de código abierto que está optimizado para satisfacción de los programadores y de la productividad. Te permite escribir un buen código favoreciendo la convención antes que la configuración. [30]

- **Merb**

Merb es un mini- framework escrito en Ruby orientado a páginas dinámicas simples y, especialmente, subida de archivos al servidor.

Merb nació como una aplicación especializada en subir archivos, algo donde Rails no brilla especialmente. La idea era usar aplicaciones Merb en conjunto con aplicaciones mayores escritas en Rails. Merb, que es la combinación de un framework Ruby, el servidor Web Mongrel y el motor de templates ERB (Mongrel + ERB = “Merb”), toma varias convenciones de Rails (controladores, vistas ERB) y al parecer es excelente para hacer aplicaciones dinámicas simples donde no es necesario el volumen y versatilidad de Rails (por ejemplo, aplicaciones sin bases de datos o utilidades de línea de comandos para facilitar tareas en la oficina). [59]

- **Ramaze**

Es un framework de aplicaciones web creado por Michael Fellinger (también conocido como manveru) para Ruby. La filosofía del mismo puede ser expresada como una mezcla de KISS y POLS, tratando de hacer las cosas simples de forma simple y las cosas complejas posibles. Esto es por supuesto nada nuevo para las personas que conozcan algo de Ruby, pero a menudo es olvidado entre la persecución de funcionalidades y características. Ramaze tiene como objetivo brindar las últimas herramientas, pero es el usuario quien debe aprovecharlo para lograr óptimos resultados a medida. Otro de los objetivos durante el desarrollo de Ramaze era hacer que todas las partes sean modular y por lo tanto reutilizable como sea posible, no solo para proporcionar una comprensión básica después de la primera vista, sino también para que sea posible la reutilización de partes de código. El propósito original de Ramaze era actuar como un framework para construir web-frameworks; esto quedo obsoleto cuando se introdujo Rack, el cual ofrece esta función a un mejor nivel sin tratar de imponer un diseño estructural del framework resultante. [34]

- **Sinatra**

Es un framework para aplicaciones web de software libre y código abierto, y Lenguaje específico del dominio escrita en Ruby. Es una alternativa a otros frameworks para aplicaciones webs como Ruby on Rails, Merb, Nitro, Camping, and Rango. Sinatra depende de Rack interfaz de servidor web. Diseñado y desarrollado por Blake Mizerany, Sinatra es pequeño y flexible. Sinatra no sigue el típico patrón modelo vista controlador que se ve en otros frameworks, como Ruby on Rails. En su lugar, Sinatra se enfoca en la "rápida creación de aplicaciones web en Ruby con el mínimo esfuerzo."1

Algunas destacadas compañías que usan Sinatra son BBC,2 Engine Yard, Heroku, GitHub, and Songbird.3 Heroku provee la mayor parte del apoyo para el desarrollo de Sinatra. [35]

2.1.5. El Patrón de Diseño Modelo Vista Controlador

2.1.5.1. Descripción

El patrón MVC (Modelo Vista Controlador) fue inventado en el contexto de Smalltak con el objetivo de separar la interface gráfica del código que hace que trabaje una aplicación. Posteriormente esta idea afectaría una gran parte del código de Smalltak y sería ampliamente aplicada por otros lenguajes orientados a objetos. En el paradigma MVC las entradas del usuario, los modelos del mundo exterior y la retroalimentación visual son explícitamente separados y manejados por tres tipos de objetos, cada uno especializado para un conjunto de tareas específicas. [20].

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original.

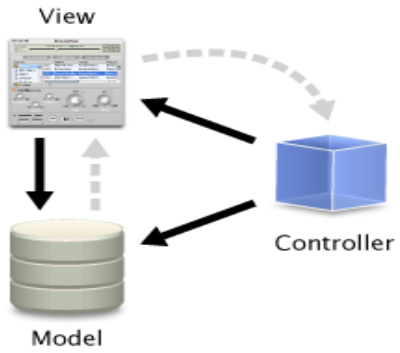


Figura II. 5: Modelo Vista Controlador

2.1.5.2. Historia del MVC

La arquitectura MVC (Model/View/Controller) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk. Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y

sincronizados de los mismos datos. Inicialmente diseñado para Aplicaciones de escritorio cuya idea principal era separar la presentación del modelo de dominio.

En aquel entonces la vista se encargaba del dibujado de mapas de bits, el controlador atendía acciones del mouse y teclado, y el modelo guardaba el estado de la ventana y la información mostrada en ella. Cada vista estaba asociada con un controlador y un modelo en su creación. Cualquier modelo podía ser asociado débilmente a cualquier vista ya que un modelo podía relacionarse con varias vistas. El controlador no se encargaba de manejar la lógica del negocio en la aplicación, su propósito era manejar la interacción del mouse y el teclado, y actualizar la vista acorde, lo cual no era una tarea trivial. [7].

- **Modelo**

Incorpora la capa del dominio y persistencia, es la encargada de guardar los datos en un medio persistente (ya sea una base de datos, un archivo de texto, XML, registro, etc.). En el modelo es donde se hace el levantamiento de todos los objetos que tu sistema debe de utilizar, es el proveedor de tus recursos, es decir, el Modelo Automóviles te tiene que dar objetos Automóvil. Es muy típico que las clases del modelo incorporen otro patrón de diseño como Active Record, ya que así es más fácil guardar en el momento en la base de datos. [20]

- **Vista**

Se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HTML, aunque pueden existir otro tipo de vistas. En la vista solo se deben de hacer operaciones simples, como ifs, ciclos, formateo, etc. [20]

- **Controlador**

Es el que escucha los cambios en la vista y se los envía al modelo, el cual le regresa los datos a la vista, es un ciclo donde cada acción del usuario causa que se inicie de nuevo un nuevo ciclo.

El Controller en cierta forma debe tener un registro de la relación entre ordenes que le pueden llegar y la lógica de negocio que le corresponde (Es como una operadora de teléfono que recibe una petición y une dos líneas).

La forma más sencilla de implementar este patrón es pensando en capas, como regla, los accesos a la base de datos se hacen en el modelo, la vista y el controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía. [20].

2.1.5.3. Ventajas del MVC [20]

Obviamente una separación total entre lógica de negocio y presentación. A esto se le pueden aplicar opciones como el multilenguaje, distintos diseños de presentación, etc. sin alterar la lógica de negocio. La separación de capas como presentación, lógica de negocio, acceso a datos es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y más fácilmente mantenibles, lo que al final resulta en un ahorro de tiempo en desarrollo en posteriores proyectos.

Al existir la separación de vistas, controladores y modelos es más sencillo realizar labores de mejora como:

- Agregar nuevas vistas.
- Agregar nuevas formas de recolectar las ordenes del usuario (interpretar sus modelos mentales).
- Modificar los objetos de negocios bien sea para mejorar el performance o para migrar a otra tecnología.
- Las labores de mantenimiento también se simplifican y se reduce el tiempo necesario para ellas. Las correcciones solo se deben hacer en un solo lugar y no en varios como sucedería si tuviésemos una mezcla de presentación e implementación de la lógica del negocio.
- Las vistas también son susceptibles de modificación sin necesidad de provocar que todo el sistema se paralice. Adicionalmente el patrón MVC propende a la especialización de cada rol del equipo, por tanto en cada liberación de una nueva versión se verán los resultados.

2.1.5.4. Desventajas del MVC [19]

- Tener que ceñirse a una estructura predefinida, lo que a veces puede incrementar la complejidad del sistema. Hay problemas que son más difíciles de resolver respetando el patrón MVC.

- La curva de aprendizaje para los nuevos desarrolladores se estima mayor que la de modelos más simples como Webforms.
- La distribución de componentes obliga a crear y mantener un mayor número de ficheros.

2.1.6. Framework Web

Según Javier J. Gutiérrez, [13]. El concepto framework es que emplea muchos ámbitos del desarrollo de sistemas software, no solo en el ámbito de aplicaciones Web. Podemos encontrar frameworks para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier ámbito que pueda ocurrírsenos.

En general, con el término framework, nos estamos refiriendo a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta

2.1.7. Framework Web Rails para Ruby

RoR se originó con una aplicación de administración de proyectos conocida como Basecamp desarrollada por el danés David Heinemeier Hansson para la compañía 37signals. Originalmente David intentó escribir Rails en PHP pero fracasó, hasta que uso Ruby. RoR es un framework para el desarrollo de aplicaciones web, software libre y basado en el patrón de diseño Modelo Vista Controlador (MVC). [6]

Figura II. 6: Logo Ruby on Rails.

2.1.7.1. Filosofía

RoR da muchas ventajas si lo comparamos con otros frameworks. Una de las principales es que es gratis (open source) y otra que posee toda una comunidad de apoyo. Rails está basado en dos principios principales de desarrollo:

• No te repitas (Don't repeat yourself DRY)
• Convención sobre configuración [6]

No te repitas significa que las definiciones deberían hacerse una sola vez. Dado que Ruby on Rails es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en ActiveRecord, las definiciones de las clases no necesitan especificar los nombres de las columnas; Ruby puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante.

Convención sobre configuración significa que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, si hay una clase Historia en el modelo, la tabla correspondiente de la base de datos es historias, pero si la tabla no sigue la convención (por ejemplo blogposts) debe ser especificada manualmente (set_table_name "blogposts"). Así, cuando se diseña una aplicación partiendo de cero sin una base de datos preexistente, el seguir las convenciones de Rails significa usar menos código (aunque el comportamiento puede ser configurado si el sistema debe ser compatible con un sistema heredado anterior). [32]

Ruby in Rails (RoR) se basa en el desarrollo ágil y RUP (Proceso Unificado Racional ó Rational Unified Process en inglés) por lo que según el contexto no siempre puede ser la mejor opción de desarrollo. Dependiendo del material humano disponible así como las

características del proyecto a desarrollar depende de que tan útil vaya a resultar como opción.

2.1.7.2. Ventajas de RoR respecto a otros frameworks

Rails usa convenciones, paquetes de programación integrados y código predefinido, diseñado para completar y usar inmediatamente sin necesidad de configuración. A diferencia de otros ambientes de programación, como aquellos basados en Java que requieren usar varios frameworks, los cuales deben ser configurados para que funcionen entre sí, para obtener todas las capacidades deseadas [6].

Entre los fundamentos de RoR están los siguientes:

• DRY (Don't Repeat Yourself) : "no te repitas a ti mismo", con esto podemos tener un formulario, y llamarlo las veces que queramos y desde donde queramos, simplemente con una línea código, o tal vez tener una tabla en nuestra base de datos, y manipular a los registros como un objeto y a sus campos como un atributo, sin necesidad de que declaremos nada.

Convención sobre configuración: *class Auto < ActiveRecord::Base end*

Con esa declaración de una clase, mapeamos a una tabla en nuestra base de datos, dicho de otra manera Rails buscara una tabla llamada 'autos', en nuestra base de datos, y porque en plural?, esto es así porque Rails cree conveniente que debe llamarse así (principio de pluralización), aunque este comportamiento se puede desactivar de una manera muy sencilla, y si no la encuentra?, pues nos dará un error.

- Uso de patrones de diseño: Modelo Vista Controlador (MVC)
- Generación de código (helpers): permiten la generación de código xhtml, xml y javascript a partir de código Ruby.
- Menos código, menos errores.
- Test integrado (unitario y funcional). [6]

2.1.7.3. Arquitectura MVC de Rails

Las piezas de la arquitectura Modelo Vista Controlador en Ruby on Rails son las siguientes:

- **Modelo**

En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos.
En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. Por ejemplo, si la clase Imagen tiene una definición has_many:comentarios, y existe una instancia de Imagen llamada a, entonces a.comentarios devolverá un array con todos los objetos Comentario cuya columna imagen_id (en la tabla comentarios) sea igual a a.id.

Las rutinas de validación de datos (p.e. validates_uniqueness_of:checksum) y las rutinas relacionadas con la actualización.
(p.e.after_destroy:borrar_archivo,before_update:actualizar_detalles).
También se especifican e implementan en la clase del modelo.

- **Vista**

En MVC, Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador. Con frecuencia en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML.

Existen en la actualidad muchas maneras de gestionar las vistas. El método que se emplea en Rails por defecto es usar Ruby Empotrado (archivos.rhtml, desde la versión 2.x en adelante de RoR archivos.html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP. También pueden construirse vistas en HTML y XML con Builder o usando el sistema de plantillas Liquid.

Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del controlador y los fragmentos pueden invocarse unos a otros.

- **Controlador**

En MVC, las clases del Controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web.
La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general en la forma http://aplicacion/ejemplo/metodo, que invoca a EjemploController#método, y presenta los datos usando el archivo de plantilla /app/views/ejemplo/método.html.erb, a no ser que el método redirija a algún otro lugar.
Rails también proporciona andamiaje, que puede construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes. [32]

2.1.7.4. Otros módulos

Además, Rails ofrece otros módulos, como Action Mailer (para enviar correo electrónico) o Active Resource que proporciona la infraestructura necesaria para crear de manera sencilla recursos REST, algo por lo que apuesta claramente Rails en sus últimas versiones desplazando así a otros modelos como SOAP y XML-RPC a los que se les daba soporte en versiones anteriores mediante Action Web Service. [32]

2.1.7.5. Gemas

Las gemas son plugins y/o códigos añadidos a nuestros proyectos Ruby on Rails, que nos permiten nuevas funcionalidades como nuevos create, nuevas funciones pre-escritas (como login de usuarios) o nuevas herramientas para el desarrollo como puedan ser Haml y SASS (la primera es una nueva forma de template basada en html pero más

sencilla y potente, y la segunda es igual pero para el caso de las CSS). Para encontrar el listado de gemas disponibles puedes ir a RubyForge. [32]

2.1.7.6. Soporte de servidores Web

Para desarrollo y pruebas, se utiliza Mongrel o WEBrick, incluido con Ruby. Para utilizar Rails en servidores en producción se está extendiendo el uso de Passenger, una suerte de mod_rails para Apache desarrollado en 2008 por la empresa holandesa Phusion. Otras opciones para producción son Nginx, Mongrel, Apache, Lighttpd con FastCGI o alguna combinación de ambos(por ejemplo utilizando Apache como proxy para los procesos Mongrel). Sobre Apache, mod ruby puede mejorar considerablemente el rendimiento, aunque su uso no se recomienda porque no es seguro utilizar múltiples aplicaciones RoR sobre Apache. [32]

2.1.7.7. Soporte de Bases de Datos

Dada que la arquitectura Rails favorece el uso de bases de datos se recomienda usar un SGBDR para almacenamiento de datos. Rails soporta la biblioteca SQLite por defecto. El acceso a la base de datos es totalmente abstracto desde el punto de vista del programador, es decir que es agnóstico a la base de datos, y Rails gestiona los accesos a la base de datos automáticamente (aunque, si se necesita, se pueden hacer consultas directas en SQL) Rails intenta mantener la neutralidad con respecto a la base de datos, la portatibilidad de la aplicación a diferentes sistemas de base de datos y la reutilización de bases de datos preexistentes. Sin embargo, debido a la diferente naturaleza y prestaciones de los SGBDRs el framework no puede garantizar la compatibilidad completa. Se soportan diferentes SGBDRs, incluyendo MySQL, PostgreSQL, SQLite, IBM DB2 y Oracle. [32]

2.1.7.8. Entornos de trabajo (IDE)

Hay muchas alternativas para trabajar con Ruby on Rails, tanto libres y gratuitas como de pago. A continuación se listan las principales:

Aptana: Multiplataforma. Nació como plugins de eclipse para la edición y desarrollo web. Actualmente puedes instalarlo como plugins o autónomo de forma independiente.

Las últimas versiones están muy bien integradas con Ruby on Rails. En este momento Aptana 3 es la versión estable.

Netbeans: Uno de los más usados, libre y totalmente gratuito. Viene muy bien integrado con JRuby (lo cual es algo lógico pues es un programa de Oracle).

TextMate: Sólo para Mac. Es el entorno más usado entre la comunidad Rails. Es pago pero su potencia y forma de trabajo favorece la producción y desarrollo con Ruby on Rails.

Gmate: Un proyecto libre y gratuito para convertir Gedit -el editor de texto de escritorio Gnome de Linux- en un clon muy aproximado de Textmate. Esto se consigue instalando diferentes plugins, temas y retocando algunas opciones. Al ser gratuito es una opción que está cogiendo muchos adeptos hoy en día.[32]

2.1.8. Framework Web Django para Python

Figura II. 7: Logo de Django

2.1.8.1. Visión General

En el Django Book, Adrian Holovaty y Jacob Kaplan-Moss, definen al desarrollo web como un acto entretenido y creativo; en el peor, puede ser una molestia repetitiva y frustrante. Django te permite enfocarte en la parte divertida (el quid de tus aplicaciones Web) al mismo tiempo que mitiga el esfuerzo de las partes repetitivas. De esta forma, provee un alto nivel de abstracción de patrones comunes en el desarrollo Web, atajos para tareas frecuentes de programación y convenciones claras sobre cómo solucionar problemas. Al mismo tiempo, Django intenta no entrometerse, dejándote trabajar fuera del ámbito del framework según sea necesario. [15]

2.1.8.2. Conceptualización de Django

Wikipedia [28], define a Django como un framework de desarrollo web de código abierto, escrito en Python, que cumple en cierta medida el paradigma del Modelo Vista Controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt. En junio del 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio "No lo repitas" (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

2.1.8.3. Características de Django

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos). [28]

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido vía RSS y/o Atom, "páginas planas" que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs. [28]

Otras características de Django son:

- Un mapeador objeto-relacional. ORM propio.
- Es un sistema cuya arquitectura es multiplataforma.
- Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema "middleware" para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan cacheo, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones).
- Consola de Administración de Proyectos Django

2.1.8.4. Arquitectura de Django

La Arquitectura de Django se basa en el patrón Modelo Vista Controlador (MVC). Sin embargo, Django define su estructura con una pequeña diferencia, llamada **modelo vista-template**:

Django sigue el patrón MVC tan al pie de la letra que puede ser llamado un framework MVC. Someramente, la M, V y C se separan en Django de la siguiente manera [15]:

M: La porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django, la cual describiremos en este capítulo.

V: La porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.

C: La porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework mismo siguiendo tu URLconf y llamando a la función apropiada de Python para la URL obtenida.

Debido a que la "C" es manejada por el mismo framework y la parte más emocionante se produce en los modelos, las plantillas y las vistas, Django es conocido como un Framework MTV. En el patrón de diseño MTV,

Los **modelos** de Django son los encargados de mapear los datos en una base de datos relacional. Esta parte es exactamente igual que en el MVC. Estos modelos se definen en ficheros con el nombre de **models.py**.

Las **url** del portal son las encargadas de definir tanto la navegación como las acciones que se puedan realizar (alteración en datos). Es un sistema algo lioso pero muy flexible. Las url se definen en los ficheros llamados **urls.py**

El controlador en Django se define como las **vistas**. Son las responsables de alterar los datos y enviarlos a las plantillas para proceder con su presentación. Las vistas se definen en los ficheros**views.py**

Las plantillas Django son las encargadas de mostrar los datos que se han pedido al usuario. Para ello utilizamos una carpeta llamada **templates** donde añadiremos ficheros de texto plano (HTML, XML, CSV, etc) con la sintaxis de **templates** Django: variables y marcas propias de Django.

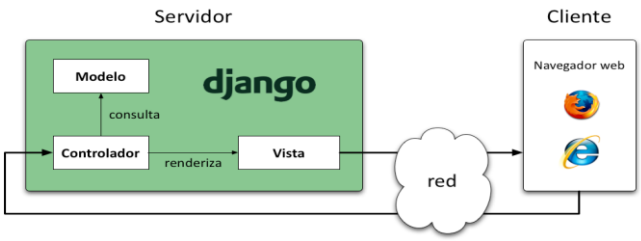


Figura II. 8: Estructura MVC Django

Patrón MVC	vs	Patrón MVT
Modelo	datos	Modelo
Controlador	operaciones	Vista
Vista	interfaz	Plantillas(Templates)

Figura II. 9: Patrón MVC vs Patrón MVT

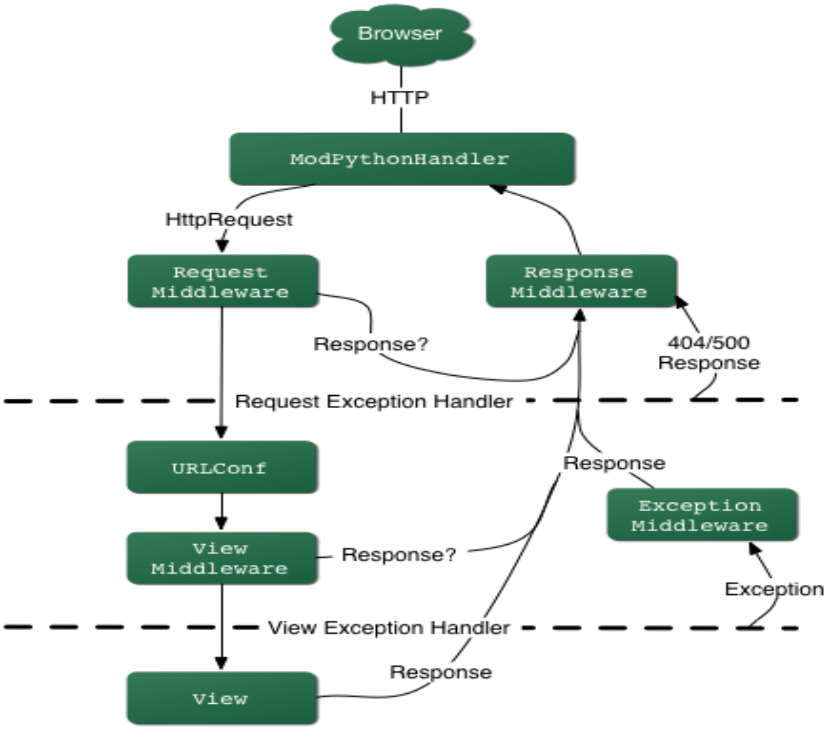


Figura II. 10: Arquitectura Django

2.1.8.5. Soporte de Base de Datos

Django es un framework donde cada uno de los modelos de datos utilizados tiene su correspondencia en una tabla de la Base de Datos, por ello Django está preparado para soportar las más comunes en el mercado simplemente cambiando la variable DATABASE_ENGINE en el settings.py [15].

Django admite cuatro motores de base de datos:

PostgreSQL (http://www.postgresql.org/)

SQLite 3 (http://www.sqlite.org/)

MySQL (http://www.mysql.com/)

Oracle (http://www.oracle.com/)

Además de estas bases de datos que son oficialmente soportadas, existen backends desarrollados por terceros que permitirán usar otro tipo de Base de Datos con Django:

- Sybase SQL Anywhere
- IBM DB2
- Microsoft SQL Server 2005
- Firebird
- ODBC

Se debe tener cuidado porque las versiones de Django y las características ORM soportadas por estos backends no oficiales varían considerablemente. Cualquier duda específica sobre estos backends serán redirigidas al soporte del desarrollador correspondiente. [17].

2.1.8.6. Soporte de Servidores Web

Django incluye un servidor web ligero que se puede usar mientras se está desarrollando un sitio web. Este servidor de desarrollo vigila el código a la espera de cambios y se reinicia automáticamente, ayudándote a hacer algunos cambios rápidos al proyecto sin necesidad de reiniciar nada.

En la etapa de producción, sin embargo, se recomienda Apache 2 con mod_python. Aunque Django soporta la especificación WSGI, por lo que puede correr sobre una gran variedad de servidores como FastCGI o SCGI en Apache u otros servidores (particularmente Lighttpd).

2.1.8.7. Requerimientos

- **Instalar el Lenguaje de Programación Python**

  En primer lugar es lógico que deber tener instalado Python, este lenguaje de programación ya es conocido por mucho de nosotros, recordemos que Django es un Framework para el desarrollo de aplicaciones web ágiles para Python.

Es bueno saber que debemos contar con Python 2.3 o superior, a pesar que se encuentran versiones más actuales con eso es suficiente o compatible.

- **Instalar Apache**

  Si bien dijimos en la presentación que Django cuenta con un servidor propio para realizar las pruebas de las aplicaciones que vamos realizando, es necesario contar con el servidor web Apache, y en especial para poder interpretar las aplicaciones en Python necesitamos también contar con los módulos para Python mod_python, FastCGI o SCGI activado.

  Con respecto a las versiones de estas herramientas Django requiere Apache 2.x y mod_python 3.x

- **Instalar un motor de Base de Datos**

  Este es un tópico bastante general debido a que necesitamos tener instalado nuestro motor de base de datos preferido, entre las opciones podemos escoger PostgreSQL, MySQL, Oracle y SQLite (recordemos que SQLite no requiere un servidor ejecutándose constantemente).

2.1.9. Desarrollo ágil de aplicaciones web

La aparición de aplicaciones y sitios Web proporciona la explotación de otros mercados y servicios antes impensables como el comercio electrónico, la enseñanza virtual, etc., y esto conlleva un importante crecimiento en el desarrollo del software sobre dicha tecnología. Ahora bien, desde el punto de vista de la ingeniería del software es importante dotar de los mecanismos adecuados, para que la realización de este tipo de aplicaciones satisfaga las necesidades tanto de los usuarios como de los clientes que contratan el desarrollo de este tipo de aplicaciones. Pero actualmente no existe una metodología universalmente aceptada, que guíe en el proceso de desarrollo de aplicaciones Web. [4].

2.1.9.1. Objetivos de las Gestión Ágil

La gestión ágil de proyectos tiene como objetivos dar garantías a las cuatro demandas principales de la industria en la que se ha generado [l]:

- Valor
- Reducción del tiempo de desarrollo
- Agilidad
- Fiabilidad.

2.1.9.2. Principales Modelos de Gestión Ágil

Si hubiera que determinar cuál es el origen de la gestión ágil de proyectos, a falta de mejor información, habría que situarlo en las prácticas adoptadas en los 80 por empresas como Honda, 3M, Canon, Fuji, Nec, Xerox, hp o Epson para el desarrollo de nuevos productos [l].

Tras detectar los patrones comunes, y los buenos resultados que ofrecían en empresas de productos tecnológicos, fue la industria del software la primera en seguir su adopción, y en la que además surgieron profesionales que documentaron y propagaron la forma específica que cada uno daba a las prácticas ágiles en sus equipos de trabajo. De esta forma han aparecido en la última década los nombres [l]:

- AD - Agile Database Techniques
- AM - Agile Modeling
- ASD - Adaptive Software Development
- AUP - Agile Unified Process
- Crystal
- FDD - Feature Driven Development
- DSDM - Dynamic Systems Development Method
- Lean Software Development
- Scrum
- TDD - Test-Driven Design
- XBreed
- XP - eXtreme Programming

2.1.10. El Modelo de Gestión SCRUM

2.1.10.1. Origen.

Scrum es una metodología ágil de desarrollo de proyectos que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción por Hirotaka Takeuchi e Ikujijo Nonaka a mediados de los 80.

Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software.

En el desarrollo de software Scrum está considerado como modelo ágil por la Agile Alliance.

2.1.10.2. Introducción al modelo

Scrum es una metodología de desarrollo muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.
Scrum es una metodología ágil, y como tal:

Es un modo de desarrollo de carácter adaptable más que predictivo.

- Orientado a las personas más que a los procesos.
- Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones.

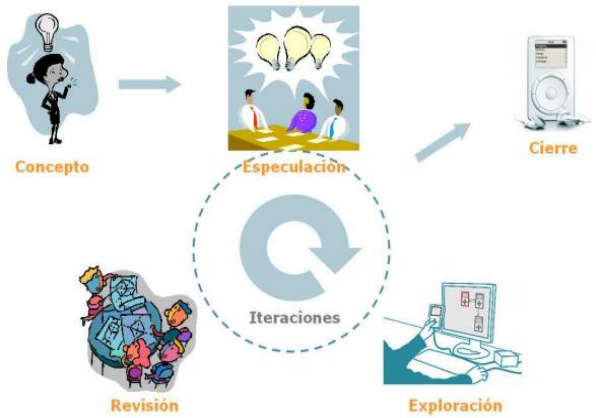


Figura II. 11: Estructura del desarrollo ágil.

Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo breve (normalmente de 30 días).
Cada uno de estos periodos de desarrollo es una iteración que finaliza con la producción de un incremento operativo del producto.

Estas iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución a través de reuniones breves diarias en las que todo el equipo revisa el trabajo realizado el día anterior y el previsto para el día siguiente.

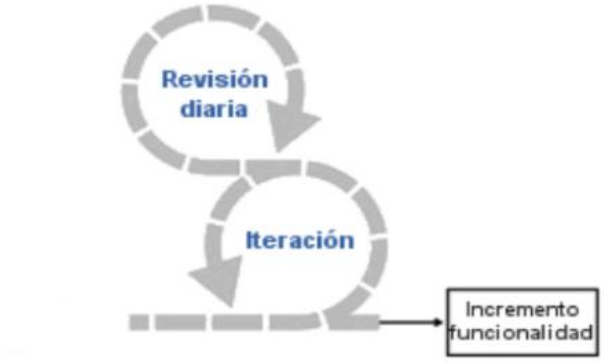


Figura II. 12: Estructura central de Scrum

- **Control de la evolución del proyecto**

Scrum controla de forma empírica y adaptable la evolución del proyecto, empleando las siguientes prácticas de la gestión ágil:

- **Revisión de las Iteraciones**

Al finalizar cada iteración (normalmente 30 días) se lleva a cabo una revisión con todas las personas implicadas en el proyecto. Este es el periodo máximo que se tarda en reconducir una desviación en el proyecto o en las circunstancias del producto

- **Desarrollo incremental**

Durante el proyecto, las personas implicadas no trabajan con diseños o abstracciones.

El desarrollo incremental implica que al final de cada iteración se dispone de una parte del producto operativa que se puede inspeccionar y evaluar.

- **Desarrollo evolutivo**

Los modelos de gestión ágil se emplean para trabajar en entornos de incertidumbre e inestabilidad de requisitos.

Intentar predecir en las fases iníciales cómo será el producto final, y sobre dicha predicción desarrollar el diseño y la arquitectura del producto no es realista, porque las circunstancias obligarán a remodelarlo muchas veces.

Para qué predecir los estados finales de la arquitectura o del diseño si van a estar cambiando. En Scrum se toma a la inestabilidad como una premisa, y se adoptan técnicas de trabajo para permitir esa evolución sin degradar la calidad de la arquitectura que se irá generando durante el desarrollo.

El desarrollo Scrum va generando el diseño y la arquitectura final de forma evolutiva durante todo el proyecto. No los considera como productos que deban realizarse en la primera "fase" del proyecto. (El desarrollo ágil no es un desarrollo en fases)

- **Auto-organización**

Durante el desarrollo de un proyecto son muchos los factores impredecibles que surgen en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al gestor de proyectos.

En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

- **Colaboración**

Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. Ésta es necesaria, porque para que funcione la auto-organización como un control eficaz cada miembro del equipo debe colaborar de forma abierta con los demás, según sus capacidades y no según su rol o su puesto.

- **Visión general del proceso**

Scrum denomina “sprint” a cada iteración de desarrollo y recomienda realizarlas con duraciones de 30 días. El sprint es por tanto el núcleo central que proporciona la base de desarrollo iterativo e incremental.

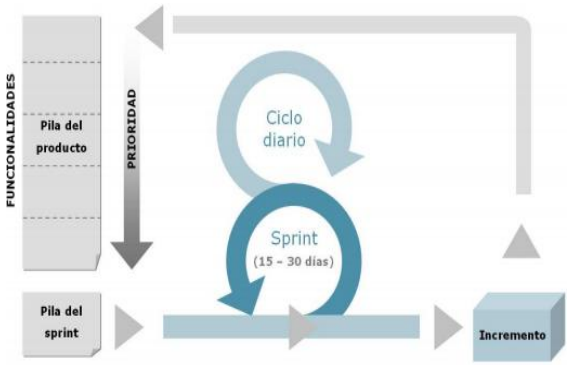


Figura II. 13: Sprint del Scrum

Los elementos que conforman el desarrollo Scrum son:

- **Las reuniones**

  - Planificación de sprint: Jornada de trabajo previa al inicio de cada sprint en la que se determina cuál va a ser el trabajo y los objetivos que se deben cumplir en esa iteración.
  - Reunión diaria: Breve revisión del equipo del trabajo realizado hasta la fecha y la previsión para el día siguiente.
  - Revisión de sprint: Análisis y revisión del incremento generado.

- **Los elementos**

  - Pila del producto: lista de requisitos de usuario que se origina con la visión inicial del producto y va creciendo y evolucionando durante el desarrollo.
  - Pila del sprint: Lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.
  - Incremento: Resultado de cada sprint

CAPÍTULO III:

ANALISIS COMPARATIVO DE LOS FRAMEWORKS DJANGO DE PYTHON FRENTE A RUBY ON RAILS

3.1. Análisis de los Frameworks

A continuación se detalla el análisis de los frameworks definidos anteriormente.

3.1.1. Django de Python

Es un framework de código abierto, el mismo que se identifica como un paquete de software, que sirve de base para otros proyectos de software en el desarrollo de aplicaciones web. Provee una estructura y metodología de trabajo propia. Este framework fomenta el desarrollo rápido, limpio y fomenta el diseño pragmático, el cual sigue el principio DRY (Don't Repeat Yourself). Provee una estructura de trabajo basada en el patrón Model Template View, así como también está en la capacidad de mapear los objetos instanciados en Python con la base de datos a través de su propio ORM.

Esta tecnología permite el diseño amigable de URL's para los buscadores acoplándose íntegramente con un completo sistema de plantillas que es de mucha utilidad para los diseñadores y a su vez genera una interfaz de administración automática. Django también es capaz de gestionar formularios, sesiones de usuarios, autenticación, cache, almacenamiento, sitemaps, internacionalización, etc. [16]

3.1.1.1. Historia.

Django nació naturalmente de aplicaciones de la vida real escritas por un equipo de desarrolladores Web en Lawrence, Kansas. Nació en el otoño boreal de 2003, cuando los programadores Web del diario Lawrence Journal-World, Adrian Holovaty y Simon Willison, comenzaron a usar Python para crear sus aplicaciones. El equipo de The World Online, responsable de la producción y mantenimiento de varios sitios locales de noticias, prosperaba en un entorno de desarrollo dictado por las fechas límite del periodismo. Para los sitios incluidos LJWorld.com, Lawrence.com y KUsports.com, los periodistas (y los directivos) exigían que se agregaran nuevas características y que aplicaciones enteras se crearan a una velocidad vertiginosa, a menudo con sólo días u horas de preaviso. Es así que Adrian y Simon desarrollaron por necesidad un framework de desarrollo Web que les ahorrara tiempo, era la única forma en que podían crear aplicaciones mantenibles en tan poco tiempo.

En el verano boreal de 2005, luego de haber desarrollado este framework hasta el punto en que estaba haciendo funcionar la mayoría de los sitios World Online, el equipo de World Online, que ahora incluía a Jacob Kaplan-Moss, decidió liberar el framework como software de código abierto. Lo liberaron en julio de 2005 y lo llamaron Django, por el guitarrista de jazz Django Reinhardt.

A pesar de que Django ahora es un proyecto de código abierto con colaboradores por todo el mundo, los desarrolladores originales de World Online todavía aportan una guía centralizada para el crecimiento del framework, y World Online colabora con otros aspectos importantes tales como tiempo de trabajo, materiales de marketing, y hosting/ancho de banda para el Web site del framework

3.1.1.2. Licenciamiento

Django fue liberado al público bajo una licencia BSD en julio de 2005. La licencia BSD [8], hasta el punto que en muchas ocasiones se refieren a licencias permisivas como \licencias tipo BSD". La licencia BSD (Berkeley Software Distribución), solo establece la obligación de dar crédito a los autores, mientras que permite tanto la redistribución binaria y la del código fuente, aunque no obliga a ninguna de las dos en ningún caso.

Así mismo se da permiso para realizar modificaciones y ser integrada con otros programas casi sin restricciones.

3.1.2. Ruby on Rails

Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la meta programación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby. [32]

3.1.2.1. Historia

David Heinemeier Hansson extrae de Ruby on Rails de su trabajo en Basecamp, una herramienta de gestión de proyectos por 37signals (ahora una compañía de aplicaciones web). Hansson lanzó por primera vez Rails como código abierto en julio de 2004, pero no compartió los derechos del proyecto hasta febrero de 2005. En agosto de 2006, el framework alcanzó un hito cuando Apple anunció que el envío de Ruby on Rails en Mac OS X 10.5 "Leopard", que fue lanzado en octubre de 2007.

La versión de Rails 2.3 fue lanzado el 15 de marzo de 2009. Principales novedades en Rails incluyen plantillas, motores, Porta y las formas anidadas del modelo. Plantillas permiten a los desarrolladores generar un esqueleto de la aplicación de gemas y configuraciones. Motores que piezas de una aplicación de reutilización completa con rutas, caminos de vista y modelos.

El 23 de diciembre de 2008, Merb, otro framework de aplicaciones web se puso en marcha, y Ruby on Rails anunció que iba a trabajar con el proyecto Merb para llevar "las mejores ideas de Merb" en Rails 3, terminando la "duplicación innecesaria" a través de ambas comunidades. Merb se fusionó con Rails como parte del lanzamiento de Rails 3.0.

Rails 3.1 fue lanzado el 31 de agosto de 2011, con las migraciones de bases de datos reversibles, así como la canalización de los activos. [32]

3.1.2.2. Licenciamiento

Rails está liberado bajo la licencia MIT. La licencia MIT es una de tantas licencias de software que ha empleado el Instituto Tecnológico de Massachusetts (MIT, Massachusetts Institute of Technology) a lo largo de su historia, y quizás debería llamarse más correctamente licencia X11, ya que es la licencia que llevaba este software de muestra de la información de manera gráfica X Window System originario del MIT en los años 1980. Pero ya sea como MIT o X11, su texto es idéntico. [32]

El texto de la licencia no tiene copyright, lo que permite su modificación. No obstante esto, puede no ser recomendable e incluso muchas veces dentro del movimiento del software de código abierto desaconsejan el uso de este texto para una licencia, a no ser que se indique que es una modificación, y no la versión original.

La licencia MIT es muy parecida a la licencia BSD en cuanto a efectos. [32]

3.2. Determinación de parámetros de productividad para la comparación

Para la comparación de los Frameworks Django y Ruby on Rails, en su ámbito de productividad se ha tomado como referencia el manual de FIM – productividad (Fondo para la Investigación y Mejoramiento de la productividad) adaptado al desarrollo de software y los parámetros seleccionados de éste en la investigación **CALIDAD SISTÉMICA Y PRODUCTIVIDAD EN EL DESARROLLO DE SISTEMAS DE SOFTWARE** [5] desarrollada por Edumilis M. MÉNDEZ, María A. PÉREZ, Anna C. GRIMÁN, Luis E. MENDOZA del Departamento de Procesos y Sistemas de la Universidad Simón Bolívar de Venezuela. Esta investigación se ha seleccionado como guía principal para analizar los parámetros de evaluación de la productividad.

Y es así que se ha considerado una serie de factores, siendo cada índice seleccionado es importante para una mejor determinación de la productividad, se ha dividido en 7 parámetros cada uno con sus sub-parámetros para determinar por separado sus

potencialidades y debilidades, Se describe a continuación los parámetros de comparación para la determinación del mejor Framework.

Tabla III. I: Parámetros e indicadores a valorar.

CLASIFICACION	PARAMETROS	INDICADORES
Patrón de Diseño Modelo Vista Controlador	**Modelo**	• Soporte para múltiples bases de datos
		• Manipulación con la base de datos
		• Desempeño con la base de datos
		• Mapeador de Objeto Relacional ORM
	Vista	• Uso de plantillas
		• Adaptación con hojas de estilo
		• Fusión de código y diseño
	Controlador	• Generación automática de formularios
		• Validación de formularios
		• Manejo visual de componentes de formulario
Principio DRY Don`t repeat yourself	**Reutilización**	• Herencia de clase base
		• Herencia de plantillas
		• Personalización de código heredado
		• Tamaño de la aplicación
Seguridad	**Seguridad de Aplicación**	• Variable uso de sesiones
		• Manejo de cookies
		• Encriptación de datos
		• Validación de datos
Producto	**Madurez de producto**	• Especificaciones
		• Licenciamiento

CLASIFICACION	PARAMETROS	INDICADORES
		• Costo
	Instalación	• Contenedor de aplicaciones
		• Entorno de desarrollo
		• Tiempo de instalación

3.2.1. Manejo del Patrón MVC

3.2.1.1.Indicador 1: Modelo/Acceso a Datos.

Describe la capacidad del framework y el análisis de los aspectos que brinda cada tecnología para interactuar con un motor de base de datos y la capacidad de manipulación de los datos.

Tabla III. II: Descripción Indicador 1, Modelo/Acceso a datos

		Descripción
Indicador 1	**Modelo/Acceso a Datos.**	Describe la capacidad del framework y el análisis de los aspectos que brinda cada tecnología para interactuar con un motor de base de datos y la capacidad de manipulación de los datos
[índice 1.1]	Soporte para múltiples bases de datos	Describe el comportamiento del framework ante los distintos motores de base de datos.
[índice 1.2]	Manipulación con la base de datos	Describe la manipulación de la base de datos y el control de los diferentes elementos que la componen.
[índice 1.3]	Desempeño con la base de datos	Describe el desempeño del framework utilizando un motor de base de datos.
[índice 1.4]	Mapeador del objeto	Describe la efectividad en la utilización

	relacional ORM	del lenguaje hacia el modelo relacional.

3.1.2.2. Indicador 2: Vista

Describe las potencialidades que presentan los framework para el desarrollo del entorno visual de las aplicaciones web.

Tabla III. III: Descripción Indicador 2, Vista

		Descripción
Indicador 2	**Vista**	Describe las potencialidades que presentan los framework para el desarrollo del entorno visual de las aplicaciones web.
[índice 2.1]	Uso de plantillas	Describe la agilidad con que los frameworks permiten utilizar un sistema basado en plantillas.
[índice 2.2]	Adaptación con hojas de estilo	Describe el fácil acoplamiento del framework con hojas de estilo en cascada.
[índice 2.3]	Fusión de código y diseño	Describe el fácil engranaje del código de la aplicación y el diseño de la vista.

3.2.3.3. Indicador 3: Controlador

Describe el desarrollo de los formularios y la capacidad que presenta cada framework para implementarlos.

Tabla III. IV: Descripción Indicador 3, Controlador

		Descripción
Indicador 3	**Controlador**	Describe el desarrollo de los formularios y la capacidad que presenta cada framework para implementarlos.

[índice 3.1]	Generación automática de formularios	Describe la capacidad que presenta la tecnología en la generación automática de formularios.
[índice 3.2]	Validación de formularios	Las ventajas de la validación de datos errados en un formulario
[índice 3.3]	Manipulación visual de componentes de formulario.	Es la destreza del framework en la manipulación visual de los componentes de un formulario

3.2.4. Principio DRY (Don`t Repeat Yourself)

3.2.4.1. Indicador 4: Reutilización

Describe la capacidad de utilizar el código que se encuentra ya implementado.

Tabla III. V: Descripción Indicador 4, Reutilización

		Descripción
Indicador 4	**Reutilización**	Describe la capacidad de utilizar el código que se encuentra ya implementado.
[índice 4.1]	Herencia de clase base	La capacidad de heredar clases hijas desde su clase base.
[índice 4.2]	Herencia de plantillas.	La posibilidad de heredar plantillas desde una plantilla base.
[índice 4.3]	Personalización de código heredado.	La facilidad en la implementación del código heredado.
[índice 4.4]	Tamaño de la aplicación	Describe el tamaño en bits que posee una aplicación web completa.

3.2.5. Seguridad

3.2.5.1.Indicador 5: Seguridad de aplicación

Describe la facilidad que proporciona el framework para generar los principales indicadores de seguridad a nivel de aplicación tales como sesiones y validaciones de datos.

Tabla III. VI: Descripción Indicador 5, Seguridad de aplicación

		Descripción
Indicador 5	**Seguridad de aplicación**	Describe la facilidad que proporciona el framework para generar los principales indicadores de seguridad a nivel de aplicación tales como sesiones y validaciones de datos.
[índice 5.1]	Variable uso de sesiones	Describe la administración en la creación de sesiones y la facilidad en su uso y gestión del framework.
[índice 5.2]	Manejo de cookies	Describe el manejo de cookies al momento de un inicio de sesión
[índice 5.3]	Encriptación de datos	Describe la eficacia del método que utiliza el framework para la encriptación
[índice 5.4]	Validación de datos	Describe los distintos métodos de validación para el ingreso correcto de datos.

3.2.6. Producto

3.2.6.1.Indicador 6: Madurez de producto

Describe la madurez del producto como herramienta de desarrollo de aplicaciones web de una manera sencilla.

Tabla III. VII: Descripción Indicador 6, Madurez del producto

Descripción

Indicador 6	**Madurez del producto**	Describe la madurez del producto como herramienta de desarrollo de aplicaciones web de una manera sencilla.
[índice 6.1]	Especificaciones	Establece las versiones que presenta el framework al momento de desarrollo, describe así la compatibilidad con las demás herramientas.
[índice 6.2]	Licenciamiento	Describe el nivel de licenciamiento que poseen las herramientas, influyen al momento de producción
[índice 6.3]	Costo	Describe que tecnología brinda una ventaja respecto al costo de desarrollo.

3.2.6.2. Indicador 7: Instalación

Describe las facilidades que presentan los frameworks al momento de realizar el proceso de instalación.

Tabla III. VIII: Descripción Indicador 7, Instalación

		Descripción
Indicador 7	**Instalación**	Describe las facilidades que presentan los frameworks al momento de realizar el proceso de instalación.
[índice 7.1]	Contenedor de aplicaciones	Describe el servidor de aplicaciones en el cual se ejecutara la aplicación.
[índice 7.2]	Entorno de desarrollo	Describe las facilidades que presenta el framework para adaptarse a un fácil entorno de desarrollo.
[índice 7.3]	Tiempo de instalación	Describe la complejidad del proceso de instalación del framework considerando el

		tiempo.

3.3. Descripción de los módulos de pruebas

Los módulos de prueba son escenarios prototipos que ayudan a verificar y obtener datos los cuales nos permiten identificar con claridad que framework presenta la mayor productividad.

Los módulos que se desarrollarán son implementados en los 2 Frameworks definidos. Estos módulos son desarrollados con las características particulares y arquitectura que presenta cada uno de los frameworks, es decir aplicando el patrón de diseño que los identifica.

En cada Framework se probara los mismos escenarios para la comparación de los parámetros de productividad definidos con anterioridad y se obtendrá los resultados mediante la experiencia en el desarrollo de cada uno de estos módulos de prueba. . La mayoría de resultados se los hace midiendo el tiempo empleado en las actividades de desarrollo.

3.3.4. Módulo 1

Es el módulo que será desarrollador para probar los siguientes parámetros de:

- Manejo del patrón MVC
- Principio DRY

3.3.4.1.Manejo del patrón MVC

Modelo: Se refiere a la conectividad a la base de datos, el uso de controladores y configuración de archivos de conexión. En este sub-módulo se creará una página que despliegue un listado de instituciones las cuales pueden ser asignadas a un determinado docente. Esto se lo realizara mediante las respectivas consultas al la base de datos en PostgresSQL.

Vista: Determina la capa de la interfaz de la aplicación y la manera en la que se muestran los datos. En este sub módulo se desarrollará una página del sistema de

control docente de la DECH, la cual consta con varios componentes de la interfaz de usuario.

Controlador: Este sub módulo lo representa la creación de una formulario para el ingreso de un nuevo docente y la validación de sus campos vacios.

3.3.4.2. Principio DRY

Este principio permite conocer las ventajas que presentan los frameworks para la reutilización en el código. Estos parámetros serán comparados utilizando los módulos anteriormente desarrollados enfocados en el DRY.

3.3.5. Módulo 2

En este módulo implementara para probar los parámetros de seguridad.

En este módulo se desarrollara un login para la autenticación de los usuarios en una aplicación web, mediante el uso de sesiones, protección de los datos que viajan en la web como las contraseñas, el uso de cookies y la validación de usuarios existentes.

3.3.6. Módulo 3

En este módulo se probaran los parámetros del Producto, en el cual se realizan cuadros comparativos que reflejan las características que presenta cada framework al momento de desarrollar aplicaciones web. Esto nos permitirá distinguir las ventajas en una producción final.

3.4. Desarrollo de los módulos de pruebas

Los distintos módulos dentro de la aplicación web prototipos son realizados utilizando los diferentes lenguajes de programación con las bondades que presentan sus respectivos frameworks, como lo son Python con Django y Ruby on Rails. Ambos siguiendo la arquitectura MVT (Modelo Vista Template) y MVC (Modelo Vista Controlador) respectivamente, que son patrones que definen la arquitectura que identifica a cada uno de los frameworks.

Las aplicaciones web prototipo serán desarrolladas en cada uno de los Frameworks, los mismos que serán implementados en las mismas condiciones y realizaran idénticas acciones, así como funciones, para probar los escenarios planteados.

3.4.4. Desarrollo de los módulos de prueba con la tecnología Django Framework

El desarrollo de la aplicación web mediante la utilización de Django sigue claramente una estructura bien definida, separando a diferentes aplicaciones que pueden existir en un proyecto web, es decir un proyecto web puede tener varias aplicaciones, lo cual nos facilita la proyección de crecimiento dentro de la creación de futuras funcionalidades.

Sigue la arquitectura MVT en lo cual se determinan claramente a los modelos, las vistas y el sistema de plantillas.

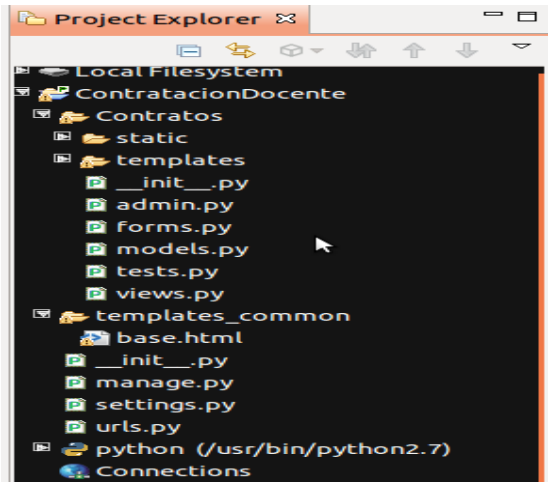


Figura III. 1: Estructura detallada de Django MVT

3.4.4.1. Módulo 1

MODELO

Dentro de este módulo utilizaremos los driver y métodos que nos permitirán acceder a la base de datos y la manipulación de los datos.

Para lo cual crearemos un nuevo proyecto en Django para posteriormente editar los parámetros de conexión en el archivo de configuración settinngs.py

En el archivo de configuración realizaremos la edición de la variables para la conexión a un motor de base de datos determinado, en este caso la conexión se la realiza ha PostgreSQL.

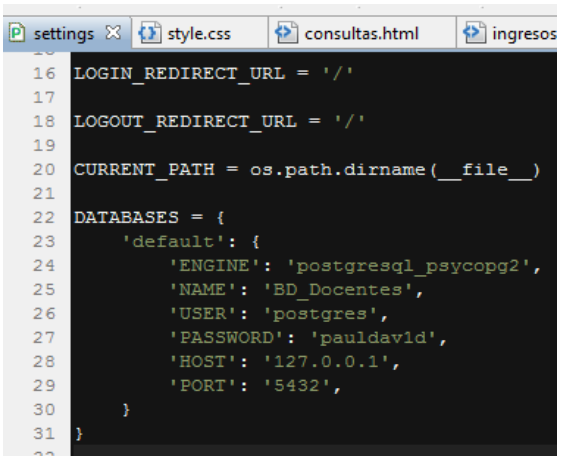


Figura III. 2Configuración de la conexión a la BD en settings.py

Los variables de conexión se configuran de la siguiente manera:

ENGINE: Se especifica el controlador del motor de base de datos a utilizar, en este caso puede ser: SQLite, PostgreSQL, MySQL y Oracle.

NAME: Indica el nombre de la base de datos a la cual nos vamos a conectar.

USER: Especifica el usuario con acceso a la base de datos antes indicada.

PASSWORD: Se refiere a la contraseña de ingreso para el usuario.

HOST: Indica la dirección del host donde se aloja el servidor de base de datos.

PORT: Se direcciona hacia el puerto de escucha del motor de base de datos.

Después de configurar las variables de conexión se procedemos a la creación de los modelos existentes en la aplicación que serán las tablas de nuestra base de datos.

models

```
from django.db import models
from django.contrib.auth.models import User

# Create your models here.

class Institucion(models.Model):
    nombre = models.CharField(max_length = 50)
    ubicacion = models.CharField(max_length = 50)

    def __str__(self):
        return self.nombre

class Docente(models.Model):
    nombre = models.CharField(max_length = 30)
    apellido = models.CharField(max_length = 30)
    cedula = models.CharField(max_length = 10)
    direccion = models.CharField(max_length = 50)
    telefono = models.CharField(max_length = 9)
    email = models.EmailField()
    instituciones = models.ManyToManyField(Institucion)

    def __str__(self):
        return self.nombre

```

Figura III. 3: Creacion de los modelos.

Una vez configurado settings.py podemos validar, generar el código SQL para la creación de los elementos de la base de datos y sincronizar al la aplicación hacia la base de datos.

Console | Problems | Contratos

```
Paul@PC001 ~/Documents/Django Projects/ContratacionDocente
$ python manage.py validate
0 errors found

Paul@PC001 ~/Documents/Django Projects/ContratacionDocente
$ python manage.py syncdb
Creating tables ...
Installing custom SQL ...
Installing indexes ...
No fixtures found.

Paul@PC001 ~/Documents/Django Projects/ContratacionDocente
$
```

Figura III. 4: Validación y sincronización a la base de datos.

Los comandos que permiten la validación, generación del código SQL y la creación de las tablas en la base de datos son:

validate: Valida algún error en los modelos especificados en models.py

sqlall *[Nombre de la aplicación]*: Genera el código SQL para la creación de la base de datos.

syncdb: Permite la sincronización de la aplicación hacia la base de datos creando las tablas necesarias que utilizará la aplicación.

Django permite la interacción hacia la base de datos utilizando el propio lenguaje de Python el cual es mapeado hacia el modelo relacional de datos, a través de su propio

ORM. Esto nos permite consultar a nuestra base de datos de una manera sencilla, ya que permite ejecutar código SQL directamente para operaciones especialmente complejas.

```
Paul@PC001 ~/Documents/Django Projects/ContratacionDocente
$ python manage.py shell
Python 2.7.2 (default, Jun 12 2011, 14:24:46) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from ContratacionDocente.Contratos.models import Institucion
>>> I = Institucion ( nombre = 'ESPOCH', ubicacion = 'Panamericana Sur')
>>> I.save()
>>> I2 = Institucion ( nombre = 'UNACH', ubicacion = 'Via a Guano')
>>> I2.save()
>>>
```

Figura III. 5: Creación de nuevas instituciones utilizando el ORM.

```
>>>
>>> lista_instituciones = Institucion.objects.all()
>>> lista_instituciones
[<Institucion: ESPOCH>, <Institucion: UNACH>]
>>>
```

Figura III. 6: Consulta de instituciones a la base de datos a través del ORM

VISTA

EL propósito de desarrollo de este sub módulo es la comprobación que posee el framework para la fácil realización de la interfaz de usuario, verificando así el uso de plantillas HTML y su adaptación a la utilización de hojas de estilo en cascada.

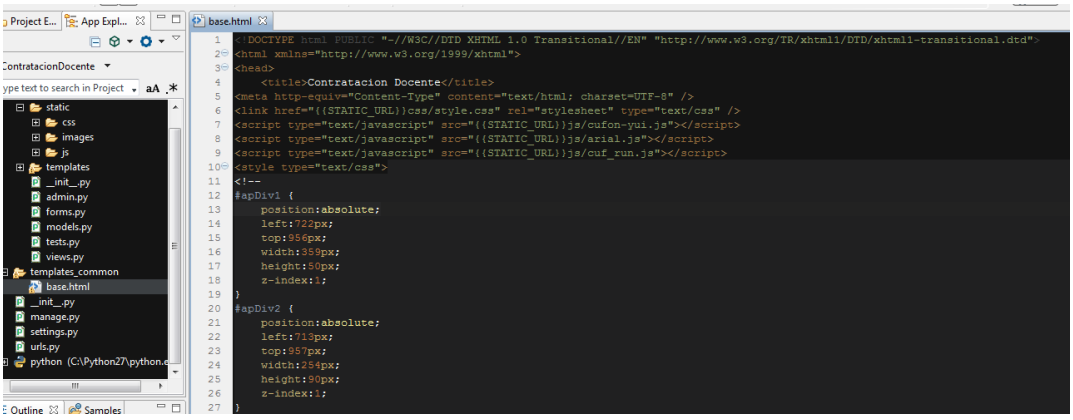


Figura III. 7: Creación del documento base HTML.

Todas las plantillas para la creación de la aplicación web se generan a partir de un documento HTML base, el cual hace uso de estilos.

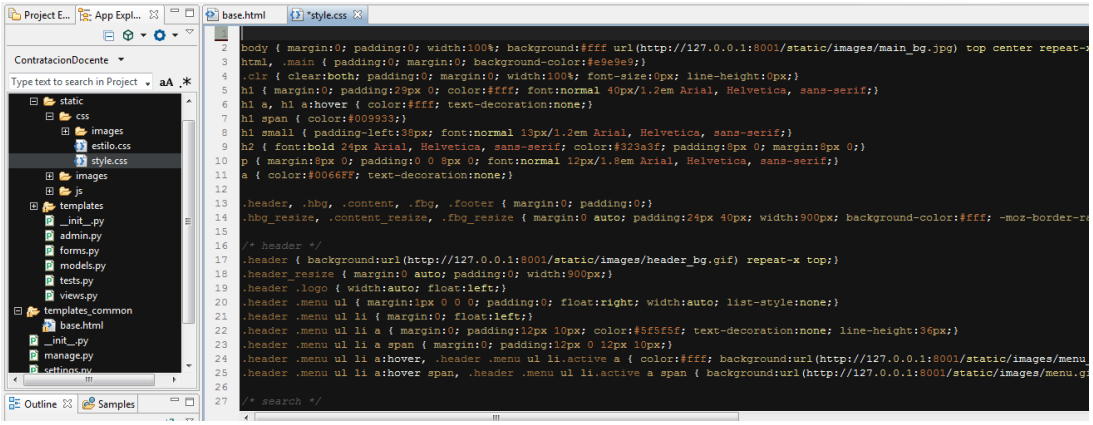


Figura III. 8: Uso de hojas de estilo CSS con Django

La fácil aplicación y creación de la demás plantillas se debe al completo uso de un sistema completo de plantillas que posee Django, y es así que los demás documentos HTML heredar sus propiedades a partir de la plantilla base, únicamente con la utilización de etiquetas.

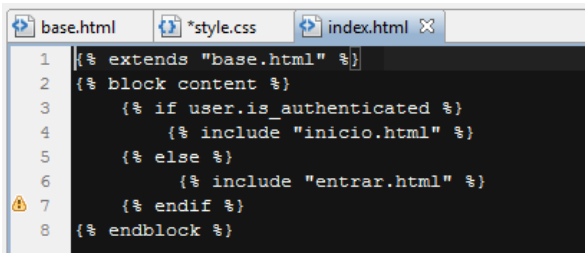


Figura III. 9: Pagina index.html con el uso de etiquetas.

Esto permite un ágil desarrollo para las demás interfaces de usuario, proporcionando así ventajas en la creación de las vistas.

Figura III. 10: Vista de la pantalla principal index.html

CONTROLADOR

El sub módulo controlador se desarrollado mediante la creación de un formulario para el ingreso de un nuevo docente, de manera que se pueda notar las ventajas en la creación automática del formulario a partir del modelo y la validación del formulario en caso que sus datos sea erróneos.

se.html | *style.css | index.html | *forms

```
from django import forms
from models import Docente, Institucion
from django.contrib.auth.models import User


class DocenteForm(forms.ModelForm):
    class Meta:
        model = Docente
```

Figura III. 11: Creación del formulario docente a partir del modelo.

Django permite la creación y generación automática de los formularios a partir de los atributos creados anteriormente en los modelos.

base.html | *style.css | index.html | *forms | add_doc.html

```
1  {% extends "base.html" %}
2      {% block content %}
3        <h2>Ingreso de Docentes</h2>
4        <form class="ingresar" action="" method="POST">
5          <table>
6              {{ form.as_table }}
7          </table>
8          <p><input type="submit" value="Guardar"></p>
9        </form>
10     {% endblock %}
```

Figura III. 12: Plantilla para ingresar un nuevo docente.

La creación del formulario será generado a partir del modelo Docente, a través de

{{ form.as_table}}

Figura III. 13: Formulario generado automáticamente a partir del modelo.

Figura III. 14: Validación en caso de que los campos estén vacios

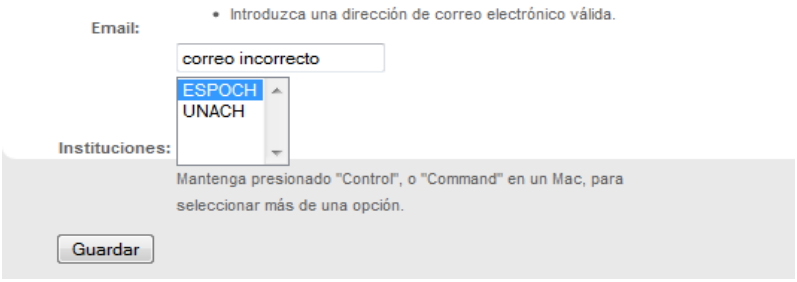


Figura III. 15: Validación en caso que los datos ingresados sean incorrectos.

3.4.4.2. Módulo 2

El desarrollo del módulo de autenticación utilizaremos el sistema de inicio de sesión propio de Django, puesto que al instalar la aplicación [redacted] tenemos la posibilidad de utilizar las vistas propias del framework y un propio sistema de autenticación.

```
url(r'^logout_ok/$',sesion_end),
url(r'^logout/$', logout_user),
url(r'^account/login/$', 'django.contrib.auth.views.login',{'template_name':'login.html'}),
```

Figura III. 16: Uso de las funciones propias de autenticación de Django

El sistema de autenticacion de Django permite utilizar un completo sitio de administracion de grupos y usuarios en el cual permite gestionar el uso de permisos, sesiones, y el uso de cookies.

```
{% block content %}
{% if errors%}
    <p class="error">Error al loguear, los datos no coinciden...</p>
{% endif %}
    <div id="div_login_logo">
        <div id="div_form">
            <form action="{% url django.contrib.auth.views.login %}" method="post">
                {% csrf_token %}
              <fieldset style="border: 1px solid #C0C0C0">
                <legend><h3>Autenticacion</h3></legend>
                <label for="username">Usuario:</label>
                <br/>
                <input type="text" name="username" value="" id="username">
                <br/>
                <label for="password">Contraseña:</label>
                <br/>
                <input type="password" name="password" value="" id="password">
                <br/>
                <input type="submit" value="Ingresar" />
                <input type="hidden" name="next" value="{{ request.get_full_path }}">
              </fieldset>
            </form>
        </div>
    </div>
{% endblock %}
```

Figura III. 17: Implementación del formulario de autenticación de usuarios.

Figura III. 18: Sitio de administración de Django

Figura III. 19: Autenticación del Sistema prototipo de pruebas.

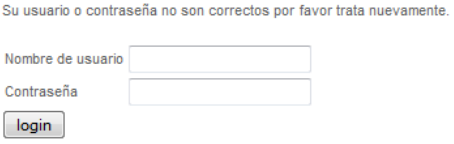


Figura III. 20: Validación de usuario incorrecto.

3.4.4.3. Módulo 3

PRODUCTO

Madurez del producto: Las versiones existentes de la tecnología Django son:

- Django 0.9
- Django 1.0
- Django 1.1
- Django 1.2
- Django 1.3 y 1.3.1
- Se está trabajando en la versión 1.4

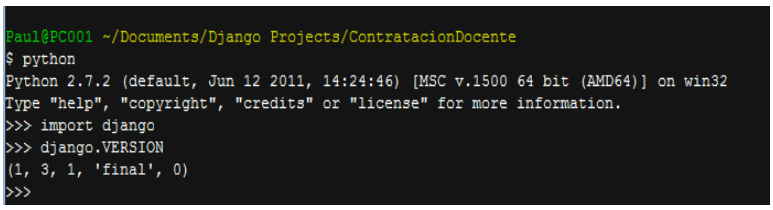


Figura III. 21: Versión del Framework Django.

Costos: La inversión en el desarrollo respecto al costo es mínima debido a la agilidad y facilidad en la construcción de las aplicaciones, en el momento de producción.

Licenciamiento: Es relacionado con el costo, ya que Django cuenta con un licenciamiento libre BSD.

INSTALACIÓN

La instalación del framework es muy sencilla, previamente teniendo instalado Python el paquete se instalara únicamente con una sola línea de código.

```
>>> Python setup.py install
```

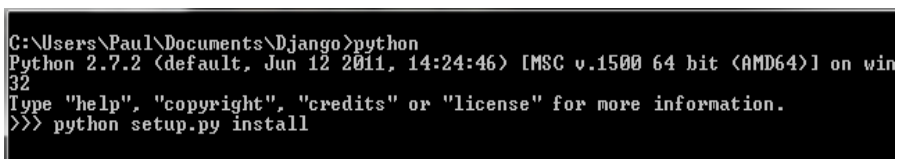


Figura III. 22: Instalación de Django

Entorno de desarrollo: El entorno de desarrollo seleccionado es Aptana, el cual es un IDE de desarrollo web, el mismo que cumple apropiadamente los requerimientos para un desarrollo más ágil.

Figura III. 23: Entorno de desarrollo web Aptana Studio 3.0

3.4.5. Desarrollo de los módulos de prueba con la tecnología Ruby on Rails Framework

3.4.5.1. Módulo 1

Virtualmente todas las aplicaciones Rails inician de la misma manera, con un comando. Este útil comando crea un esqueleto de la aplicación Rails en un directorio de su elección. Para empezar, se debe crear un directorio para los proyectos de Rails y, a continuación, ejecute el comando Rails para hacer la aplicación:

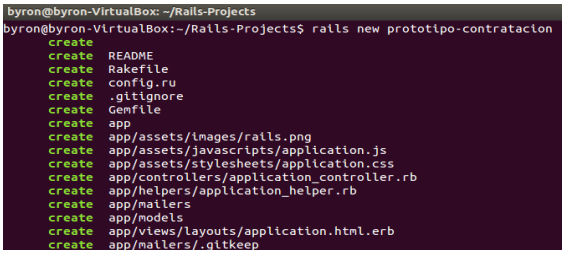


Figura III. 24: Creación de un proyecto con RoR

Esto crea una cantidad de archivos y directorios. Esta es una estructura de directorios y ficheros estándar de Rails.

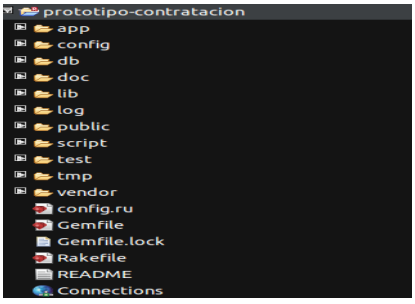


Figura III. 25: Estructura de directorios de un proyecto con RoR

Antes de todo se debe modificar el archivo **Gemfile,** el cual contiene las gemas que indican con que librerías de RoR va a trabajar en la aplicación, para este caso que estamos utilizando PostgreSQL se debería añadir **gem 'pg'** a dicho archivo.

En Rails se trabaja con 3 entornos de trabajo: desarrollo, prueba y producción, lo cual indica que se puede trabajar con tres motores de dase de datos diferentes. Se debe configurar los parámetros, en este caso vamos a configurar la conexión a la base de datos PostgreSQL para los tres entornos, lo cual se lo hace en el archivo **config/database.yml** en el cual se debe configurar:

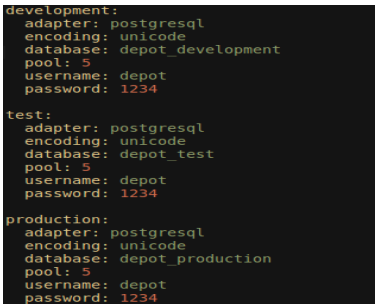


Figura III. 26: Configuración de la base de datos.

En donde:

adapter: indica con que adaptador de base de base de datos se va a trabajar.

encoding: Es un parámetro que indica la codificación que trabajaremos, de preferencia debemos dejarlo en Unicode, que es el que es el valor por defecto

database: Este parámetroes el nombre de la base de datos.

pool: su valor por defecto es 5.

username: aquí se indica el usuario con permisos para realizar que utilizará la base de datos

password: corresponde a la contraseña del usuario anterior.

Scaffolding

Ahora que ya se tiene lista la conexión con la base de datos, se puede a comenzar a crear los modelos vistas y controladores, para lo cual se va utilizar el comando scaffold

que simplifica estas acciones ya que nos ayuda creando el modelo la vista y el controlador, así como la migración que creara la tabla en la base de datos.

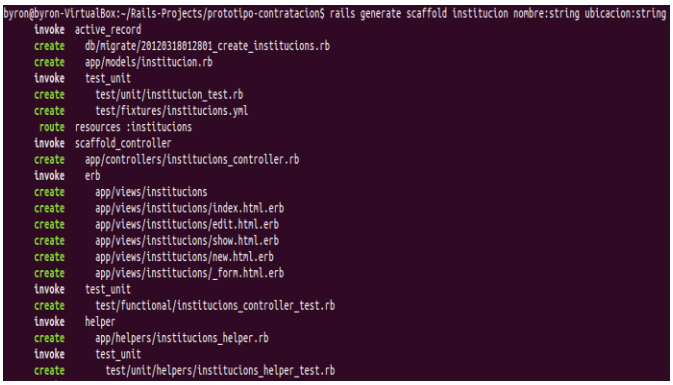


Figura III. 27: Creación mediante scaffold del MVC para Institución.

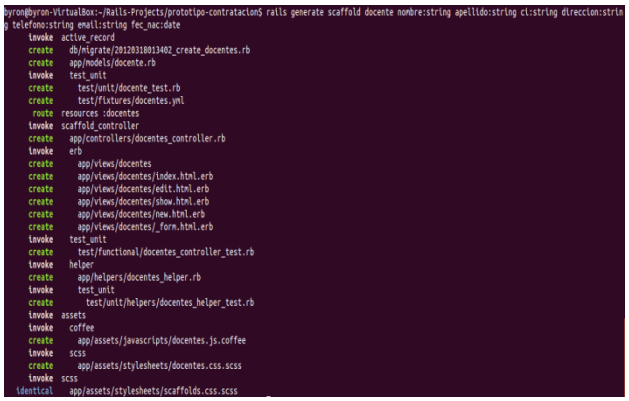


Figura III. 28: Creación mediante scaffold del MVC para Docente

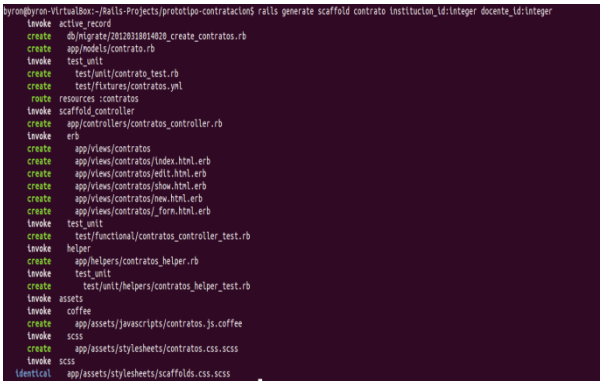


Figura III. 29: Creación mediante scaffold del MVC para Contrato

MODELO

En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averigua automáticamente qué tabla usar y qué columnas tiene.

Las definiciones de las clases también detallan las relaciones entre clases con sentencias de mapeo objeto relacional. En nuestro prototipo, la clase Institución tiene una definición has_many:contratos, aquí también es en donde se deben realizar las rutinas de validación de datos:

```
class Contrato < ActiveRecord::Base
belongs_to :docente
belongs_to :institucion
end
```

Figura III. 30: Modelo de la clase Contrato.

```
class Docente < ActiveRecord::Base
has_many :contratos, dependent: :destroy
validates :nombres, :apellidos, :ci, :direccion, :telefono, :email, presence: true
validates :ci, uniqueness: true
end
```

Figura III. 31: Modelo de la clase Docente.

```
class Institucion < ActiveRecord::Base
has_many :contratos
validates :nombre, :ubicacion, presence: true
validates :nombre, uniqueness: true
end
```

Figura III. 32: Modelo de la clase Institución.

VISTA

En MVC, Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador. Con frecuencia en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML.

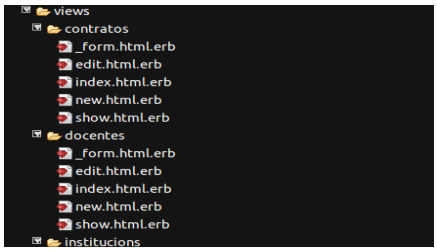


Figura III. 33: Estructura de las Vistas generadas por scaffold.

Existen en la actualidad muchas maneras de gestionar las vistas. El método que se emplea en Rails por defecto es usar Ruby empotrado (archivos.html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP.

Las vistas generadas automáticamente son 5, claro que esto depende de lo que se necesite y ustedes variar es decir podrían ser menos o más:

1._form.html.erb :sirve como plantilla para las demás.

```
<%= form_for(@contrato) do |f| %>
  <% if @contrato.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@contrato.errors.count, "error") %> prohibited this contrato from being saved:</h2>

      <ul>
      <% @contrato.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :institucion_id %><br />
    <%= f.number_field :institucion_id %>
  </div>
  <div class="field">
    <%= f.label :docente_id %><br />
    <%= f.number_field :docente_id %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Figura III. 34: Código HTML con código Ruby embebido del _form.html.erb

index.html.erb :muestra una lista de todos los registros de una tabla determinada.

```
<h1>Listing contratos</h1>

<table>
  <tr>
    <th>Institucion</th>
    <th>Docente</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @contratos.each do |contrato| %>
  <tr>
    <td><%= contrato.institucion_id %></td>
    <td><%= contrato.docente_id %></td>
    <td><%= link_to 'Show', contrato %></td>
    <td><%= link_to 'Edit', edit_contrato_path(contrato) %></td>
    <td><%= link_to 'Destroy', contrato, confirm: 'Are you sure?', method: :delete %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New Contrato', new_contrato_path %>
```

Figura III. 35: Código Html con código Ruby embebido del index.html.erb

show.htm.erb :muestra los detalles de un determinado registro.

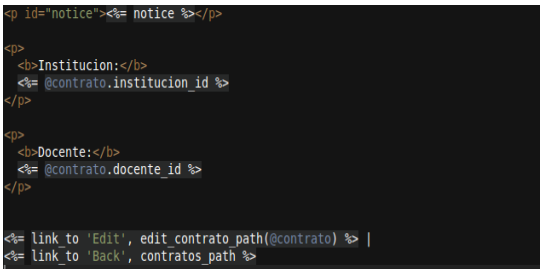


Figura III. 36: Código Html con código Ruby embebido del show.html.erb

new.html.erb :crea un nuevo registro.

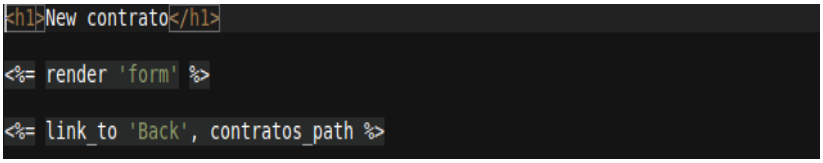


Figura III. 37: Código Html con código Ruby embebido del new.html.erb

edit.htm.erb :edita un registro determinado.

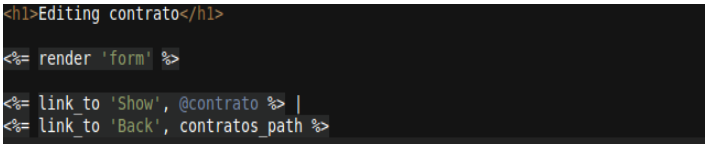


Figura III. 38: Código Html con código Ruby embebido del edit.html.erb

Todas las vistas que anteriormente se han mostradas son dinámicas y modificables, se les pueden aplicar apariencias con hojas de estilo, además no solo pueden ser dinámicas sino también con contenido estático.

La adaptación con Hojas de estilo se lo hace simplemente cargando las Hojas de estilo al proyecto, y utilizándolas en nuestras vistas en este caso se utiliza como plantilla madre a: **app/views/layouts/application.html.erb,** la que servirá como base y en esta se irán cargando el contenido de las demás vistas a través de una línea de código <%= yield %>

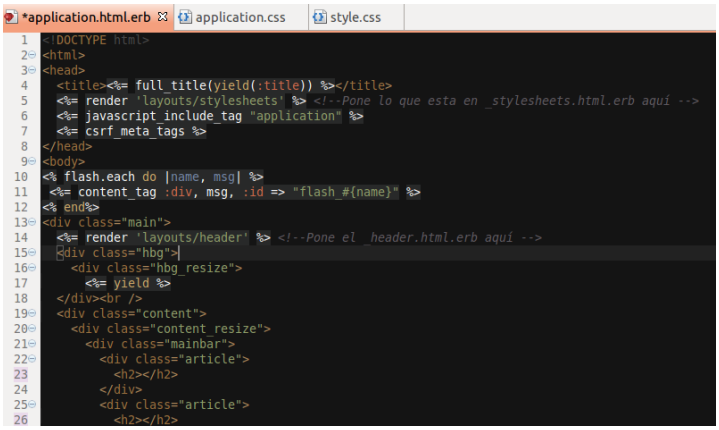


Figura III. 39: Plantilla base de la aplicación prototipo.

Al final aplicando la misma plantilla HTML con su respectivo estilo la página de inicio quedaría de la siguiente manera:

Figura III. 40: Página de inicio de la aplicación prototipo.

Controlador

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos.

Como Rails proporciona andamiaje (scaffolding), que nos ayuda a construir rápidamente la mayor parte de la lógica y vistas necesarias para realizar las operaciones más frecuentes, en este caso los controladores que se generaron son:

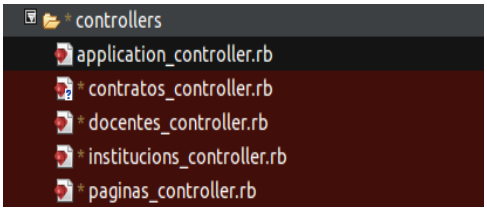


Figura III. 41: Controladores de la aplicación prototipo.

El controlador application_controller.rb no fue generado automáticamente, pero este si se creó solo al momento de crear la nueva aplicación prototipo.

```
class InstitucionsController < ApplicationController
  before_filter :authenticate_user!
  # GET /institucions
  # GET /institucions.json
  def index
    @institucions = Institucion.all
    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @institucions }
    end
  end

  # GET /institucions/1
  # GET /institucions/1.json
  def show
    @institucion = Institucion.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @institucion }
    end
  end

  # GET /institucions/new
  # GET /institucions/new.json
  def new
    @institucion = Institucion.new
    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @institucion }
    end
  end
```

Figura III. 42: Ejemplo de un controlador estándar generado por scaffolding.

En la siguiente Figura se muestra el formulario de un ingreso de una nueva institución:

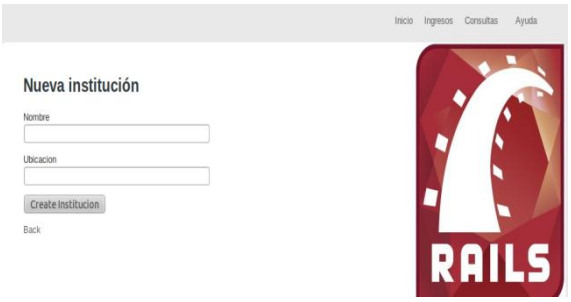


Figura III. 43: Ejemplo de un controlador estándar generado por scaffolding.

Ahora se prueba la validación de datos del formulario, cabe recalcar que el controlador es el encargado de llevar los datos del formulario (vista) hacia el modelo y este es quien se encarga de validar si los datos son correctos o no caso contrario este le dirá al controlador que hay errores, que serán mostrados en la vista.

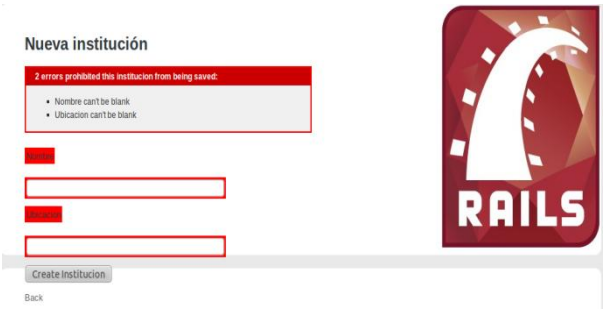


Figura III. 44: Ejemplo de validación de datos.

3.4.5.2. Módulo 2

En este módulo implementaremos la autenticación en RoR, para lo cual de varias soluciones hemos tomado la solución llamada ***devise,*** esta solución de autenticación es basada en Warden (que está basado en Rack), Warden permite que varias aplicaciones compartan la autenticación. Tiene autenticación: vía HTTP, POST, single access token u OAuth2, además permite tener varios tipos de usuario (scopes).

```
gem 'devise', '1.1.rc0'
```

```
bundle install
$ rails generate devise_install
      create  config/initializers/devise.rb
      create  config/locales/devise.en.yml

===============================================================================

Some setup you must do manually if you haven't yet:

  1. Setup default url options for your specific environment. Here is an
     example of development environment:

       config.action_mailer.default_url_options = { :host => 'localhost:3000' }

     This is a required Rails configuration. In production is must be the
     actual host of your application

  2. Ensure you have defined root_url to *something* in your config/routes.rb.
     For example:

       root :to => "home#index"

===============================================================================
```

Figura III. 45: Instalación de devise.

Para utilizar devise primero se debe haber instalado su gema, y haber hecho la generación del modelo User, que es parecido a cualquier otro modelo ActiveRecord pero tiene una llamada al método devise, que es donde sucede la magia de la autenticación. El método devise recibe como argumentos una lista de los módulos que se quiere que sean soportados en el prototipo, en nuestro ejemplo vemos los módulos que veíamos antes: :rememberable y :recoverable. Es fácil añadir o quitar módulos de esta lista, con lo que personalizaríamos la funcionalidad de autenticación de Devise para

ajustarla a las necesidades del prototipo. Por ejemplo se ha quitado :confirmable porque no es necesario que los usuarios tengan que confirmar su correo.

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :token_authenticatable, :lockable, :timeoutable and :activatable
  # :confirmable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  # Setup accessible (or protected) attributes for your model
  attr_accessible :email, :password, :password_confirmation
end
```

Figura III. 46: Clase User que se utiliza en la autenticación con devise.

Figura III. 47: Interfaz para realizar el login.

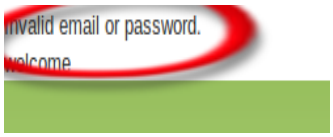


Figura III. 48: Error al ingresar un usuario o contraseña incorrectos.

Figura III. 49: Autenticación correcta

3.4.5.3. Módulo 3

Producto

Madurez del producto: RoR posee las siguientes versiones:

Tabla III. IX: Versiones de RoR

Versión	Fecha
1.0	December 13, 2005
1.2	January 19, 2007
2.0	December 7, 2007
2.1	June 1, 2008
2.2	November 21, 2008
2.3	March 16, 2009
3.0	August 29, 2010
3.1	August 31, 2011
3.2	January 20, 2012

Para el desarrollo del prototipo se está utilizando la versión 3.1.3

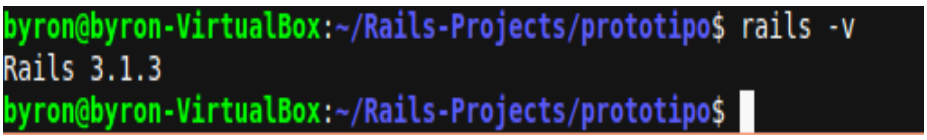


Figura III. 50: Versión de Rails

Además, durante el proceso de desarrollo se puede utilizar el Versionamiento de la aplicación web utilizando **Git**, es decir que durante el proceso de desarrollo la aplicación se puede ir respaldando antes de hacer un cambio, a esto en Rails se le denomina como Versionamiento.

Costos: RoR realmente no tiene costo en el desarrollo, pero en el momento de ponerlo a producción obviamente intervendrán valores que tendrán que ver con el dominio, el alojamiento de la aplicación web en un Web hosting.

Licenciamiento: El framework Rails posee un licenciamiento Software libre MIT, mientras que el lenguaje de programación licenciamiento BSD, que a la final los dos dejan libre su uso, ya que ambos tipos de licenciamiento son Software libre.

Instalación: La instalación de esta tecnología es sencilla pero a la vez un tanto larga, el proceso de instalación suele durar de 20 a 30 minutos, ya que primero se debe instalar el leguaje Ruby, las gemas para Rubygems, y por último el framework Rails, y lo recomendado para esta versión de Rails es usar RVM (Ruby versión manager), que como su nombre lo indica es un administrador de Ruby.

Una vez instalado RVM para la instalación de Rails 3.1 se realizó lo siguiente:

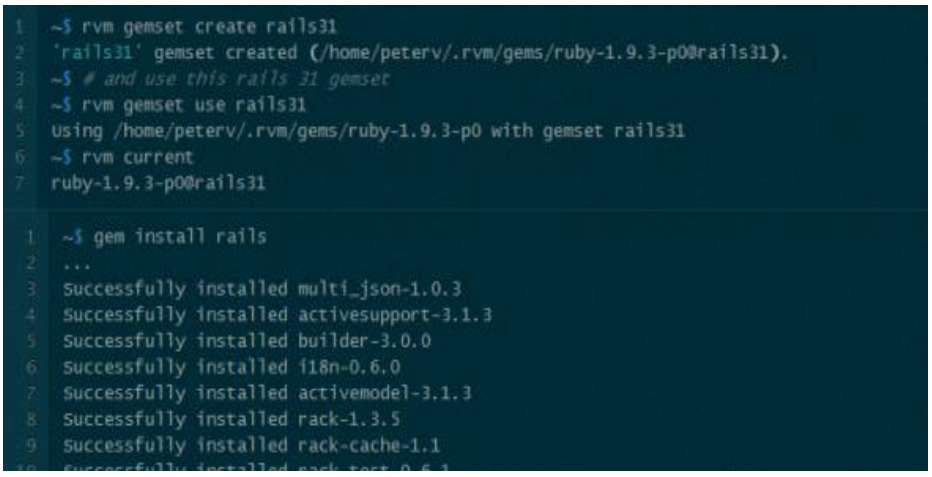


Figura III. 51: Instalación de Rails 3.1.3

Entorno de desarrollo: el IDE utilizado es Aptana Studio 3.0, debido a que se adapta, a las necesidades en el desarrollo, sobre todo en la navegación por los directorios de la aplicación, además también es Software libre.

3.5. Análisis Comparativo

Los resultados de los indicadores con sus respectivos índices se realizan un cuadro comparativo de los Frameworks Django y Ruby on Rails, cuyas pruebas de desarrollo fueron realizadas bajo los mismos escenarios.

La calificación para cada parámetro se determinara de acuerdo a la escala que se mostrara a continuación, lo cual nos permitirá determinar la tecnología que se adapta más a un desarrollo más productivo y ágil de aplicaciones web.

Tabla III. X: Valoración cualitativa y cuantitativa.

Regular	**Bueno**	**Muy Bueno**	**Excelente**
<70%	>=70% y <80%	>=80% y <95%	>=95%

La evaluación para los indicadores es de acuerdo al tiempo y experiencia de desarrollo, para lo cual la valoración variará entre uno y cuatro.

Tabla III. XI: Escala de valoración cualitativa y cuantitativa para los indicadores

Valor Cualitativo		**Valor Representativo**
Insuficiente	No Satisfactorio	☺
Parcial	Poco Satisfactorio	☺☺
Suficiente	Satisfactorio	☺☺☺
Excelente	Muy Satisfactorio	☺☺☺☺

Tabla III. XII: Equivalencias de los valores cuantitativos.

Valor Cuantitativo	1	2	3	4
	1 - 10	11 - 13	15 - 17	18 - 20
Equivalencias	0.25	0.50	0.75	1

Para la realización de la comparación se utilizará la siguiente nomenclatura:

X *= Representa el puntaje obtenido por la tecnología Django.*

Y *= Representa el puntaje obtenido por la tecnología Ruby on Rails.*

W = *Representa el puntaje sobre el cual será evaluado el parámetro.*

Cdj = *Representa el puntaje alcanzado de Django en el parámetro.*

Cror = *Representa el puntaje alcanzado de Ruby on Rails en el parámetro.*

Ct = *Representa el puntaje por el cual es evaluado el parámetro.*

Pdj *= Calificación porcentual obtenida por Django.*

Pror *= Calificación porcentual obtenida por Ruby on Rails.*

Las fórmulas que se utilizarán en el proceso del análisis comparativo son las siguientes:

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

3.5.4. Manejo del patrón MVC

3.5.4.1. Indicador 1: Modelo/Acceso a datos

El acceso a la base de datos es un parámetro muy importante en un desarrollo ágil de aplicaciones web, por lo cual se analizara los aspectos necesarios para la manipulación de los datos y acceso a los mismos.

- **Índice 1.1: Soporte a múltiples bases de datos:** En este índice se valorizará tomando en consideración los números de motores de base de datos que soporta el framework, en base a la construcción del modulo 1 de pruebas (MODELO)

Tabla III. XIII: Valoración del Índice 1.1: Soporte a múltiples bases de datos

Valoración	
Numero de Bases de datos que soporta	Valoración cualitativa
Hasta 1	No Satisfactorio
Hasta 2	Poco Satisfactorio

Valoración	
Hasta 3	Satisfactorio
>= 4	Muy Satisfactorio

- **Índice 1.2: Manipulación con la base de datos:** En este índice se valorizará de acuerdo al tiempo en configurar la conexión a la base de datos, en base a la construcción del modulo 1 de pruebas (MODELO).

Tabla III. XIV: Valoración del Índice 1.2: Manipulación con la base de datos

Valoración	
Tiempo en minutos	Valoración cualitativa
16 a 20	No Satisfactorio
11 a 15	Poco Satisfactorio
6 a 10	Satisfactorio
1 a 5	Muy Satisfactorio

- **Índice 1.3: Desempeño con la base de datos:** Este índice valorizará el tiempo en la creación de los modelos para las instituciones y docentes que posteriormente serán las tablas en la base de datos, en base a la construcción del modulo 1 de pruebas (MODELO).

Tabla III. XV: Valoración del Índice 1.3: Desempeño con la base de datos

Valoración	
Tiempo en minutos	Valoración Cualitativa
16 a 20	No Satisfactorio
11 a 15	Poco Satisfactorio
6 a 10	Satisfactorio
1 a 5	Muy Satisfactorio

- **Índice 1.4: Mapeador del objeto relacional:** Este índice valorizará el tiempo de realizar una inserción de una institución través de la utilización del ORM en el Módulo 1.

Tabla III. XVI: Valoración del Índice 1.4: Mapeador del objeto relacional

Valoración	
Tiempo en minutos	Valoración Cualitativa
16 a 20	No Satisfactorio
11 a 15	Poco Satisfactorio
6 a 10	Satisfactorio
1 a 5	Muy Satisfactorio

3.5.4.2. Valoraciones le parámetro Modelo / Acceso a datos

Tabla III. XVII: Resultados del Indicador 1: Modelo

Indicadores	Django		Ruby on Rails	
	Valor Cualitativo	Valor Obtenido/4	Valor Cualitativo	Valor Obtenido/4
Soporte a múltiples DBMS	Muy Satisfactorio	4	Muy Satisfactorio	4
Manipulación con la BD	Muy Satisfactorio	4	Muy Satisfactorio	4
Desempeño con la BD	Satisfactorio	3	Muy Satisfactorio	4
Mapeador objeto relacional	Muy Satisfactorio	4	Muy Satisfactorio	4

3.5.4.3. Interpretación

Soporte para múltiples bases de datos: El soporte desde el framework hacia la administración de la información mediante los distintos motores de base de datos es muy importante, ya que Django, así como Ruby on Rails soportan 4 motores de base de datos, por lo que ambos han obtenido una calificación de 4 puntos que equivale a Muy satisfactorio.

Manipulación con la base de datos: Este es un índice que nos permite identificar la eficiencia que presenta la tecnología para conectarse a uno u otro motor de base de datos por lo que el tiempo en realizar la conexión a la base de datos describe esta característica. Después de realizar las pruebas de conexión respecto al tiempo, Django así como Ruby on Rails se lo puede implementar en un tiempo menor a cinco minutos, puesto que ambos frameworks obtienen una calificación 4 puntos que equivale a Muy satisfactorio.

Desempeño con la base de datos: El desarrollo de los modelos en los respectivos frameworks, nos permitió determinar el desempeño que posee cada tecnología con la base de datos. Puesto que el tiempo de implementación de los modelos en Django estuvo en el rango de 6 a 10 minutos obtuvo una calificación de 3 puntos que equivale a Satisfactorio, no obstante Ruby on Rails presenta una mejor manipulación y facilidad en la creación de los modelos siendo estos desarrollados en un rango de 1 a 5 minutos por lo que RoR obtiene una calificación de 4 puntos que equivale a Muy Satisfactorio.

Mapeador Objeto Relacional: La utilización de ORM que presenta cada framework permite interactuar con la base de datos desde el propio lenguaje mediante la creación de objetos. El tiempo en realizar una inserción de una institución desde el ORM en el módulo 1, permite establecer que framework presenta un mejor manejo del Mapeador objeto relacional, obteniendo como resultado que Django y Ruby on Rails realizan esta acción en menos de 5 minutos, ambos obtienen una calificación de 4 puntos, que equivale a Muy Satisfactorio.

3.5.4.4. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$$Cdj: 4 + 4 + 3 + 4 = 15$$

$$Cror: 4 + 4 + 4 + 4 = 16$$

$$Ct: 4 + 4 + 4 + 4 = 16$$

$$Pdj: \left(\frac{15}{16}\right) * 100\% = 93.75\%$$

$$Pror: \left(\frac{16}{16}\right) * 100\% = 100\%$$

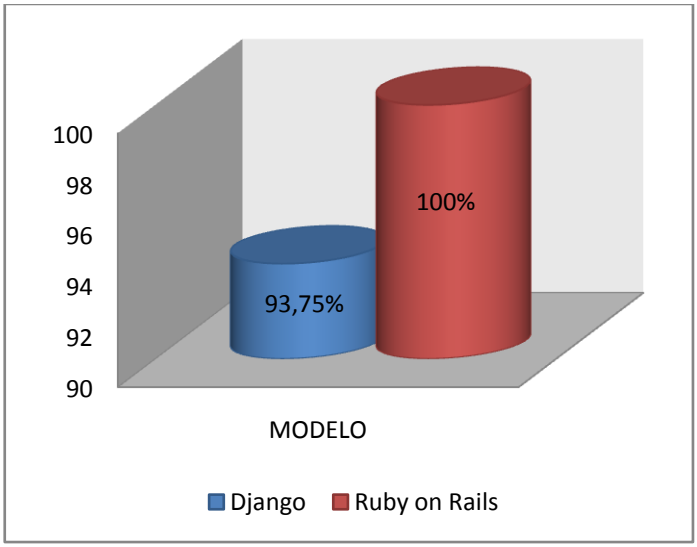


Figura III. 52: Resultado Final del Indicador 1: Modelo

3.5.4.5. Representación de Resultados

Tabla III. XVIII: Modelo, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	**Valor Calificativo**	**Valor Representativo**	**Valor Calificativo**
Soporte a múltiples DBMS	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Manipulación con la BD	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Desempeño con la BD	☺☺☺	Suficiente	☺☺☺☺	Excelente
Mapeador objeto relacional	☺☺☺☺	Excelente	☺☺☺☺	Excelente

3.5.5. Indicador 2: Vista

El parámetro Vista posee 3 índices cada uno de los cuales mediante la facilidad en su implementación del Módulo 2 de pruebas, se obtuvieron resultados evaluando valores mínimos y máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 2.1: Uso de plantillas:** En este índice se valorizará la capacidad que tiene el framework en la utilización de un sistema de plantillas, de acuerdo a la facilidad en su implementación, su calificación será de 1 a 20, en base al modulo 1 de pruebas, la capa de las interfaces (VISTA)

Tabla III. XIX: Valorización del Índice 2.1: Uso de plantillas

Valoración	
Implementación/20	Valoración

	Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 2.2: Adaptación con hojas de estilo:** En este índice se valorizará tomando en consideración la facilidad en la adaptación de las paginas creadas con hojas de estilo CSS, permitiendo personalizar varias páginas sin tener que repetir el código, en base al modulo 1 de pruebas, la capa de las interfaces (VISTA)

Tabla III. XX: Valorización del Índice 2.2: Adaptación con hojas de estilo.

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 2.3: Fusión de código y diseño:** En este índice tendrá valorizaciones basadas en la fusión del código de la aplicación con el diseño de la misma, estableciendo una mejora adaptación en el mismo, en base al modulo 1 de pruebas, la capa de las interfaces (VISTA).

Tabla III. XXI: Valorización del Índice 2.3: Fusión de código y diseño

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio

15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

3.5.5.1. Valoraciones

Tabla III. XXII: Resultados del Indicador 2: Vista

Indicadores	Django		Ruby on Rails	
	Valor Cualitativo	**Valor Obtenido /20**	**Valor Cualitativo**	**Valor Obtenido /20**
Uso de plantillas	Satisfactorio	17	Satisfactorio	16
Adaptación con hojas de estilo	Muy Satisfactorio	18	Satisfactorio	17
Fusión de código y diseño	Satisfactorio	19	Satisfactorio	17

3.5.5.2. Interpretación

Uso de plantillas: Django framework presenta un completo sistema de plantillas, el cual su implementación es de manera muy sencilla, nos permite realizar paginas de manera rápida y ágil, pero al ser un sistema de plantillas ideal para diseñadores mas no para programadores por lo que la creación de la plantilla posee un pequeño grado de complejidad es por eso que obtiene una calificación de 17 que equivale a Satisfactorio. Por otra parte Ruby on Rails no posee un sistema de plantillas pero tiene la ventaja de que permite la Herencia de Plantillas en la cual la aplicación mantiene cargada un plantilla en el navegador y sobre esta se van cargando las vistas necesarias, debido a que no posee un sistema propio de plantillas obtiene una calificación de 16 que equivale a Satisfactorio.

Adaptación con hojas de estilo: La utilización de estilos en las paginas HTML es muy importante para un diseño personalizado, Django permite la administración de los recursos a través del sistema de plantillas, permite crear estilos para la adaptación de una sola página base de fácil implementación por lo que obtiene una calificación de 18 que equivale a Muy Satisfactorio. Por otro lado RoR, se adapta perfectamente con las hojas de estilo, además puede ser utilizado con una mejora a las hojas de estilo CSS llamado SCSS (de "Sassy CSS"), que mejora incrementando herencia, uso de variables, etc., lo cual mejora la estructura de una hoja de estilo, por esto RoR obtiene una calificación Satisfactoria de 17 puntos.

Fusión de código y diseño: Como se dijo anteriormente el sistema de plantillas que utiliza Django es ideal para los diseñadores mas no para los programadores por lo que de manera intencional permite la inclusión directa de código en el diseño HTML, permitiendo una correcta fusión por lo que se valora a Django con una calificación de 19 que equivale a Muy Satisfactorio. Mientras que RoR también permite esta fusión, pero resulta un tanto complejo debido a que se debe crear muchas vistas lo cual en base a lo experimentado hace el trabajo más tedioso, y por tanto se requiere algo más de tiempo, por estas razones se obtiene una calificación de 17 puntos equivalente a Satisfactorio.

3.5.5.3. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$Cdj: 17 + 18 + 19 = 54$

$Cror: 16 + 17 + 17 = 50$

$Ct: 20 + 20 + 20 = 60$

$Pdj: \left(\frac{54}{60}\right) * 100\% = 90\%$

$Pror: \left(\frac{50}{60}\right) * 100\% = 83.33\%$

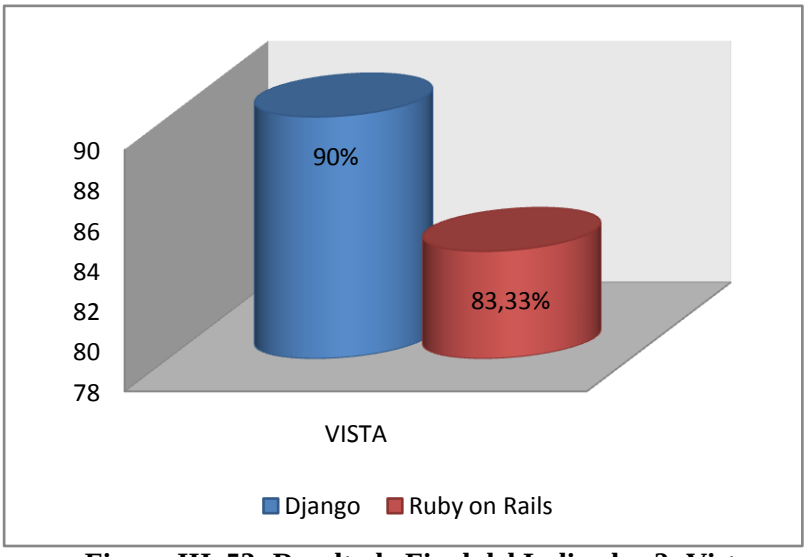


Figura III. 53: Resultado Final del Indicador 2: Vista

3.5.5.4. Representación de Resultados

Tabla III. XXIII: Representación del Indicador 2: Vista, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Representativo	Valor Calificativo
Uso de plantillas	☺☺☺	Suficiente	☺☺☺	Suficiente
Adaptación con hojas de	☺☺☺☺	Excelente	☺☺☺	Suficiente

estilo				
Fusión de código y diseño	☺☺☺☺	Excelente	☺☺☺	Suficiente

3.5.6. Indicador 3: Controlador

EL parámetro controlador posee 3 índices cada uno de los cuales fue evaluado mediante el desarrollo de un formulario para la inserción de un docente, tomando en consideración el tiempo en su implementación, tomando sus valores mínimos, máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 3.1: Generación automática de formularios:** En este índice se valorizará tomando en consideración la facilidad que presente el framework para generar automáticamente un formulario, en base el modulo 1 de pruebas, (creación de formularios).

Tabla III. XXIV: Valorización del Índice 3.1: Generación automática de formularios

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 3.2: Validación de formularios:** En este índice se valorizará la validación que presenta el formulario en caso de que los datos ingresados sean erróneos, en base el modulo 1 de pruebas, (creación de formularios).

Tabla III. XXV: Valorización del Índice 3.2: Validación de formularios

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 3.3: Manejo visual de componentes de formulario:** En este índice se valorizará el entorno visual que presenta el framework para el desarrollo de los formularios mediante el uso de sus componentes, en base el modulo 1 de pruebas, (creación de formularios).

Tabla III. XXVI: Valorización del Índice 3.3: Manejo visual de componentes de formulario

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

3.5.6.1. Valoraciones

Tabla III. XXVII: Resultados del Indicador 3: Controlador.

Indicadores	**Django**		**Ruby on Rails**	
	Valor Cualitativo	**Valor Obtenido/20**	**Valor Cualitativo**	**Valor Obtenido/20**

Generación automática de formularios	Muy Satisfactorio	19	Muy Satisfactorio	18
Validación de formularios	Muy Satisfactorio	20	Muy Satisfactorio	19
Manejo visual de componentes de formulario	No Satisfactorio	10	No Satisfactorio	10

3.5.6.2. Interpretación

Generación automática de formularios: Django permite una generación automática de formulario a partir de los modelos lo cual brinda un ahorro en el tiempo y facilita la reutilización, los formularios se generan creando modelos de tipo formulario y se generan automáticamente tomando los atributos de los modelos, es por esto que obtuvo una calificación de 19 que equivale a Muy Sobresaliente. Por otra parte RoR igualmente realiza la generación de los formularios de forma automática por medio de simples comandos que toman como referencia la estructura de la base de datos desde un archivo del proyecto "schema.rb", se crean los campos del formulario según los tipos de dato, algo muy rápido y simple, pero la forma automática que hace esta tarea tiene su desventaja ya que se crea el formulario a la manera de RoR y en idioma ingles, lo que obliga a que tengamos que personalizar el formulario, una tarea no compleja pero que va a requerir un poco más de tiempo, por lo mencionado RoR obtiene un valor Muy Satisfactorio de 18 puntos.

Validación de formularios: La validación de formularios es muy importante a nivel de aplicación, Django permite una validación automática de la información que se ingresa en el formulario debido a que el formulario se maneja como un objeto creado a partir de una clase de tipo FORM, este objeto presenta propiedades y métodos que permiten validar de una manera sencilla y automática el formulario de acuerdo a tipo de dato que se requiere que ingrese por lo que Django obtiene una calificación de 20 que equivale a

Muy Satisfactorio. Mientras que en RoR la validación también se lo hace de forma sencilla escribiendo simples líneas de código en el Modelo correspondiente a la tabla de la BD, a diferencia de Django RoR realiza la validación en el modelo, lo cual le hace obtener la calificación de 19 siendo muy satisfactorio.

Manejo visual de componentes de formulario: Los componentes de los formularios son manejados como componentes de un objeto, dependiendo de la implementación del formulario, pero al no contar Django con un entorno que permita gestionar de manera visual los componentes de un formulario obtiene una calificación de 10 que equivale a No Satisfactorio. Al igual RoR, tampoco posee esta característica por lo tanto obtiene la misma calificación.

3.5.6.3. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$$Cdj: 19 + 20 + 10 = 49$$

$$Cror: 18 + 19 + 10 = 47$$

$$Ct: 20 + 20 + 20 = 60$$

$$Pdj: \left(\frac{49}{60}\right) * 100\% = 81{,}66\%$$

$$Pror: \left(\frac{47}{60}\right) * 100\% = 78.33\ \%$$

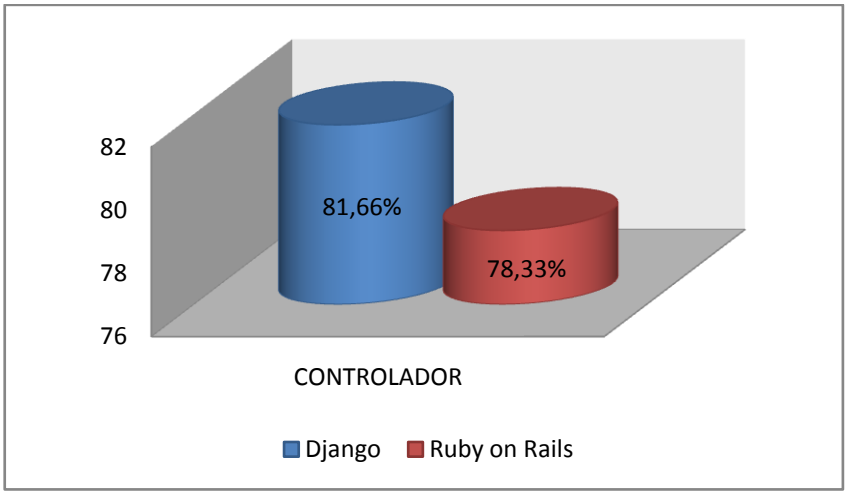


Figura III. 54: Resultado Final del Indicador 3: Controlador

3.5.6.4. Representación de Resultados

Tabla III. XXVIII: Representación del Indicador 3: Controlador, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Representativo	Valor Calificativo
Generación automática de formularios	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Validación de formularios	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Manejo visual de componentes de formulario	☺	Insuficiente	☺	Insuficiente

3.5.7. Indicador 4: Reutilización

El parámetro Reutilización posee 4 índices, para los cuales se realizo una valoración de acuerdo al desarrollo del módulo 1, con el objetivo de probar el principio DRY. Sus

resultados obtenidos serán evaluados tomando sus valores mínimos, máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 4.1: Herencia de clase base:** En este índice se valorizará la herencia desde la clase base de la cual se heredan los atributos y características para los modelos.

Tabla III. XXIX: Valorización del Índice 4.1: Herencia de clase base

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 4.2: Herencia de plantillas:** En este índice se valorizará la facilidad en la implementación de la herencia de plantillas desde una plantilla base en el modulo 1.

Tabla III. XXX: Valorización del Índice 4.2: Herencia de plantillas

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 4.3: Personalización de código heredado:** En este índice se valorizará la personalización del código heredado desde las plantilla base

Tabla III. XXXI: Valorización del Índice 4.3: Personalización de código heredado.

Valoración	
Implementación/20	Valoración Cualitativa
1 a 10	No Satisfactorio
11 a 13	Poco Satisfactorio
15 a 17	Satisfactorio
18 a 20	Muy Satisfactorio

- **Índice 4.4: Tamaño de la aplicación:** En este índice se valorizará el tamaño en Megabytes que posee la aplicación web desarrollada en el framework

Tabla III. XXXII: Valorización del Índice 4.4: Tamaño de la aplicación

Valoración		
Tamaño en MB	Implementación/20	Valoración Cualitativa
> 3	1 a 10	No Satisfactorio
2 a 3	11 a 13	Poco Satisfactorio
1 a 2	15 a 17	Satisfactorio
0 a 1	18 a 20	Muy Satisfactorio

3.5.7.1. Valoraciones

Tabla III. XXXIII: Resultados del Indicador 4: Reutilización

Indicadores	**Django**		**Ruby on Rails**	
	Valor Cualitativo	**Valor Obtenido/20**	**Valor Cualitativo**	**Valor Obtenido/20**

Herencia de clase base	Muy Satisfactorio	18	Muy Satisfactorio	19
Herencia de plantillas	Muy Satisfactorio	20	Muy Satisfactorio	19
Personalización de código heredado	Satisfactorio	17	Satisfactorio	17
Tamaño de la aplicación	Muy Satisfactorio	20	No Satisfactorio	10

3.5.7.2. Interpretación

Herencia de clase base: El framework Django en la implementación de los modelos permite fácilmente heredar las propiedades y atributos desde la clase base MODELS, posibilitando le reutilización de objetos, atributos y propiedades que ya se encuentran implementadas por lo que obtiene una calificación de 18 que equivale a Muy Satisfactorio. En cambio RoR y debido a que el lenguaje mismo Ruby es puramente orientado a objetos permite la herencia inclusive de las características de cualquier tipo de dato, igualmente en el framework la herencia de la clase base (Active Record), se la utiliza en todos las clases de la aplicación, con lo mencionado RoR obtiene una puntuación de 19, siendo Muy Satisfactorio.

Herencia de plantillas: El sistema de plantillas de Django permite crear una plantilla base de la cual permite heredar todas los propiedades de la misma hacia sus plantillas hijas, permitiendo incluir archivos HTML y bloques de contenedores, lo cual nos beneficia al momento de crear rápidamente distintas paginas, razones que le permiten obtener una calificación de 20 puntos que equivale a Muy Satisfactorio. En RoR existe una similitud en este sentido ya que igualmente se mantiene una plantilla de base y sobre esta se van cargando pedazos de código HTML, y otros con código mixto Ruby que en este caso son las Vistas, mismas que son invocadas con simples líneas de código, es por esto que se le ha calificado con 19 siendo Muy Satisfactorio.

Personalización de código heredado: La versatilidad y el dinamismo que presentan tanto Django así como Ruby on Rails al momento de permitir la personalización del

código que se hereda da muchas ventajas para que una aplicación web se adapte con facilidad a los cambios por lo que ambas tecnologías obtienen una calificación de 17 puntos que equivale a Satisfactorio.

Tamaño de la Aplicación: Este indicador es muy importante ya que permite obtener una clara conjetura de la reutilización a la baja utilización de Bytes para realizar las mismas funciones con un tamaño de aplicación que se encuentra en el rango de 0 a 1 MB, Django obtiene una calificación de 20 puntos. Mientras tanto que Ruby on Rails presenta características en la creación de la aplicación que generan archivos los cuales incrementan el tamaño de la aplicación haciéndola menos ágil, teniendo esta un tamaño de aplicación mayor a 3 MB, por lo que obtiene una calificación de 10 puntos que equivale a No Satisfactorio.

3.5.7.3. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$$Cdj: 18 + 20 + 17 + 20 = 75$$

$$Cror: 19 + 19 + 17 + 10 = 65$$

$$Ct: 20 + 20 + 20 + 20 = 80$$

$$Pdj: \left(\frac{75}{80}\right) * 100\% = 93{,}75\%$$

$$Pror: \left(\frac{65}{80}\right) * 100\% = 81.25\ \%$$

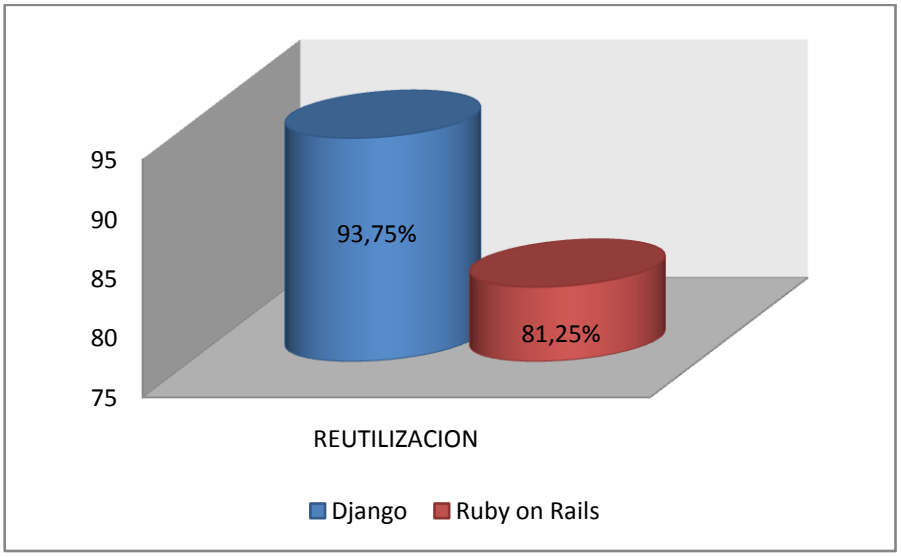


Figura III. 55: Resultado Final del Indicador 4: Reutilización

3.5.7.4. Representación de resultados

Tabla III. XXXIV: Representación del Indicador 4: Reutilización, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Representativo	Valor Calificativo
Herencia de clase base	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Herencia de plantillas	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Personalización de código heredado	☺☺☺	Suficiente	☺☺☺	Suficiente
Tamaño de la aplicación	☺☺☺	Suficiente	☺	Insuficiente

Indicador 5: Seguridad de la aplicación

La sección de Seguridad de la aplicación posee 4 índices cada uno de los cuales se realizó la valoración tomando en consideración la autenticación de un usuario, en la construcción del modulo 2 de pruebas. Sus resultados obtenidos serán evaluados tomando sus valores mínimos, máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 5.1: Variable uso de sesiones:** En este índice tendrá valorizaciones bajo la consideración del uso de variables de sesión, en base al modulo 2 de pruebas.

Tabla III. XXXV: Valorización del Índice 5.1: Variable uso de sesión

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 5.2: Manejo de cookies:** En este índice se valorizará la utilización de cookies para los controles de acceso a las paginas, en base al modulo 2 de pruebas.

Tabla III. XXXVI: Valorización del Índice 5.2: Manejo de cookies

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 5.3: Encriptación de datos:** En este índice se valorizará la seguridad en el método de encriptación utilizado por el framework para proteger las contraseñas, en base al modulo 2 de pruebas.

Tabla III. XXXVII: Valorización del Índice 5.3: Encriptación de datos

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 5.4: Validación de datos:** En este índice se valorizará la validación del usuario que se autentica en caso que sus datos sean incorrectos).

Tabla III. XXXVIII: Valorización del Índice 5.4: Validación de datos

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

3.5.7.5. Valoraciones

Tabla III. XXXIX: Resultados del Indicador 5: Seguridad de la aplicación

Indicadores	**Django**		**Ruby on Rails**	
	Valor Cualitativo	**Valor**	**Valor Cualitativo**	**Valor**

		Obtenido/20		Obtenido/20
Variable uso de sesiones	Muy Satisfactorio	19	Muy Satisfactorio	19
Manejo de cookies	Muy Satisfactorio	18	Muy Satisfactorio	19
Encriptación de datos	Muy Satisfactorio	19	Muy Satisfactorio	19
Validación de datos	Muy Satisfactorio	20	Muy Satisfactorio	19

3.5.7.6. Interpretación

Variable uso de sesion: Django permite utilizar un sistema de autenticación con propiedades generadas por el mismo framework, lo cual al momento de la creacion de usuarios y su respectiva autenticacion permite la utilizacion de variables de uso de seción con la identificacion del usuario autenticado, por lo que obtiene una calificacion de 19 puntos que equivale a Muy Satisfactorio. En RoR esto se lo hace de manera automática instalando y configurando componentes propios del framework (Device), lo cual se adapta perfectamente a la necesidad de el uso de sesiones en cualquier tipo de aplicación que necesite autenticación, lo mencionado anteriormente le da a RoR una puntuación de 19 equivalente a Muy Satisfactorio.

Manejo de cookies: La manipulación de las cookies ya son fijadas por el propio framework, lo cual permite a un navegador u otro saber si es el mismo usuario que esta realizando la peticion, en Django es increíblemente sencillo ya que cada objeto de petición tiene un objeto COOKIES que actúa como un diccionario lo cual lo podemos usar para leer cualquier cookie desde el navegador por lo que obtiene una calificacion de 18 que equivale a Muy Satisfactorio. Por otro lado en RoR esto es configurado por el mismo componente de autenticación, las cookies son manejadas por Action Controller, pero como se meciono, esto ya el framework lo hace de forma automática la califiación obtenida es de 19 equivalente a Muy Satisfactorio.

Encriptacion de datos: Debido a la importancia en la encriptacion de datos al momento de manejar informacion critica como las contraseñas, nos damos cuenta en la eficiencia de ambos frameworks, puesto a que ambos utilizan el metodo de encriptacion MD5, tanto Django asi como Ruby on Rails obtienen una valoracion de 19, que equivale a Excelente.

Validación de datos: Django permite validar sus datos automaticamente mediante la utilizacion de vistas propias del framework sin tener que crearlas de manera manual indicando mediante una plantilla el error en la validacion de los usuarios razones por la cual obtiene una calificacion de 20 puntos que equivale a Muy Satisfactorio. En este sentido RoR difiere ya que las validaciones las realiza en el Módelo, es aquí en donde se deben añadir lineas de código para realizar las validaciones las cuales funcionan muy bien por eso su calificación de 19, Muy Satisfactorio.

3.5.7.7. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

Cdj: $19 + 18 + 19 + 20 = 76$

$Cror$:19+19+19+19=76

Ct: $20 + 20 + 20 + 20 = 80$

Pdj: $\left(\frac{76}{80}\right) * 100\% = 95\%$

$$Pror: \left(\frac{76}{80}\right) * 100\% = 95\%$$

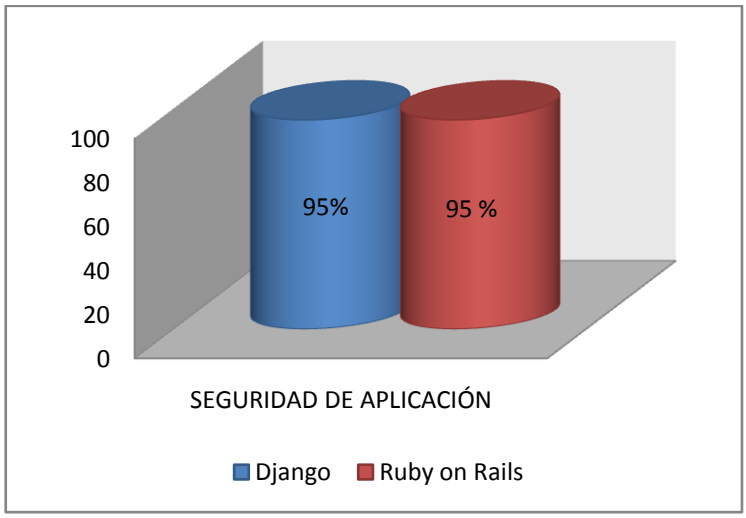


Figura III. 56: Resultado Final del Indicador 5: Seguridad de aplicación

3.5.7.8. Representación de Resultados

Tabla III. XL: Representación del Indicador 5: Seguridad de aplicación, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Representativo	Valor Calificativo
Variable uso de sesiones	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Manejo de cookies	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Encriptación de datos	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Validación de datos	☺☺☺☺	Excelente	☺☺☺☺	Excelente

Indicador 6: Madurez de producto

La Madurez del producto posee 4 índices cada uno de los cuales se considera la madurez del paquete de desarrollo en si como framework tomando en cuenta su valoración en base al modulo 3 de pruebas. Sus resultados obtenidos serán evaluados tomando sus valores mínimos, máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 6.1: Especificaciones:** En este índice tendrá valorizaciones consideran la compatibilidad con las demás herramientas, en base el modulo 3 de pruebas.

Tabla III. XLI: Valorización del Índice 6.1: Especificaciones

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 6.2: Licenciamiento:** En este índice se valorizará la clase de licenciamiento que posee el framework, en base al modulo 3 de pruebas.

Tabla III. XLII: Valorización del Índice 6.2: Licenciamiento

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio

16 a 20	Muy Satisfactorio

- **Índice 6.3: Costo:** En este índice se valorizará el costo que incurre el desarrollo de una aplicación web que será puesta en producción, en base al modulo 3 de pruebas.

Tabla III. XLIII: Valorización del Índice 6.3: Costo

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

3.5.7.9. Valoraciones

Tabla III. XLIV: Resultados del Indicador 6: Madurez del producto

Indicadores	**Django**		**Ruby on Rails**	
	Valor Cualitativo	**Valor Obtenido/20**	**Valor Cualitativo**	**Valor Obtenido/20**
Especificaciones	Muy Satisfactorio	17	Muy Satisfactorio	17
Licenciamiento	Muy Satisfactorio	19	Muy Satisfactorio	19
Costo	Muy Satisfactorio	20	Muy Satisfactorio	19

3.5.7.10. Interpretación

Especificaciones: Tanto el framework Django asi como Ruby on Rails son tecnologias multiplataformas con módulos abiertos lo que me permite obtener de acuerdo a las especificaciones de la tecnologia, una facil compatibilidad con demas herramientas por

lo que ambas tecnologias obtienen una calificacion de 17 puntos que equivale a Muy Satisfactorio.

Licenciamiento: La tecnologia Django asi como Ruby on Rails son herramientas de desarrollo software libre con licencias BSD y MIT respectivamente lo que nos representa un ahorro economico. Tanto la licencia BSD de Django, como la MIT de Ruby on Rails son de sismilares caracteristicas por lo que ambos obtienen una calificacion de 19 puntos.

Costo: Este indicador es muy importante ya que nos representa la rentabilidad en el desarrollo de un producto de software, Django y Ruby on Rails son herramientas de software libre con similar licenciamiento en lo que se diferencia es en su tiempo de implementacion por lo que Django presenta una ventaja sobre RoR, por lo que obtiene una calificacion de 20 puntos contra 19 puntos de RoR, que equivalen a Muy Satisfactorio.

3.5.7.11. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$$Cdj: 17 + 19 + 20 = 56$$

$$Cror: 17 + 19 + 19 = 55$$

$$Ct: 20 + 20 + 20 = 60$$

$$Pdj: \left(\frac{56}{60}\right) * 100\% = 93{,}33\%$$

$$Pror: \left(\frac{55}{60}\right) * 100\% = 91{,}66\%$$

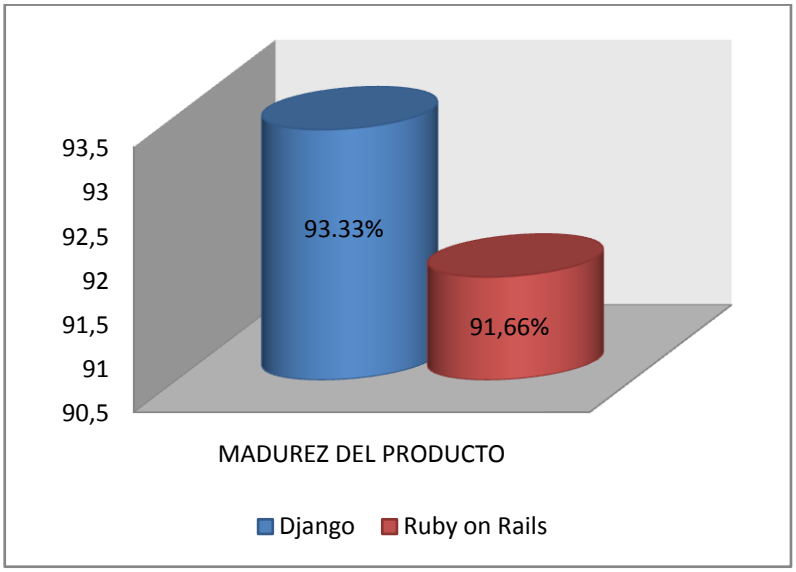


Figura III. 57: Resultado Final del Indicador 6: Madurez del producto

3.5.7.12. Representación de Resultados

Tabla III. XLV: Representación del Indicador 6: Madurez del producto, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Cualitativo	Valor Cuantitativo
Especificaciones	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Licenciamiento	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Costo	☺☺☺☺	Excelente	☺☺☺☺	Excelente

Indicador 7: Instalación

La instalación posee 3 índices cada uno de los cuales se considera la complejidad en l proceso de instalación y el entorno de desarrollo del framework, en base al ambiente de pruebas de instalación de las tecnologías, mediante la experimentación. Sus resultados

obtenidos serán evaluados tomando sus valores mínimos, máximos para cada índice, así realizar una escala de valorización y determinar los valores no satisfactorio, poco satisfactorio, satisfactorio y muy satisfactorio respectivamente.

- **Índice 7.1: Contenedor de aplicaciones:** En este índice se refiere a los servidores web en los que se puede levantar nuestra aplicación.

Tabla III. XLVI: Valorización del Índice 7.1: Especificaciones

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 7.2: Entorno de desarrollo:** En este índice se valorizará las herramientas IDE, como entornos de desarrollos amigables.

Tabla III. XLVII: Valorización del Índice 7.2: Entorno de desarrollo

Valoración	
Implementación/20	Valoración Cualitativa
1 a 5	No Satisfactorio
6 a 10	Poco Satisfactorio
11 a 15	Satisfactorio
16 a 20	Muy Satisfactorio

- **Índice 7.3: Tiempo de instalación:** En este índice se valorizará el tiempo empleado en el proceso de instalación del framework, definiendo su complejidad en mencionado proceso.

Tabla III. XLVIII: Valorización del Índice 7.3: Tiempo de instalación

Valoración		
Tiempo en minutos	Implementación/20	Valoración Cualitativa
>30	1 a 5	No Satisfactorio
21 a 30	6 a 10	Poco Satisfactorio
11 a 20	11 a 15	Satisfactorio
1 a 10	16 a 20	Muy Satisfactorio

3.5.7.13. Valoraciones

Tabla III. XLIX: Resultados del Indicador 7: Instalación

Indicadores	Django		Ruby on Rails	
	Valor Cualitativo	Valor Obtenido/20	Valor Cualitativo	Valor Obtenido/20
Contenedor de aplicaciones	Muy Satisfactorio	18	Muy Satisfactorio	18
Entorno de desarrollo	Muy Satisfactorio	19	Satisfactorio	15
Tiempo de instalación	Muy Satisfactorio	20	Poco Satisfactorio	10

3.5.7.14. Interpretación

Contenedor de Aplicaciones: La elección de los servidores web o los contenedores en donde se alojaran las aplicaciones web es muy importante para poner una aplicación en produccion, ambas tecnologias pueden ser implementadas bajo un servidor apache y sus variantes de acuerdo a que tecnologia sea (apache mod_python, apache mod_rails para cada tecnologia respectivamente, FastCGI, Lighttpd). Razones por las cuales amabas tecnologias obtienen una calificacion de 18 puntos que equivale a Muy Satisfactorio.

Entorno de Desarrollo: Existen multimples IDEs como entornos de desarrollo los cuales son muy bien utilizados por Django, debido a esa compatibilidad de entornos Django obtiene una calificacion de 19 puntos que equivale a Muy Satisfactorio. Por otra parte en RoR si bien existen varios IDEs los que mejor se adaptan a este framework son de pago, y el mejor de los gratuitos (NetBeans) la empresa propietaria ha dejado de seguir dando soporte para esta Tecnología, si bien aun existe todavía los plugins para su utilización a medida q vayan saliendo mas versiones de RoR este quedará obsoleto, razones que le dan una calificación de 15 equivalente a Satisfactorio.

Tiempo de Instalacion: Este indicador es fundamental debido a la facilidad que presenta cada tecnologia respecto a su instalacion, debido a que el tiempo y facilidad que presta Django siendo esta inferior a los 10 minutos obtiene una calificacion de 20 puntos que equivale a Muy Satisfactorio. Por otro lado Ruby on Rails presenta ciertas complejidades que incrementan el uso del tiempo en su instalacion siendo esta realizada en el rango de 21 a 30 minutos teniendo una calificacion de 10 puntos que equivale a Poco Satisfactorio.

3.5.7.15. Calificación

Cálculo de los porcentajes.

$$Cdj = \sum X$$

$$Cror = \sum Y$$

$$Ct = \sum W$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pror = \left(\frac{Cror}{Ct}\right) * 100\%$$

$Cdj: 18 + 19 + 20 = 57$

$Cror: 18 + 15 + 10 = 43$

$Ct: 20 + 20 + 20 = 60$

$$Pdj: \left(\frac{57}{60}\right) * 100\% = 95\%$$

$$Pror: \left(\frac{43}{60}\right) * 100\% = 71{,}66\%$$

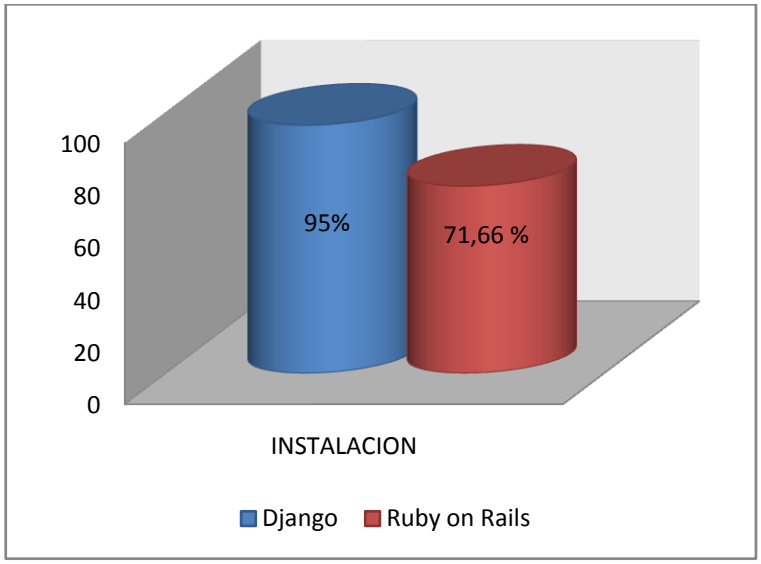


Figura III. 58: Resultado Final del Indicador 7: Instalación

3.5.7.16. Representación de Resultados

Tabla III. L: Representación del Indicador 7: Instalación, con sus diferentes Índices

Indicadores	Django		Ruby on Rails	
	Valor Representativo	Valor Calificativo	Valor Representativo	Valor Calificativo
Contenedor de aplicaciones	☺☺☺☺	Excelente	☺☺☺☺	Excelente
Entorno de desarrollo	☺☺☺☺	Excelente	☺☺☺	Suficiente
Tiempo de instalación	☺☺☺☺	Excelente	☺☺	Parcial

3.6. Puntajes Alcanzados

Después del análisis de los indicadores de productividad propuestos, mediante la aplicación de experimentación y observación en la construcción de los módulos de prueba, se obtuvieron como resultado valores cuantitativos que reflejan el desenvolvimiento de cada una de las tecnologías de acuerdo a sus características en función a la productividad de desarrollo. Estos valores son el resultado de las pruebas efectuadas tanto a los módulos de prueba 1, 2 y 3, tomando en consideración factores como el tiempo de desarrollo, implementación, tamaño de la aplicación, facilidad de uso.

Tabla III. LI: Cuadro Resultados de Indicador e Índice

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
Patrón de Diseño Modelo Vista Controlador	**Modelo**	• Soporte para múltiples bases de datos	4	4
		• Manipulación con la base de datos	4	4
		• Desempeño con la base de datos	3	4
		• Mapeador de Objeto Relacional ORM	4	4
	Vista	• Uso de plantillas	17	16
		• Adaptación con hojas de estilo	18	17
		• Fusión de código y diseño	19	17
	Controlador	• Generación automática de	19	18

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
		formularios		
		• Validación de formularios	20	19
		• Manejo visual de componentes de formulario	10	10
Principio DRY Don`t repeat yourself	**Reutilización**	• Herencia de clase base	18	19
		• Herencia de plantillas	20	19
		• Personalización de código heredado	17	17
		• Tamaño de la aplicación	20	10
Seguridad	**Seguridad de Aplicación**	• Variable uso de sesiones	19	19
		• Manejo de cookies	18	19
		• Encriptación de datos	19	19
		• Validación de datos	20	19
Producto	**Madurez de producto**	• Especificaciones	17	17
		• Licenciamiento	19	19
		• Costo	20	19
	Instalación	• Contenedor de	18	18

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
		aplicaciones		
		• Entorno de desarrollo	19	15
		• Tiempo de instalación	20	10
Totales			382	352

3.7. Diagrama General de Resultados

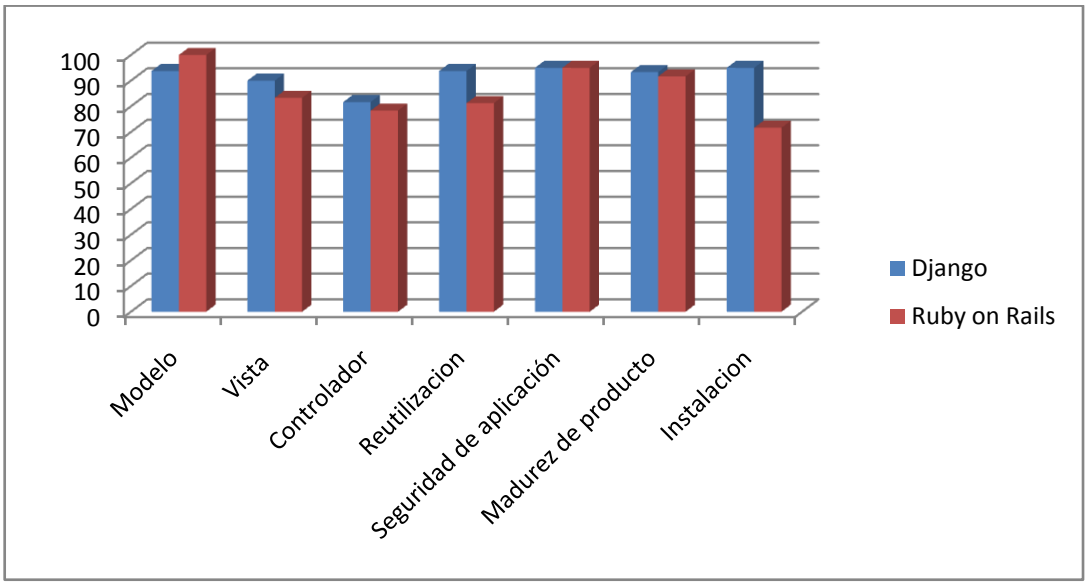


Figura III. 59: Grafica general de resultados

Tabla III. LII: Resultados generales

	Modelo	Vista	Controlador	Reutilización	Seguridad de	Madurez de producto	Instalación
Django	93,75	90	81,66	93,75	95	93,33	95
Ruby on Rails	100	83,33	78,33	81,25	95	91,66	71,66

3.8. Interpretación

Con la tabulación de los resultados obtenidos en la evaluación de los frameworks, se da a conocer de una manera global el comportamiento de cada tecnología respecto a la productividad en el desarrollo de aplicaciones.

El Modelo fue el encargado de probar la versatilidad con que cada tecnología se comporta al momento de actuar con los datos y los motores que gestionan estos datos, en donde se destaca Ruby on Rails mostrando un mejor comportamiento y ágil creación de los modelos.

La Vista fue el parámetro que pudo darnos una mejor perspectiva de la rápida creación de las interfaces de usuario y su comportamiento con las páginas creadas de manera estática, mostrando Django una ventaja ante Ruby on Rails manejando tecnologías muy similares.

El controlador nos permite enfocarnos en la fácil y rápida creación de formularios de una aplicación, las ventajas que nos brinda determinado framework en el control de los datos que se ingresan en los formularios en donde Django presenta virtudes que hacen que sobresalga ante Ruby on Rails.

La reutilización es un parámetro muy importante el mismo que es muy determinante en un desarrollo ágil de aplicaciones, mediante la valoración de los indicadores en la reutilización pudimos observar que Django es mejor en cuanto la reutilización tanto en la utilización de funciones como de código adaptándose de una mejor manera al principio DRY.

La seguridad a nivel de aplicación está definida por la depuración de los datos que son ingresados al momento de realizar una autenticación al sistema, y ya que ambos frameworks presentan características similares en el manejo de los indicadores de seguridad, tanto como Django así como Ruby on Rails permiten aprovechar las ventajas de una correcto sistema de autenticación y seguridad en el desarrollo de una aplicación.

La Madurez del producto nos permite identificar factores muy importantes que son fundamentales al momento de un desarrollo ágil, los mismos que al final del desarrollo

se traducen en una rentabilidad que genere el desarrollo con una u otra tecnología presentando Django características que permiten sobresalir frente a Ruby on Rails.

Un proceso de instalación define la versatilidad de una herramienta y describe un mejor concepto en la utilización e implementación de la misma. Django presenta una mejor adaptación con entornos de desarrollo y mas optimo proceso de instalación destacándose sobre Ruby on Rails.

3.9. Análisis de Resultados

Los resultados que presenta cada framework al establecer mismos criterios y escenarios de desarrollo son claros ya que ambas tecnologías presentan características similares se puede establecer el framework que se adapta de mejor manera a un desarrollo más ágil de aplicaciones web.

Luego de la interpretación de los resultados realizado anteriormente se destaca el Frameworks Django para realizar un desarrollo más productivo, ideal para desarrollar el sistema para el control de docentes contratados de la Dirección de Estudios de Chimborazo con los objetivos planteados al principio de la tesis, ya que Django presenta grandes características en la conexión de base de datos con soporte para varias plataformas. Formularios y validación de datos, manejo de sesiones y un más óptimo proceso de instalación y adaptación con los entornos de desarrollo.

En conclusión por tener características que permiten distinguir una ventaja sobre Ruby on Rails, concluimos que el Framework Django es el más adecuado para un desarrollo más ágil de la aplicación web.

3.10. Comprobación de Hipótesis

La hipótesis planteada es:

H1: La aplicación de la tecnología Python con Django mejora la productividad en el desarrollo ágil de aplicaciones web que al usar la tecnología Ruby on Rails.

3.10.4. Operacionalización de variables

Operacionalización Conceptual

Tabla III. LIII: Operacionalización conceptual

VARIABLE	TIPO	CONCEPTO
Frameworks	Dependiente Compleja Cualitativa	Según Javier J. Gutiérrez [13], El concepto framework se emplea muchos ámbitos del desarrollo de sistemas software, no solo en el ámbito de aplicaciones Web. Podemos encontrar frameworks para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier ámbito que pueda ocurrírsenos.
Productividad	Dependiente Compleja Cualitativa	Según la INES [15], En el terreno de las metodologías de desarrollo de software, se aprecia una necesaria mejora en la puesta en práctica de dichas metodologías de desarrollo, así como la flexibilización de éstas para potenciar la productividad de las mismas sin renunciar a la calidad de los mismos. La mejora de la efectividad y la productividad en el desarrollo de software está ligada a la utilización de buenas prácticas de Ingeniería de Software.

Operacionalización Metodológica

Tabla III. LIV: Operacionalización Metodológica

VARIABLE	CATEGORIA	INDICADORES	TÉCNICA	FUENTE DE VERIFICACION
VARIABLE DEPENDIENTE FRAMEWORKS	**FRAMEWORKS**	DJANGO RUBY ON RAILS	Observación Directa	Web, Aptana Studio
	APLICACIÓN	APLICACIÓN		
VARIABLE DEPENDIENTE PRODUCTIVIDAD	**MODELO**	Soporte para múltiples bases de datos	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas
		Manipulación con la base de datos	Observación Experimentación	Web, Aptana Studio, Módulo 1 de pruebas
		Desempeño con la base de datos	Observación Experimentación	Web, Aptana Studio, Módulo 1 de pruebas
		Mapeador de objeto relacional	Observación Experimentación	Web, Apatana Studio, ORM, Módulo 1 de pruebas
	VISTA	Uso de plantillas	Observación Experimentación	Web, Aptana Studio, Módulo 1 de pruebas
		Adaptación con hojas de estilo	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas

VARIABLE	CATEGORIA	INDICADORES	TÉCNICA	FUENTE DE VERIFICACION
		Fusión de código y diseño	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas
	CONTROLADOR	Generación automática de formularios	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas
		Validación de formularios	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas
		Manejo visual de componentes de formulario	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 de pruebas
	REUTILIZACION	Herencia de clase base	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 y 2 de pruebas
		Herencia de plantillas	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 y 2 de pruebas
		Personalización de código heredado	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 y 2 de pruebas

VARIABLE	CATEGORIA	INDICADORES	TÉCNICA	FUENTE DE VERIFICACION
		Tamaño de la aplicación	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 1 y 2 de pruebas
	SEGURDAD DE APLICACION	Variable uso de sesiones	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 2 de pruebas
		Manejo de cookies	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 2 de pruebas
		Encriptación de datos	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 2 de pruebas
		Validación de datos	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 2 de pruebas
	MADUREZ DE PRODUCTO	Especificaciones	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 3 de pruebas
		Licenciamiento	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 3 de pruebas
		Costo	Observación	Web, Aptana,

VARIABLE	CATEGORIA	INDICADORES	TÉCNICA	FUENTE DE VERIFICACION
			Experimentación	Documentación, Internet, Módulo 3 de pruebas
	INSTALACION	Contenedor de aplicaciones	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 3 de pruebas
		Entorno de desarrollo	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 3 de pruebas
		Tiempo de instalación	Observación Experimentación	Web, Aptana, Documentación, Internet, Módulo 3 de pruebas

3.10.5. Tabla de valores finales obtenidos.

Tabla III. LV: Resultado Finales

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
Patrón de Diseño Modelo Vista Controlador	**Modelo**	• Soporte para múltiples bases de datos	4	4
		• Manipulación con la base de datos	4	4
		• Desempeño con la base de datos	3	4

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
		• Mapeador de Objeto Relacional ORM	4	4
	Vista	• Uso de plantillas	17	16
		• Adaptación con hojas de estilo	18	17
		• Fusión de código y diseño	19	17
	Controlador	• Generación automática de formularios	19	18
		• Validación de formularios	20	19
		• Manejo visual de componentes de formulario	10	10
Principio DRY Don`t repeat yourself	**Reutilización**	• Herencia de clase base	18	19
		• Herencia de plantillas	20	19
		• Personalización de código heredado	17	17
		• Tamaño de la aplicación	20	10
Seguridad	**Seguridad de Aplicación**	• Variable uso de sesiones	19	19
		• Manejo de	18	19

CLASIFICACION	PARAMETROS	INDICADORES	Django	RoR
		cookies		
		• Encriptación de datos	19	19
		• Validación de datos	20	19
Producto	**Madurez de producto**	• Especificaciones	17	17
		• Licenciamiento	19	19
		• Costo	20	19
	Instalación	• Contenedor de aplicaciones	18	18
		• Entorno de desarrollo	19	15
		• Tiempo de instalación	20	10
Totales			382	352

Tabla III. LVI: Valores y Porcentajes Finales

	TOTAL	
	Valor	Porcentaje
Django	382	91,82
Ruby o Rails	352	84,61

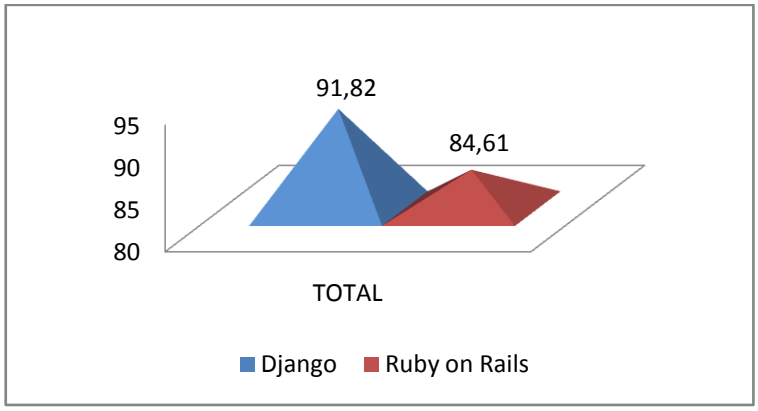


Figura III. 60: Resultado Final del Análisis

Haciendo referencia a la tabla III.LVI y por observación directa se concluye que en la hipótesis planteada: ***"La aplicación de la tecnología Python con Django mejora la productividad en el desarrollo ágil de aplicaciones web que al usar la tecnología Ruby on Rails."***. Django posee el valor más alto que es 382 puntos sobre 352 de Ruby o Rails, valor máximo posible, superándolo en un 7,21%, Django mejora la productividad en el desarrollo ágil en un 91,82% que equivale a Muy bueno. Por lo tanto se concluye que la hipótesis H1 es verdadera.

CAPÍTULO IV:

DESARROLLO DEL SISTEMA DE CONTROL DE DOCENTES CONTRATADOS DE LA DIRECCION DE ESTUDIOS CHIMBORAZO

En la actualidad en el desarrollo de aplicaciones web no se consideran aspectos como robustez, agilidad y confiabilidad que son indispensables en el momento en que las aplicaciones web se encuentran operativas, por lo que estas aplicaciones son susceptibles a fallos e incluso pueden impedir hacer modificaciones cuando se encuentren en operación; acarreando como problema final que el sistema sea desechado por el usuario final, para evitar esto se procede a diseñar la aplicación web cumpliendo con lo requerido por la DECH a la vez que cumpliendo con el alcance y objetivos planteados.

Para resolver el problema de la contratación docente, se diseña un sitio web, en conjunto con normas e instructivos para el uso de la misma, y así dar facilidad de información y registro de docentes y sus contratos.

La función principal de la aplicación web será registrar los contratos realizados por DECH a los docentes, y demás funciones que se derivan de esta, como registro de

docentes, instituciones, etc. Todo esto con una interfaz amigable, con las debidas restricciones de seguridad y privacidad.

4.1. Ingeniería de la Información

La metodología usada para el desarrollo de la aplicación web es SCRUM, una metodología que permite la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.

El modelo de desarrollo Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto.

Scrum, más que una metodología de desarrollo software, es una forma de auto-gestión del equipo de programadores. El grupo de programadores decide cómo hacer sus tareas y cuánto van a tardar en ello (Sprint). Scrum ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro

4.1.1. Definición del Ámbito

La Dirección de Educación de Chimborazo, a través de la División de Recursos Humanos, se encuentra en proceso de contratación docente para cubrir los distintos requerimientos de las instituciones educativas de la provincia. El procesamiento de la información de los docentes e instituciones educativas se lo realiza de forma manual, mediante la recopilación de documentos escritos y el manejo de tablas de Excel, enfrentando problemas de dificultad, lentitud, desorganización e inconsistencias de datos, los cuales imposibilitan la optimización del tiempo al momento de realizar la contratación de los docentes.

Se necesita crear una solución informática que permitirá automatizar el proceso de contratación docente, el que conste con una interfaz amigable con el usuario, estableciendo políticas de acceso para el ingreso al sistema, además tener un control de la información de los docentes los cuales están y serán contratados por la Dirección. La

aplicación web facilitará la realización de reportes, los mismos que detallaran información de los docentes, las escuelas y la ubicación geográfica donde se encuentran laborando. Esto podrá facilitar a los analistas de recursos humanos tener un control más óptimo al momento de ejecutar nuevos contratos, renovaciones, terminaciones y renuncias de los maestros lo cual establecerá una mejor administración del personal docente, y la distribución hacia las distintas instituciones educativas.

Los roles implicados para el desarrollo de este sistema siguiendo Scrum serán:

Comprometidos:

- **Propietario del producto:** Ing. Walter Mejía, responsable de obtener el mayor valor de producto para los usuarios y resto de implicados.
- **Equipo de desarrollo:** Paul Cumba, Byron Barreno que es el grupo de trabajo que se encarga del desarrollo del producto.
- **Scrum Manager:** Ing. Danilo Pastor, gestor del equipo, es responsable del funcionamiento y de la productividad del equipo de desarrollo.

Implicados:
Asesores de contratación del área de talento humano de la DECH.

Tipos de Usuarios y Permisos de Acceso: Los Usuarios del sitio web tendrá la siguiente categoría:

- Usuario Administrador.
- Usuarios de Contratación

El usuario Administrador, es el encargado de la administración del sitio con todos los privilegios para realizar cualquier tipo de modificación de los datos, además es el único con acceso al sitio de administración del sistema.

Los usuarios de Contratación, son los encargados de realizar los contratos, con la posibilidad de ingresar, editar los datos de docentes e instituciones.

Módulos del sistema web: Tendrá 4 módulos principales
Módulo de acceso: permite a los usuarios iniciar sesión, de esta manera restringir el acceso según los privilegios de usuario.
Módulo de ingresos: en este módulo se realizan tareas de crear, editar, y eliminar docentes e instituciones así como la asignar los contratos de un docente a una o a varias instituciones.
Módulo de consultas: dispone de las consultas, que se van a realizar hacia la base de datos, en modo de reportes imprimibles.
Módulo de contenido estático: posee información de la aplicación, tal como el acerca de, etc.

4.1.2. Estudio de Factibilidad

4.1.2.1. Factibilidad Técnica

- **Recurso Humano**

Existe el recurso humano capacitado para realizar la administración del sistema, obteniendo una operación garantizada y uso garantizado.

- **Recurso Hardware**

✓ Intel Core 2 Duo 2.1Ghz
✓ DVD-RW
✓ Teclado, Mouse, Monitor

- **Recurso Software**

✓ *Sistema Operativo:* Microsoft Windows Seven Ultimate / Linux Ubuntu

✓ *Base de Datos:* Postgresql

✓ *Servidor Web:* Apache

✓ *Tecnología de Desarrollo:* IDE Aptana Studio 3, framework Django 1.3.1 para leguaje de programación Python 2.4.3.

4.1.2.2. Factibilidad Operativa

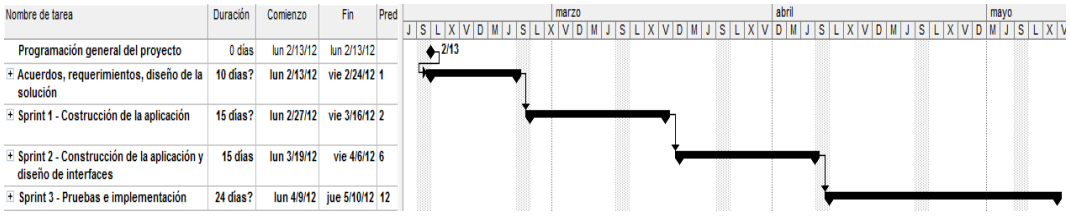
Encargado del departamento de Sistemas en la DECH
Usuarios del departamento de Recursos humanos.

4.1.2.3. Factibilidad Legal

Existe la autorización de las autoridades respectivas por lo que no existe ningún tipo de impedimento legal para el desarrollo del sistema

4.1.3. Agenda del proyecto según SCRUM

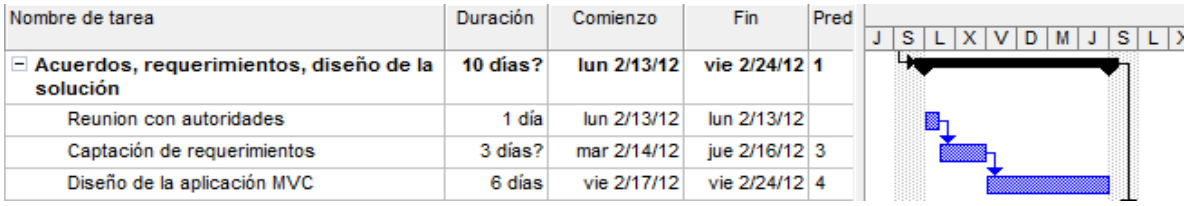
Se presenta la planificación general de los Sprint planificados inicialmente y su duración



Nombre de tarea	Duración	Comienzo	Fin	Pred
Programación general del proyecto	0 días	lun 2/13/12	lun 2/13/12	
Acuerdos, requerimientos, diseño de la solución	10 días?	lun 2/13/12	vie 2/24/12	1
Sprint 1 - Costrucción de la aplicación	15 días?	lun 2/27/12	vie 3/16/12	2
Sprint 2 - Construcción de la aplicación y diseño de interfaces	15 días	lun 3/19/12	vie 4/6/12	6
Sprint 3 - Pruebas e implementación	24 días?	lun 4/9/12	jue 5/10/12	12

Figura IV. 1: Planificación previa de desarrollo.

En esta etapa se definirán los acuerdos estableciendo la delimitación del sistema y sus requerimientos, según la complejidad del problema se establecerán los productos entregables (sprint) y sus resectivas fechas de entrega, hasta el producto final.



Nombre de tarea	Duración	Comienzo	Fin	Pred
Acuerdos, requerimientos, diseño de la solución	**10 días?**	**lun 2/13/12**	**vie 2/24/12**	**1**
Reunion con autoridades	1 día	lun 2/13/12	lun 2/13/12	
Captación de requerimientos	3 días?	mar 2/14/12	jue 2/16/12	3
Diseño de la aplicación MVC	6 días	vie 2/17/12	vie 2/24/12	4

Figura IV. 2: Actividades del Sprint 1

4.1.3.1. Especificación de actividades del ProductBacklog

Después de haber captado lo requerido para el sistema por parte de la Dirección Hispana de Educación de Chimborazo (DECH), se ha planteado que:

Para el desarrollo de la aplicación web, se han establecido tres productos entregables ***(sprint),*** de los cuales el primero comprenderá la construcción de la aplicación MVC, con los métodos de ingreso, actualización, y eliminación.

El segundo Sprint, incluye la depuración de la aplicación corrigiendo falencias del primer producto entregado, e incorporar plantillas HTML que utilizan estilos CSS que mejoren la apariencia de la Aplicación.

El tercer y último sprint, al igual que el anterior empieza por corregir falencias de los productos anteriores, la puesta en entorno de producción de la aplicación, pruebas de la aplicación, nuevas correcciones y/o modificaciones, y finalmente la publicación definitiva de la aplicación en la web.

Tabla IV. I: PRODUCT BACKLOG - SISCONDECH.

ID	Tarea
1	**SPRINT 1**
1.1	Inicio del proyecto
1.1.1	Creación del nuevo proyecto
1.2	Construcción de la aplicación MVC
1.2.1	Construcción del módulo de Acceso (login)
1.2.2	Construcción del módulo de ingresos
1.2.2	Construcción del módulo de consultas
1.2.2	Construcción del módulo de contenido estático
2	**SPRINT 2**
2.1	Depuración del producto entregado en el Sprint 1
2.2	Continuación de la construcción de la aplicación MVC
2.2.1	Implementación de método buscar para docentes e instituciones para la contratación

ID	Tarea
2.2.2	Implementación autocompletado de docentes e instituciones para la contratación
2.2.2	Mejora de las Restricciones en el modelo
2.3	Diseño de las interfaces
3	**SPRINT 3**
3.1	Depuración de la aplicación
3.2	Publicación en WebHosting
3.3	Pruebas del Sistema
3.4	Corrección de posibles errores
3.5	Publicación final

Planificación Sprint Backlog 1

La planificación correspondiente al Sprint 1 tiene una duración de 15 días,el cual se inicia el día lunes 27 de febrero y finaliza el día viernes16 de marzo del presente año.

Figura IV. 3: Planificación Sprint Backlog 1

Tabla IV. II: SPRINT BACKLOG 1 - Construcción de la aplicación

ID	Tarea	Tipo	Estado	Responsable
1	**Inicio del proyecto**			
1.1	**Creación del nuevo proyecto**	Desarrollo	Finalizado	Paul Cumba
1.2	Conexión con	Base de datos / Desarrollo	Finalizado	Paul Cumba

ID	Tarea	Tipo	Estado	Responsable
	Base de Datos			
1.2.1	Crear Base de Datos Postgresql	Base de datos	Finalizado	Byron Barreno
1.2.2	Crear usuario para la base de datos	Base de datos	Finalizado	Byron Barreno
2	**Construcción de la aplicación MVC**		Finalizado	
2.1	**Construcción del módulo de Acceso (login)**		Finalizado	
2.1.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.1.2	Agregación al modelo de la clase Users	Desarrollo	Finalizado	Paul Cumba
2.1.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.1.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.1.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo**		Finalizado	

ID	Tarea	Tipo	Estado	Responsable
	de ingresos			
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Agregación al modelo para Docentes, Instituciones, Contratos, y demás tablas para la BD	Desarrollo	Finalizado	Paul Cumba
2.2.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo de consultas**		Finalizado	
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Agregación al modelo según las clases para las consultas	Desarrollo	Finalizado	Paul Cumba
2.2.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba

ID	Tarea	Tipo	Estado	Responsable
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo de contenido estático**		Finalizado	
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Edición del modelo para el contenido estático (Home, help, about_us).	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2.2.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno

Planificación Sprint Backlog 2

La planificación correspondiente al Sprint 2 tiene una duración de 15 días,el cual se inicia el día lunes 27 de febrero y finaliza el día viernes16 de marzo del presente año.

Nombre de tarea	Duración	Comienzo	Fin	Predo
Sprint 2 - Construcción de la aplicación y diseño de interfaces	**15 días**	**lun 3/19/12**	**vie 4/6/12**	**6**
Depuración de Sprint 1	5 días	lun 3/19/12	vie 3/23/12	
Implementación de funciones (Autocompletado, restricciones)	5 días	lun 3/26/12	vie 3/30/12	13
Integración con plantillas html y estilos css	5 días	lun 4/2/12	vie 4/6/12	14

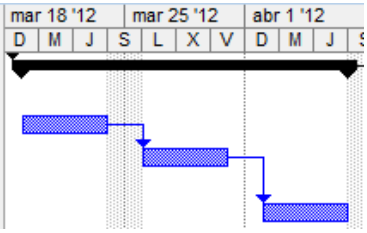


Figura IV. 4: Planifiación Sprint Backlog 2

Tabla IV. III: SPRINT BACKLOG 2 Construcción de la aplicación y diseño de interfaces

ID	Tarea	Tipo	Estado	Responsable
1	**Depuración del producto entregado en el Sprint 1**	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2	**Construcción de la aplicación MVC**			
2.1	**Implementación de un buscar para docentes e instituciones para la contratación**		Finalizado	
2.1.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.1.2	Edición del modelo	Desarrollo	Finalizado	Paul Cumba
2.1.3	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Implementación autocompletado de docentes e instituciones para la contratación**			
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Edición del modelo	Desarrollo	Finalizado	Paul Cumba
2.2.3	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno

ID	Tarea	Tipo	Estado	Responsable
2.3	Mejora de las Restricciones en el modelo			
2.3.1	Edición del modelo	Desarrollo	Finalizado	Paul Cumba
3	**Diseño de las interfaces**			
3.1	Implementación de hojas de estilo	Diseño de Interfaz	Finalizado	Byron Barreno
3.2	Creación de la plantilla base	Diseño de Interfaz	Finalizado	Byron Barreno
3.3	Integración de la plantilla HTML y sus estilos, con la aplicación	Diseño de Interfaz	Finalizado	Byron Barreno
3.4	Mejora de detalles en la interfaz (Renderización, banner, imágenes, etc)	Diseño de Interfaz	Finalizado	Byron Barreno

Planificación Sprint Backlog 3

La planificación correspondiente al Sprint 3 tiene una duración de 24 días, este corresponde ya al producto final a ser entregado, el cual se inicia el día lunes 9 de abril y finaliza el día jueves 10 de mayo del presente año.

Nombre de tarea	Duración	Comienzo	Fin	Pred
Sprint 3 - Pruebas e implementación	**24 días?**	**lun 4/9/12**	**jue 5/10/12**	**12**
Depuración de Sprint 2	5 días?	lun 4/9/12	vie 4/13/12	
Publicación en Web Hosting	2 días	lun 4/16/12	mar 4/17/12	17
Pruebas del sistema	10 días	mié 4/18/12	mar 5/1/12	18
Depuración de errores	15 días	mié 4/18/12	mar 5/8/12	18
Publicación final de la aplicación	2 días	mié 5/9/12	jue 5/10/12	20

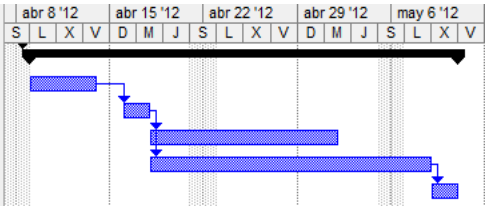


Figura IV. 5: Planificación Sprint Backlog 3

Tabla IV. IV: SPRINT BACKLOG 3 Pruebas e implementación de la aplicación

ID	Tarea	Tipo	Estado	Responsable
1	**Depuración de la aplicación (Sprint 2)**			

ID	Tarea	Tipo	Estado	Responsable
1.1	Personalización de formularios	Desarrollo	Finalizado	Paul Cumba
1.2	Restricciones de acceso a la aplicación	Desarrollo	Finalizado	Paul Cumba
1.3	Inserción de registros reales (Docentes, Instituciones, Contratos, etc.)	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2	**Publicación en WebHosting**			
2.1	Elección del proveedor Hosting	Redes	Finalizado	Byron Barreno
2.2	Configuración del servidor de BD y servidor Web.	Sistema Operativo	Finalizado	Byron Barreno
2.3	Respaldo de la BD	Base de Datos	Finalizado	Paul Cumba
2.4	Restauración de la BD en el servidor remoto	Base de Datos	Finalizado	Byron Barreno
2.5	Respaldo de la Aplicación.	Desarrollo	Finalizado	Paul Cumba
2.6	Carga de la aplicación hacia el servidor remoto	Desarrollo	Finalizado	Byron Barreno
2.7	Configuración de parámetros en la aplicación, para el funcionamiento en el Hosting	Desarrollo	Finalizado	Paul Cumba
2.8	Adquisición de dominio	Redes	Finalizado	Byron Barreno
3	**Pruebas del Sistema**			
3.1	Carga masiva de registros	Utilización del Sistema	En curso	Usuarios del sistema

ID	Tarea	Tipo	Estado	Responsable
3.2	Edición y eliminación de registros	Utilización del Sistema	En curso	Usuarios del sistema
3.3	Generación de Reportes	Utilización del Sistema	En curso	Usuarios del sistema
4	**Corrección de posibles errores**	Desarrollo	Finalizado	Byron Barreno
5	**Publicación final**	Desarrollo	Finalizado	Byron Barreno

4.1.4. Elementos del Proyecto

Pila del Producto Requerimientos

4.1.4.1. Requerimientos Funcionales

Req 1. La aplicación debe permitir su uso mediante la web.

Req 2. El sistema debe permitir la autenticación de usuarios, diferenciando los permisos sobre los objetos de la aplicación.

Req 3. El sistema debe permitir ingresar la información de docentes.

Req 4. El sistema debe permitir consultar la información de docentes.

Req 5. El sistema debe permitir actualizar la información de docentes.

Req 6. El sistema debe permitir ingresar la información de instituciones.

Req 7. El sistema debe permitir consultar la información de instituciones.

Req 8. El sistema debe permitir actualizar la información de instituciones.

Req 9. El sistema debe permitir ingresarla información de un nuevo contrato.

Req 10. El sistema debe permitir consultar la información de los contratos.

Req 11. El sistema debe permitir actualizar la información de los contratos.

Req 12. El sistema debe permitir consultar la información de los docentes con contratos vigentes.

Req 13. El sistema debe permitir consultar la información de los docentes con contratos finalizados.

Req 14. El sistema debe permitir consultar la información de los docentes contratos por año.

Req 15. El sistema debe permitir consultar la información de instituciones por ubicación geográfica.

Req 16. El sistema debe permitir consultar la información de los contratos.

Req 17. El sistema debe permitir consultar la información de los contratos vigentes.

Req 18. El sistema debe permitir consultar la información de los contratos inactivos.

Req 19. El sistema debe permitir consultar la información de los contratos finalizados.

4.1.4.2. Requerimientos No Funcionales

A continuación se muestran los requerimientos no funcionales más relevantes del software con sus respectivas características:

- **Rendimiento**

- Tiempos de respuesta al, abrir una página web.

- **Fiabilidad**

- Capacidad para tolerar errores en un 90%.

- Capacidad para tolerar sobrecargas en el volumen de información, de usuarios o de procesos en un 80%.

- **Disponibilidad**

- El sistema estará funcionando 24 horas al día los 365 días del año.

- **Seguridad**

- El sistema tendrá un formulario de autenticación.
- Uso de sesiones de usuario.

- **Mantenibilidad**

- Documentación del diseño y de la codificación de la solución.

- **Escalabilidad**

- Diseño de la arquitectura empleando MVC

- **Reusabilidad**

- Uso de estándares en los formatos para los datos.

- **Interfaces**

- Interfaces web para carga de archivos de entrada/salida.
- Intuitivas y amigables las interfaces

4.2. Análisis del Sistema

4.2.1. Definir Casos de Uso esenciales en formato extendido

CASO DE USO AUTENTIFICACIÓN

Tabla IV. V Caso de Uso Autenticación

IDENTIFICAR CASO DE USO:	CU_Autentificación
NOMBRE DEL CASO DE USO	Autentificación de Usuarios
ACTORES	Personal de RRHH (Usuarios)
PROPÓSITO	Realizar el proceso de identificación de los empleados de RRHH que utilizan el sistema
VISIÓN GENERAL	El usuario requiere autenticarse en el sistema; ingresa su nombre de usuario y contraseña y de acuerdo a la función que

	desempeñe el usuario se abrirá la pantalla correspondiente.
TIPO	Primario, real y expandido
REFERENCIAS	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DE LOS ACTORES	**RESPUESTA DEL SISTEMA**
1. Este caso de uso inicia cuando el usuario desea autenticarse para realizar la función que le compete	2. Muestra pantalla para que el usuario ingrese los datos respectivos
3. El usuario ingresa su Usuario y Contraseña	4. Valida los datos ingresados
	5. Presenta una ventana con un menú en el cual el usuario podrá realizar las funciones asignadas.
CURSOS ALTERNATIVOS	
2.1.- El sistema no presenta pantalla para ingreso de datos	
Mensaje de error 4.1.- "Usuario o contraseña incorrectos" 4.2.- No existe el usuario con esa sesión Mensaje de advertencia 4.3.- "Ingrese su sesión" 4.4.- "Ingresar Clave" 4.5.-"Clave Incorrecta"	

CASO DE USO CREAR USUARIOS

Tabla IV. VI Caso de Uso Crear Usuario

IDENTIFICAR CASO DE USO:	CU_Crear_Usuario
NOMBRE DEL CASO DE USO:	Crear_Usuario
ACTORES:	Administrador
PROPÓSITO:	Cambiar la información de algún usuario especifico

VISIÓN GENERAL:	Se realiza el registro de las personas que podrán tener acceso para ingresar al sistema
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando usuario se autentifica en modo Administración Dango	2.-Muestra pantalla de administración del sitio.
3.-Ingresa al registro de usuarios	
4.- Llena el formulario con información del usuario	
5.- Asigna los permisos al usuario sobre los objetos.	6.- Valida información
7.- Guarda cambios	7.- Crea el nuevo registro en la Base de datos
CURSOS ALTERNATIVOS	
Mensajes de error : 4.1 “Por favor, corrija los siguientes errores.”	
Mensaje de Información: 13.1 “Usuario creado con éxito”	

CASO DE USO CAMBIAR DATOS USUARIO

Tabla IV. VII Caso de Uso Cambiar Datos de Usuario

IDENTIFICAR CASO DE USO:	CU_Cambiar_Datos_Usuario
NOMBRE DEL CASO DE USO:	Cambiar _ Datos_Usuario
ACTORES:	Administrador
PROPÓSITO:	Cambiar la información de algún usuario especifico
VISIÓN GENERAL:	Los usuarios podrán realiza el cambio de información excepto de sus datos principales

TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando usuario se autentifica en modo Administración Dango	2.-Muestra pantalla para autentificación de usuario
3.-Ingresa sus datos (sesión y clave)	4.- Valida datos del usuario
	5- Muestra pantalla principal correspondiente a los permisos de dicho usuario
6.- Selecciona la opción Cambio de datos del usuario	
7.- Cambia datos	
8.-Registra cambio utilizando un botón de la interfaz asignado.	9.- Actualiza los cambios en la base de datos del sistema.
	10. Muestra mensaje confirmando que la actualización se ha realizado.
CURSOS ALTERNATIVOS	
Mensajes de error : 4.1 “No existe usuario” 4.2 “Ingrese correctamente los datos”.	
Mensaje de Información: 13.1 “Los datos fueron actualizados”	

CASO DE USO CREACION DE DOCENTES

Tabla IV. VIII Caso de Uso Creación de Docentes

IDENTIFICAR CASO DE USO:	CU_Crear_Docente
NOMBRE DEL CASO DE USO:	Crear_Docente
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar el registro de una determinado

	Docente
VISIÓN GENERAL:	El proceso inicia cuando una nueva contratación se va a dar y el docente no se encuentra en la base de datos, entonces se necesita registrar un nuevo docente, para lo cual el usuario debe ingresar a los ingresos de docentes y llenar el formulario.
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos

CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando un nuevo docente va a ser contratado se acerca a Administrador a solicitar el registro de su información.	
2.- Administrador solicita documentos para poder realizar la respectiva creación de cuenta.	
3.- Docente entrega documentos y entrega el Numero de Cedula, especialidad.	4.-Muestra formulario de ingreso docente
5.- Administrador llena los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado el registro correctamente.

CURSOS ALTERNATIVOS
Mensajes de error : 6.1 “Los campos marcados con asterisco son de ingreso obligatorio” 6.2 “Ingrese correctamente el número de cedula”
Mensajes de Advertencia: 7.1 “No se puede realizar la creación del docente”
Mensaje de Información:

7.2 “Docente ingresado correctamente”

CASO DE USO EDICIONDE DOCENTES

Tabla IV. IX Caso de Uso Edición de Docentes

IDENTIFICAR CASO DE USO:	CU_Editar_Docente
NOMBRE DEL CASO DE USO:	Editar_Docente
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar la edición de un registro de un determinado Docente
VISIÓN GENERAL:	El proceso inicia cuando requiere agregar datos de un docente ya existente, o cambiar datos del mismo
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando la información de un docente está incompleta o con algún error de tipiado.	
2.- Administrador solicita los datos a corregir o completar.	
3.- Administrador o usuario ingresa al módulo de edición de registros.	4.-Muestra formulario con la información actual de docente
5.- Administrador llena y/o cambia los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado el registro correctamente.
CURSOS ALTERNATIVOS	

Mensajes de error : 6.1 “Los campos marcados con asterisco son de ingreso obligatorio” 6.2 “Ingrese correctamente el número de cedula”
Mensajes de Advertencia: 7.1 “No se puede realizar la edición del docente”
Mensaje de Información: 7.2 “Actualización exitosa”

CASO DE USO CREACION DE INSTITUCIONES

Tabla IV. X Caso de Uso Creación de Instituciones

IDENTIFICAR CASO DE USO:	CU_Crear_Institucion
NOMBRE DEL CASO DE USO:	Crear_Institucion
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar el registro de una determinada Institución
VISIÓN GENERAL:	El proceso inicia cuando una Institución requiere de personal docente, y esa institución no se encuentra en la Base de Datos.
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando una nueva institución se necesita ingresar al sistema.	
2.- Administrador solicita datos de la institución	
3.- Administrador o usuario ingresa al	4.-Muestra formulario de ingreso

módulo de ingreso de instituciones.	instituciones.
5.- Administrador o usuario llena los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado el registro correctamente.
CURSOS ALTERNATIVOS	
Mensajes de error : 6.1 "Los campos marcados con asterisco son de ingreso obligatorio"	
Mensajes de Advertencia: 7.1 "No se puede realizar la creación de la institución"	
Mensaje de Información: 7.2 "Institución ingresada correctamente"	

CASO DE USO EDICIONDE INSTITUCIONES

Tabla IV. XI Caso de Uso Edición de Instituciones

IDENTIFICAR CASO DE USO:	CU_Editar_Institución
NOMBRE DEL CASO DE USO:	Editar_Institución
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar la edición de un registro de determinada Institución
VISIÓN GENERAL:	El proceso inicia cuando se requiere cambiar datos de una institución determinada.
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**

1.- Se inicia cuando la información de un docente está incompleta o con algún error de tipiado.	
2.- Administrador solicita los datos a corregir o completar.	
3.- Administrador o usuario ingresa al módulo de edición de registros.	4.-Muestra formulario con la información actual de instituciones
5.- Administrador llena y/o cambia los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado el registro correctamente.
CURSOS ALTERNATIVOS	
Mensajes de error : 6.1 “Los campos marcados con asterisco son de ingreso obligatorio”	
Mensajes de Advertencia: 7.1 “No se puede realizar la edición de la institución”	
Mensaje de Información: 7.2 “Actualización exitosa”	

CASO DE USO INGRESO DE CONTRATOS

Tabla IV. XII Caso de Uso Ingreso de Contratos

IDENTIFICAR CASO DE USO:	CU_Ingreso_Contrato
NOMBRE DEL CASO DE USO:	Crear_ Ingreso_Contrato
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar el registro de nuevo contrato
VISIÓN GENERAL:	El proceso inicia cuando se requiere realizar la contratación de un docente, a una o a varias instituciones.

TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando se requiere realizar una contratación.	
2.- Administrador solicita datos del nuevo contrato.	
3.- Administrador o usuario ingresa al módulo de ingreso de contratación.	4.-Muestra formulario de ingreso de contratos.
5.- Administrador o usuario llena los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado la contratación correctamente.
CURSOS ALTERNATIVOS	
Mensajes de error : 6.1 “Los campos marcados con asterisco son de ingreso obligatorio”. 6.2 “Debe ingresar al menos una institución” 6.3 “El nombre de la institución ingresado no existe en la base de datos” 6.4 “El nombre del docente ingresado no existe en la Base de datos”	
Mensajes de Advertencia: 7.1 “No se puede realizar la contratación”	
Mensaje de Información: 7.2 “Contratación exitosa”	

CASO DE USO EDICIÓNDE CONTRATOS

Tabla IV. XIII Caso de Uso Edición de Contratos

IDENTIFICAR CASO DE USO:	CU_Editar_Contrato
NOMBRE DEL CASO DE USO:	Crear_Editar_Contrato

ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar la edición de un contrato
VISIÓN GENERAL:	El proceso inicia cuando se requiere modificar un contrato ya se debido a que el docente va a ser asignado a mas instituciones, por correcciones del mismo, o añadir detalles del contratado.
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos

CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando se requiere editar una contratación.	
2.- Administrador solicita datos a modificar y/o agregar.	
3.- Administrador o usuario ingresa al módulo de edición de contratos	4.-Muestra formulario de edición de contratos, con la información actual del contrato
5.- Administrador o usuario llena los campos requeridos	6.- Valida datos del formulario
	7.- Presenta un mensaje que informa que se ha realizado la actualización correctamente.

CURSOS ALTERNATIVOS
Mensajes de error : 6.1 “Los campos marcados con asterisco son de ingreso obligatorio”. 6.2 “Debe ingresar al menos una institución” 6.3 “El nombre de la institución ingresado no existe en la base de datos”
Mensajes de Advertencia: 7.1 “No se puede realizar la contratación”
Mensaje de Información:

7.2 “Actualización exitosa”

CASO DE USO GENERACIÓN DE CONTRATO

Tabla IV. XIV Caso de Uso Edición de Contratos

IDENTIFICAR CASO DE USO:	CU_Generar_Contrato
NOMBRE DEL CASO DE USO:	Generar_Contrato
ACTORES:	Administrador, Usuario
PROPÓSITO:	Realizar la impresión del contrato
VISIÓN GENERAL:	El proceso inicia cuando ya todo el proceso de previo a la contratación se ha realizado, y tanto autoridades como el/la docente a contratar están listos para la firma del contrato.
TIPO:	Primario, real y expandido
REFERENCIAS:	Requerimientos
CURSO TÍPICO DE EVENTOS	
ACCIÓN DEL ACTOR	**RESPUESTAS DEL SISTEMA**
1.- Se inicia cuando se requiere imprimir el documento para la firma del contrato	
2.- Usuario solicita datos del contrato	
3.- Usuario ingresa a la sección de generación del contrato.	4.-Muestra formulario para ingresar el dato del docente.

5.- Usuario llena los campos requeridos	6.- Busca datos del formulario
	7.- Presenta el documento del contrato redactado
8.- Presiona el botón imprimir.	
CURSOS ALTERNATIVOS	
Mensajes de error : 6.1 “No ha ingresado el dato del docente”. 6.2 “El docente ingresado no tiene ningún contrato.”	
Mensajes de Advertencia: 7.1 “No se puede generar el contrato”	
Mensaje de Información: 7.2 “Contrato generado con éxito”	

Definir los Casos de Uso

Esta actividad nos permite representar los casos de uso a partir de la nomenclatura definida en UML, tanto para la representación de actores, casos de uso, interacciones y relaciones.

DIAGRAMA DE CASO DE USO AUTENTIFICACIÓN

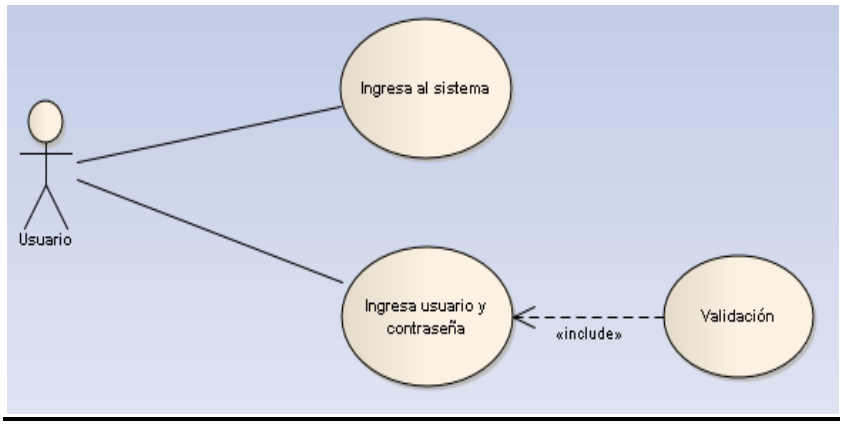


Figura IV. 6: Diagrama de Caso de Uso Autenticación

DIAGRAMA DE CASOS DE USO CREACION DE CUENTA

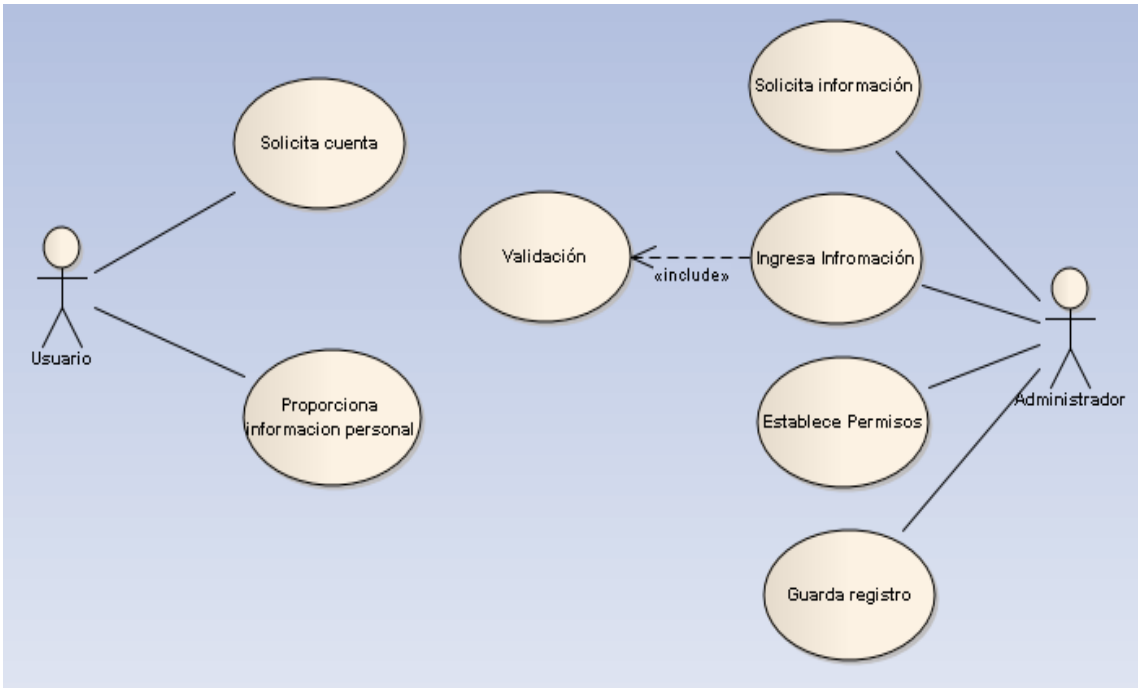


Figura IV. 7: Diagrama de Caso de Uso Creación de Cuenta

DIAGRAMA DE CASO DE USO CAMBIAR DATOS USUARIO

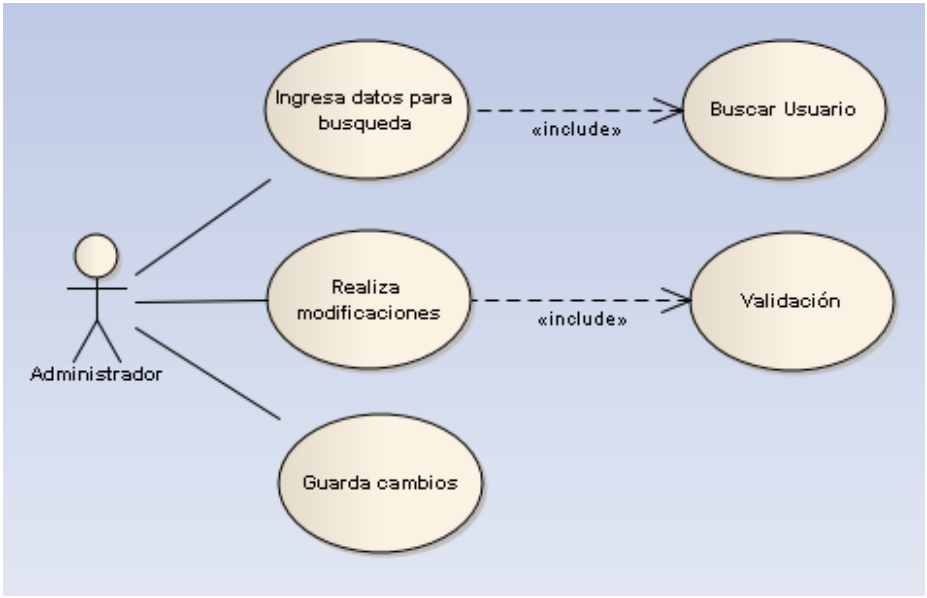


Figura IV. 8: Diagrama de Caso de Cambiar Datos de Usuario

DIAGRAMA DE CASOS DE USO DEINGRESOS

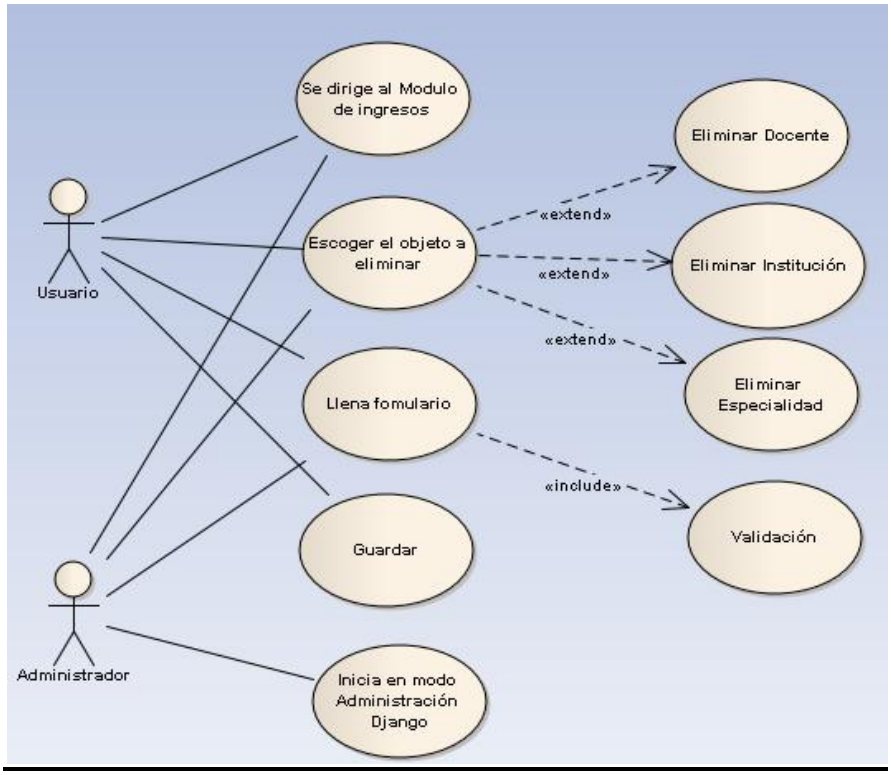


Figura IV. 9: Diagrama de Caso de Uso de Ingresos.

DIAGRAMA DE CASOS DE EDICION DEREGISTROS

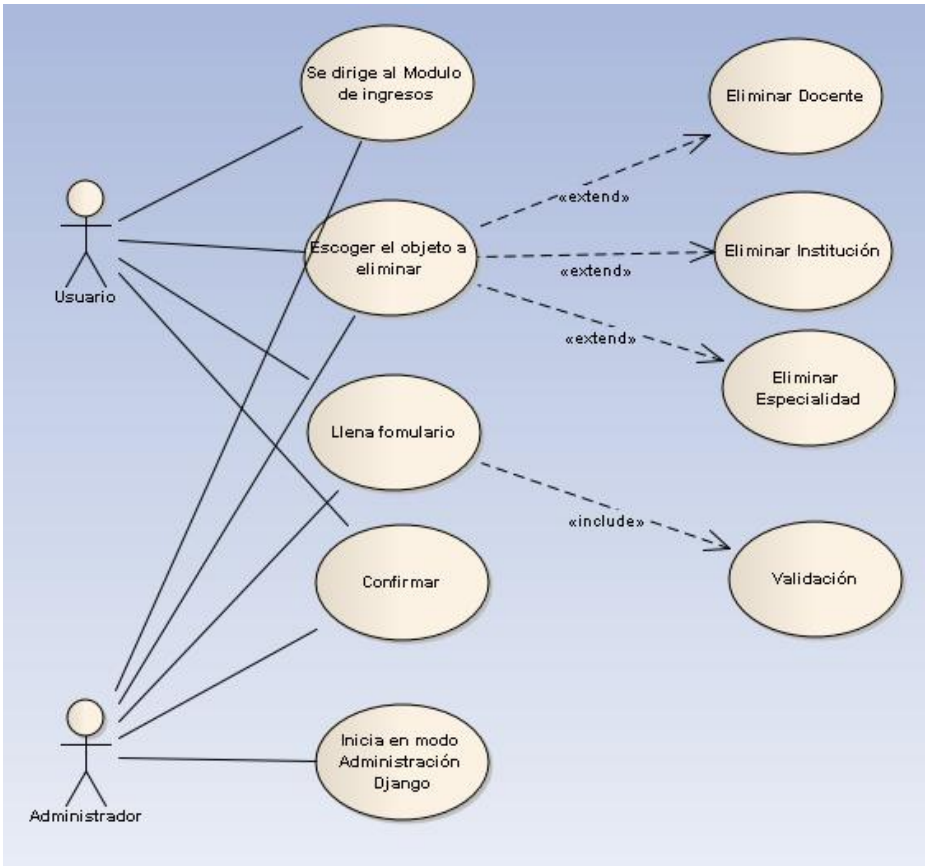


Figura IV. 10: Diagrama de Caso de Uso Edición de Registros.

DIAGRAMA DE CASOS DE ELIMINACIÓN DEREGISTROS

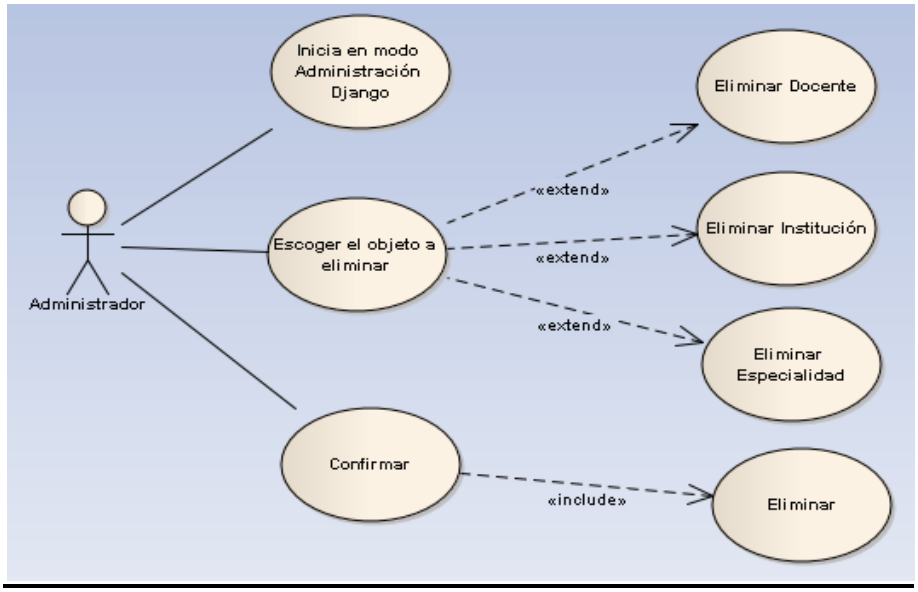


Figura IV. 11: Diagrama de Caso de Uso Eliminación de Registros.

DIAGRAMA DE CASOS DE USO DE CONTRATACIÓN

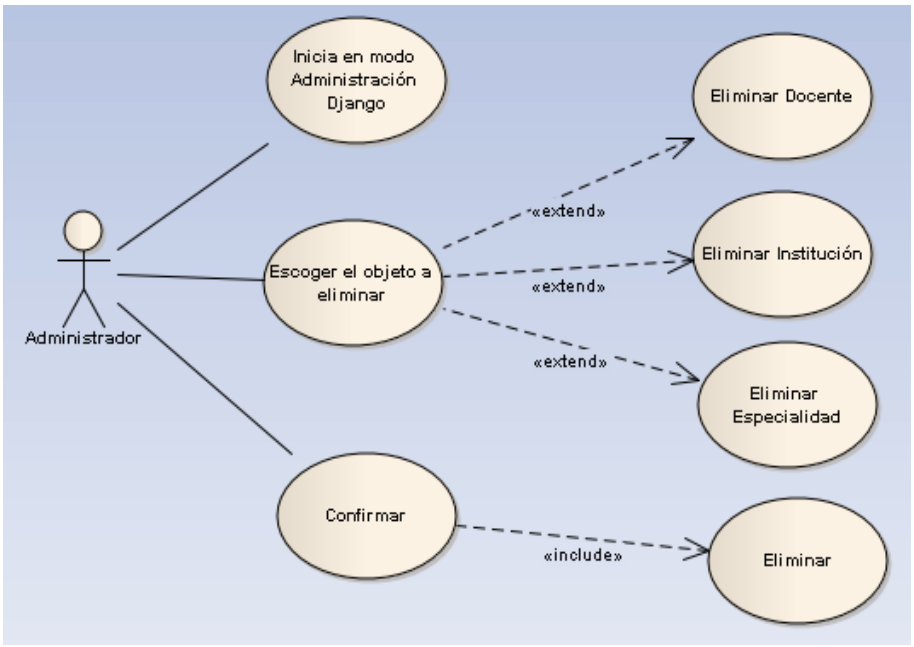


Figura IV. 12: Diagrama de Caso de Uso de Contratación.

DIAGRAMA DE CASO DE USO UTILIZACION SISTEMA

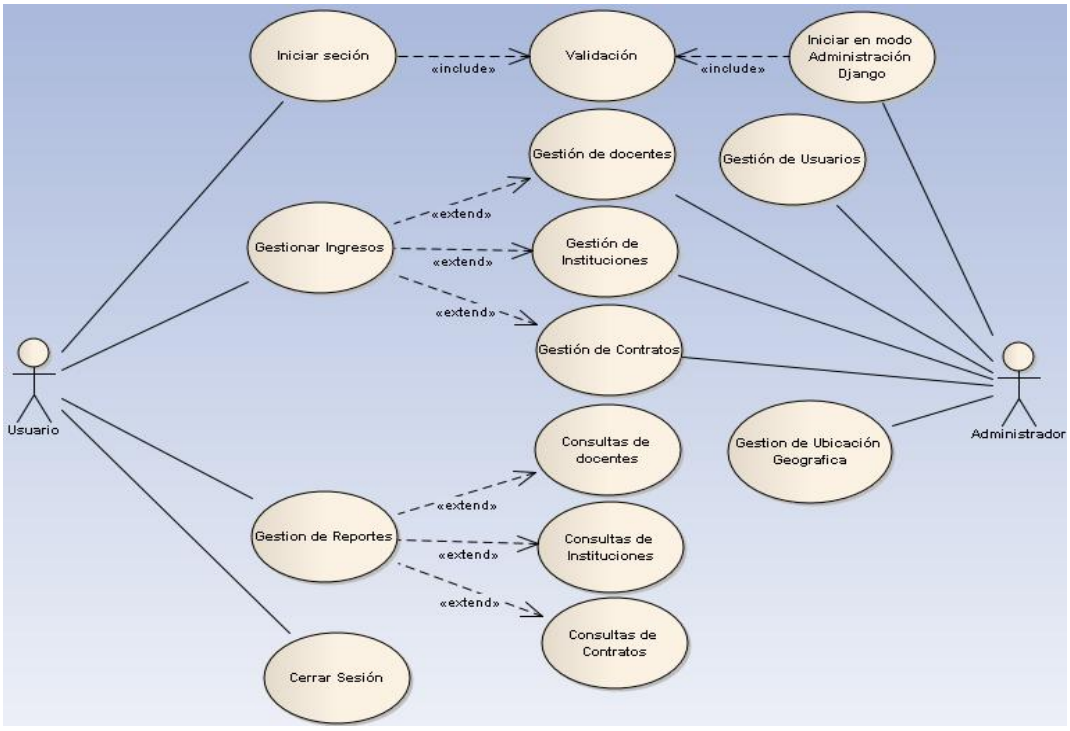


Figura IV. 13: Diagrama de Caso de Uso Utilización Sistema

4.2.2. Diagrama de Secuencias

DIAGRAMA DE SECUENCIA AUTENTICACIÓN

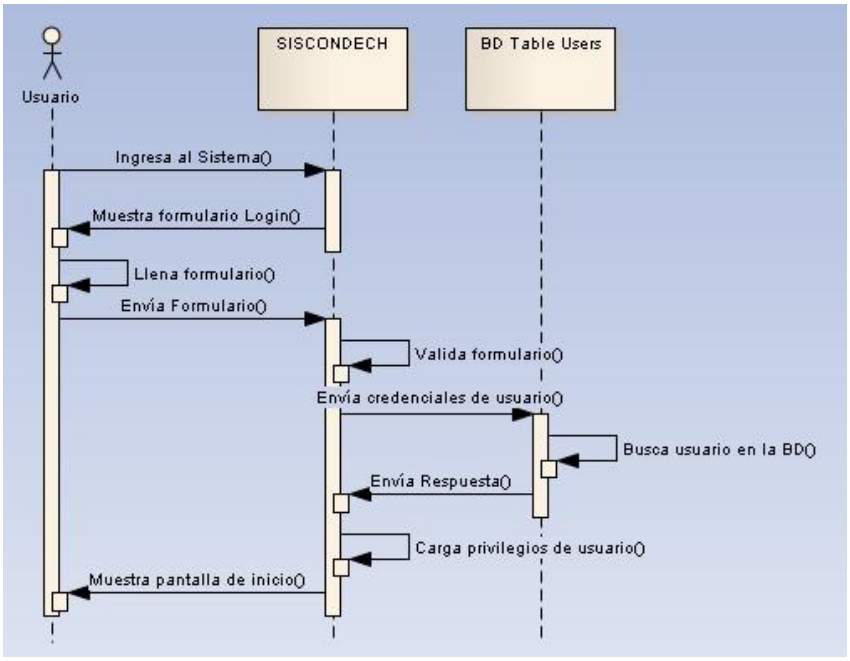


Figura IV. 14: Diagrama de Secuencias Autenticación

DIAGRAMA DE SECUENCIA CREACION DE CUENTA

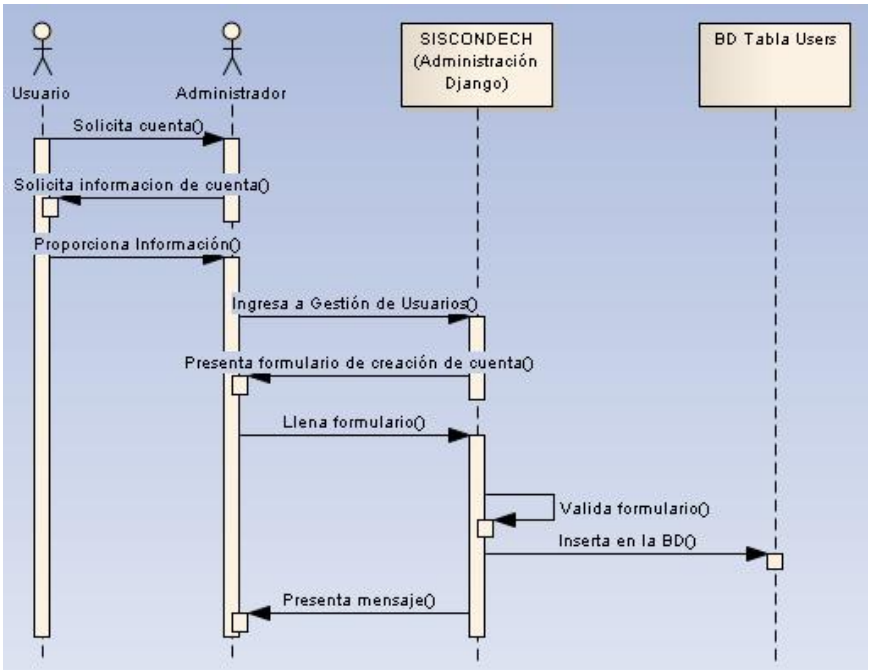


Figura IV. 15: Diagrama de Secuencias Creación de Cuenta

DIAGRAMA DE SECUENCIA CAMBIAR DATOS USUARIO

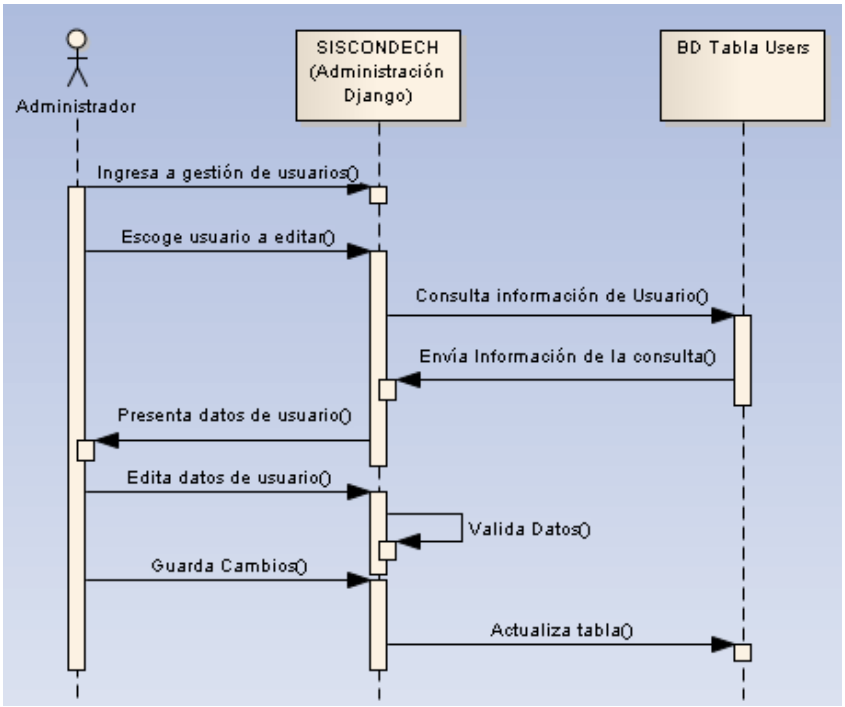


Figura IV. 16: Diagrama de Secuencias Cambiar Datos Usuario

DIAGRAMA DE SECUENCIAS DE INGRESOS

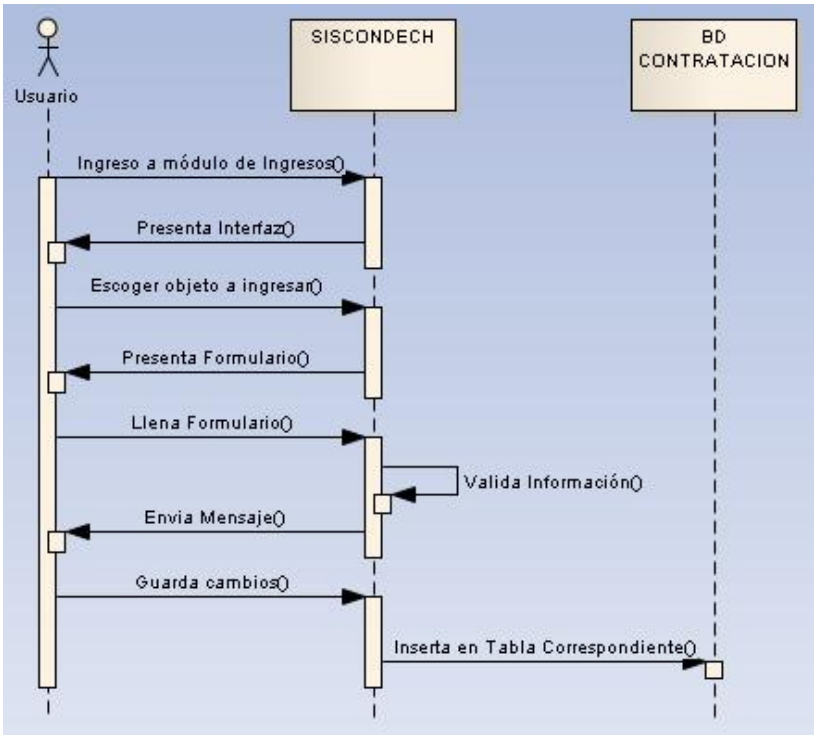


Figura IV. 17: Diagrama de Secuencia de Ingresos

DIAGRAMA DE SECUENCIA DE EDICIÓN DE REGISTROS

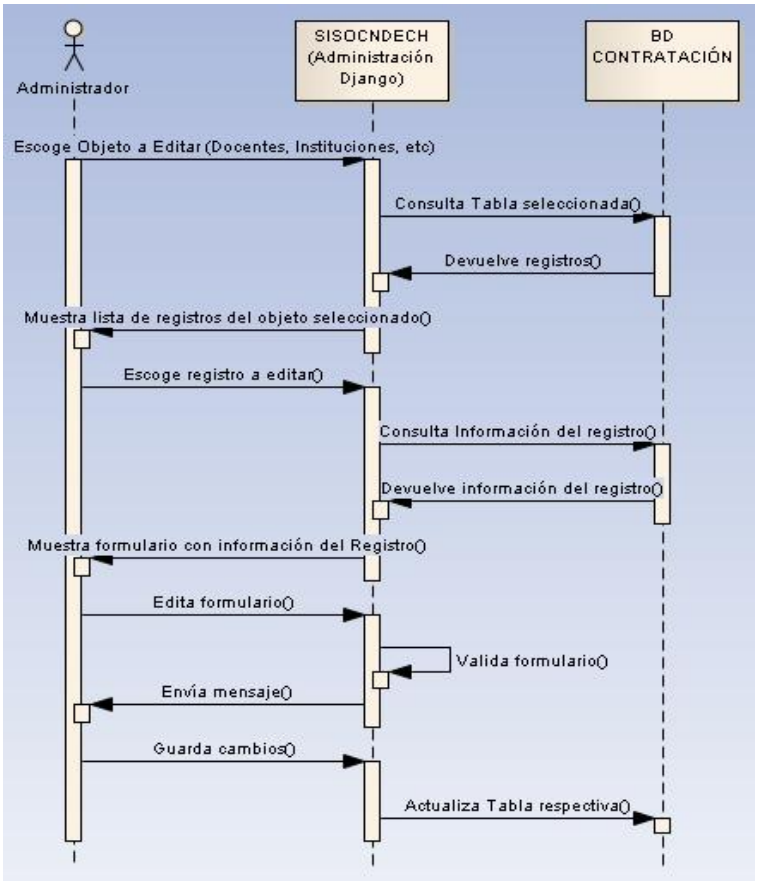


Figura IV. 18: Diagrama de Edición de Registros

DIAGRAMA DE SECUENCIA DE ELIMINACIÓN DE REGISTROS

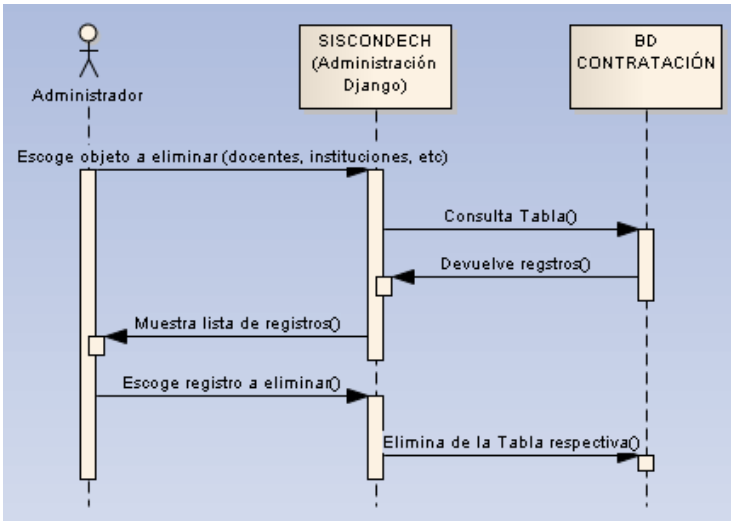


Figura IV. 19: Diagrama de Eliminación de Registros

DIAGRAMA DE SECUENCIA DE CONTRATACIÓN

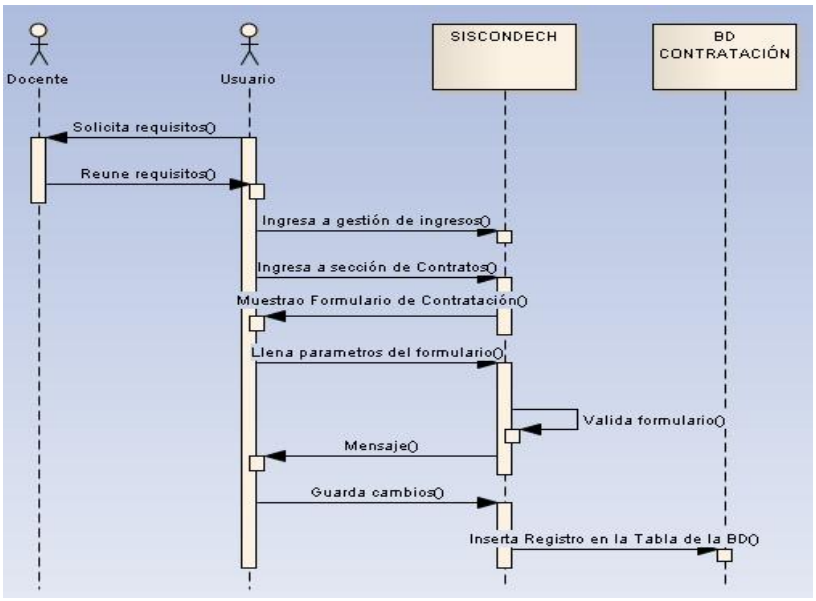


Figura IV. 20: Diagrama de Contratación

DIAGRAMA DE SECUENCIAS UTILIZACION SISTEMA

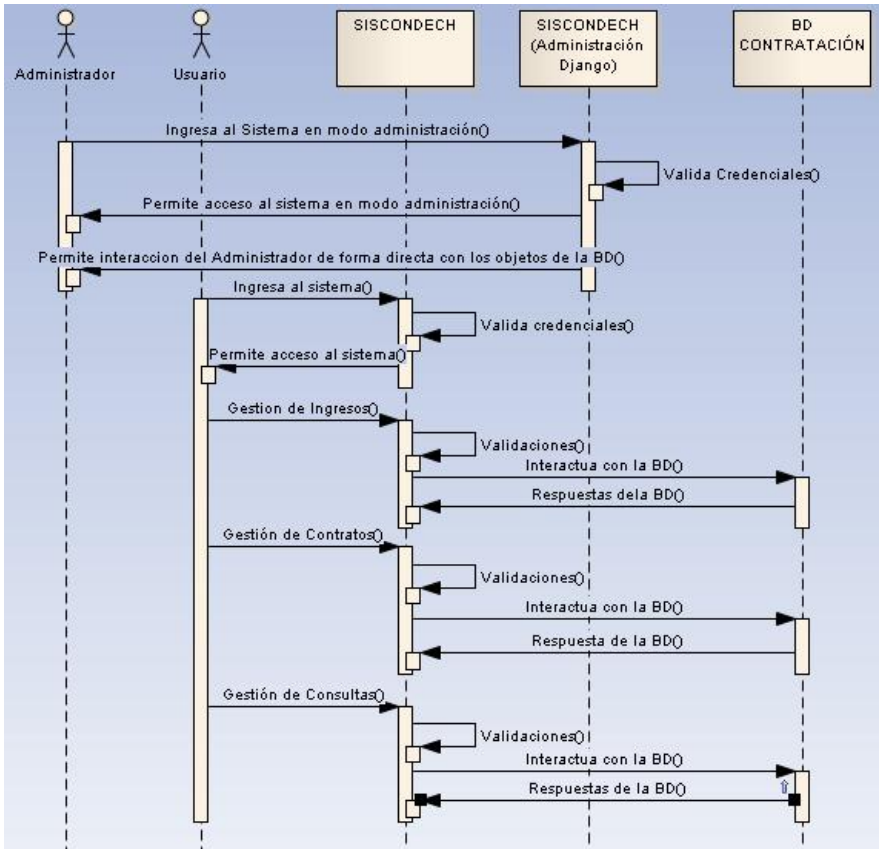


Figura IV. 21: Diagrama de Secuencias Utilización Sistema

4.3. Diseño

4.3.1. Diagrama de Clases

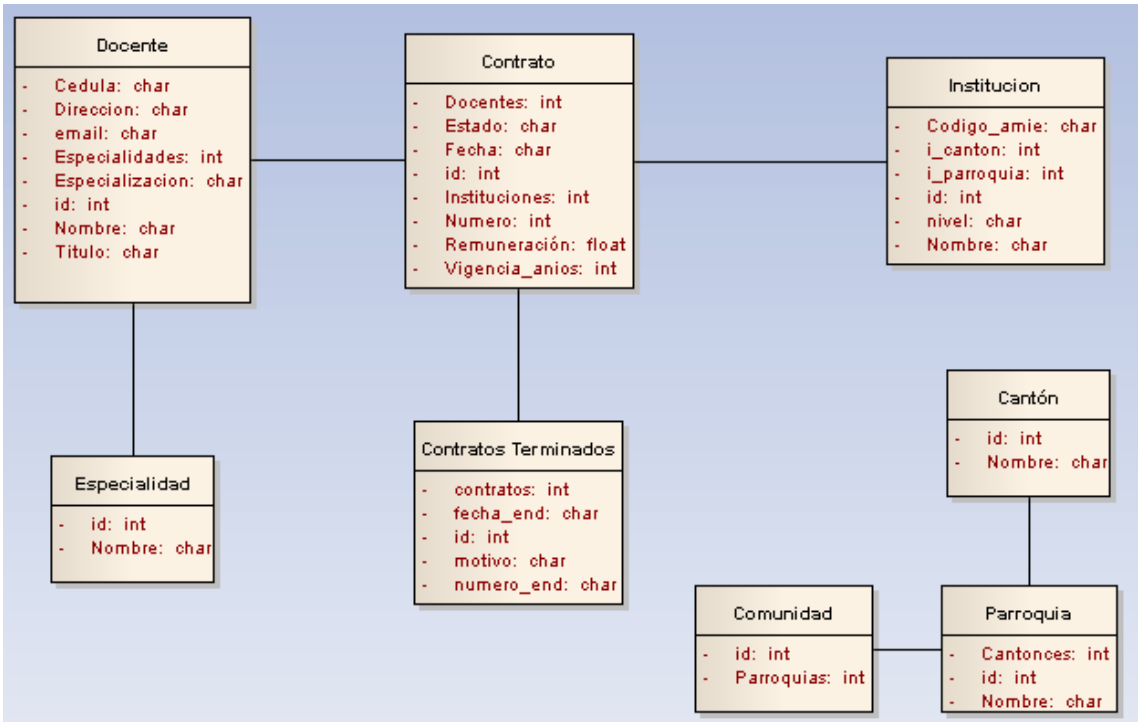


Figura IV. 22: Diagrama de Clases

4.3.2. Diagrama de Componentes

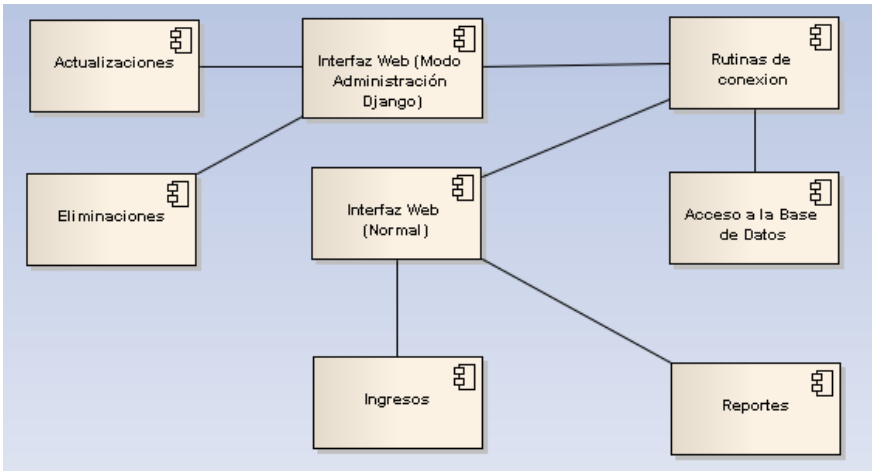


Figura IV. 23: Diagrama de Componentes

4.3.3. Diagrama de Nodos

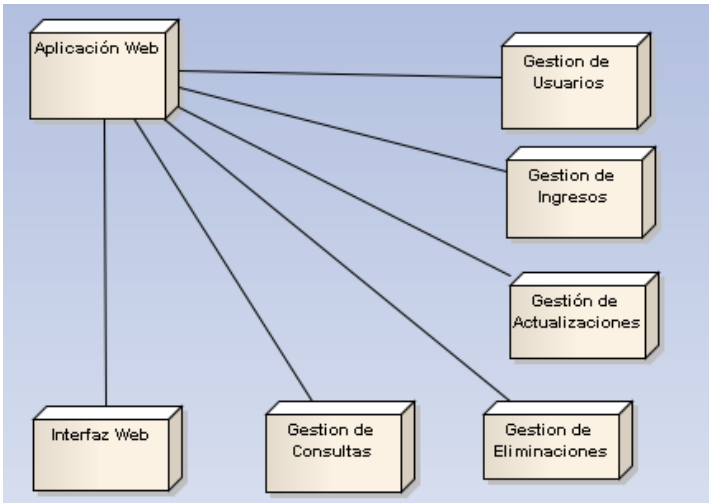


Figura IV. 24: Diagrama de Nodos

4.4. Implementación y Pruebas

4.4.1. Definición de estándares de Programación

Para realizar la codificación del sistema se han definido los siguientes estándares:

- El nombre de las tablas se encuentra con letras minúsculas.
- El nombre de los campos empieza con una letra minúscula, seguido de una línea hacia abajo en el caso de más de una palabra.
- Los métodos que interactúan con la base de datos en el caso de funciones de inserción, deben devolver como resultado el código del campo que ha sido insertado.

4.4.2. Pruebas Unitarias

Para asegurar el correcto funcionamiento del Sistema se han probado sus métodos de forma independiente, enviando datos de entrada desde el código, para luego obtener los a través de los diferentes métodos para realizar consultas. Se han probado especialmente todas las funciones para validaciones de datos.

4.4.3. Pruebas de Módulos y del Sistema

Las pruebas finales consistieron en verificar que la información ingresada se vea inmediatamente reflejada en las consultas del sistema, esto sirve para comprobar que la información se está registrando correctamente en la base de datos.

Se provocaron errores intencionales para verificar el correcto funcionamiento del sistema, así como de las funciones de validación de datos, como por ejemplo:
- Realizar consultas a tablas vacías
- Ingresar campos incorrectos

Tratar de ingresar información diferente al tipo de dato correcto, como tratar de ingresar texto en los campos que son numéricos, ingresar formatos de fechas diferentes, tratar de ingresar datos técnicos en servicios que no admiten esta información

CONCLUSIONES

1. Tanto Django así como Ruby on Rails presentan similares características las cuales nos brindan un optimo beneficio al momento de desarrollar ágilmente aplicaciones web diferenciadas entre sí mas no por su conceptualidad sino por su implementación y estructura. Lo cual nos permite obtener una clara visión en el enfoque de un desarrollo más productivo de aplicaciones web.

2. El realizar un estudio de la productividad de desarrollo de software nos permite determinar con claridad las prestaciones que nos brinda una aplicación como producto final, y debido a su importancia se ha considerado los factores más importantes entre los cuales destacamos: El patrón MVC, Reutilización, la seguridad de aplicación, madurez de producto e instalación.

3. Al realizar el estudio del Modelo en los distintos frameworks se puede determinar que de acuerdo a su implementación en los prototipos de prueba Django presenta 93,75% contra Ruby on Rails con el 100% presentando mejores características respecto al acceso con los datos.

4. El estudio de la vista es muy importante puesto que nos refleja como una aplicación web se presenta a los usuarios finales. Al ser implementada en los prototipos de prueba Django presenta un 90% frente al 83,33% que presenta Ruby on Rails demostrando Django una ventaja de 6,67% que permite determinar su ventaja al momento de realizar la capa de presentación en una aplicación.

5. El Controlador es un indicador que determina la funcionalidad de las aplicaciones web mediante los métodos y funciones que son desarrollados en determinado framework. Por su mejor adaptación entre el lenguaje y el acceso a datos Django muestra un 81,66% de eficiencia frente a Ruby on Rails con un 78,33% con una ventaja de 3,33%.

6. El indicador de reutilización demuestra la correcta utilización de objetos que pueden ser reutilizados lo cual permite aprovechar el trabajo anterior, economizar tiempo, y reducir la redundancia, posibilitando un óptimo desarrollo de aplicaciones web. Para lo cual Django con 93,75% de eficiencia mostrando una ventaja de 12,5% frente a 81,25% de Ruby on Rails.

7. El estudio de la seguridad de aplicación demanda una correcta utilización de factores que determinan un correcto cuidado de la información de una aplicación al momento de la autenticación de usuarios. Ambos frameworks presentan características similares las que representan un 95% de eficiencia en la seguridad de aplicación.

8. El análisis de la madurez del producto permite establecer indicadores que determinan la factibilidad de utilizar una determinada tecnología, la misma que es traducida en la rentabilidad que genera un producto software. Es así que Django presenta un 93,33% contra el 91,66% de Ruby on Rails de madurez de producto teniendo una ventaja de 1,67%.

9. El indicador de instalación proporciona una idea clara de la optimización en el tiempo y la facilidad en la implementación del framework lo cual se ve reflejado en un desarrollo más productivo de aplicaciones web. Django proporciona un 95% frente a 71,66% de Ruby on Rails superándolo con 23,34% del parámetro de instalación.

10. El análisis efectuado y mediante los resultados obtenidos, permite describir que la tecnología Python con Django presenta el 91,82% frente a 84,61% de Ruby on Rails. Por lo que se concluye que la aplicación de la Tecnología Python con Django mejora la productividad en el desarrollo ágil de aplicaciones web.

11. La investigación realizada acerca de la productividad en el desarrollo ágil de aplicaciones web permite determinar la tecnología apropiada para el desarrollo del Sistema de Contratación Docente de la DECH (SISCONDECH), el mismo que de acuerdo al estudio efectuado se lo ha desarrollado con tecnología Django bajo

Python con PostgreSQL. Este sistema permitirá a los analistas de contratación docente de la División de Recursos Humanos de la DECH, obtener control, supervisión, automatización y agilidad en el proceso de contratación de los docentes.

RECOMENDACIONES

1. Es recomendable que estos frameworks a pesar de ser multiplataforma, sean utilizados en alguna distribución Linux, tanto para entornos de desarrollo, pruebas y producción.

2. Al realizar un análisis comparativo de productividad en el desarrollo web utilizando Framework, se debe escoger muy bien los parámetros de comparación, se recomienda que entre los parámetros se considere: la arquitectura del framework (en este caso se han tomado de acuerdo a características del paradigma MVC), también se pueden establecer parámetros que permitan probar el acoplamiento existente con plantillas HTML que usen estilos, la reutilización de código, etc., en fin se deben comparar las características necesarias para llegar a un desarrollo web productivo.

3. Es recomendable destacar la importancia de definir escenarios de prueba bajo las mismas características ya que nos permitirán establecer de mejor manera y con hechos reales, las características y beneficios que nos proporciona cada Framework para el desarrollo Web.

4. Para explotar de mejor manera los beneficias de utilizar estas tecnologías lo recomendable es tener conocimientos de HTML, Hojas de estilo CSS, JavaScript, además del Sistema Operativo Linux, como se mencionó anteriormente cualquier distribución.

5. Para personas que se inicien en estas tecnologías es recomendable que antes de buscar un IDE, empiecen el desarrollo con editores de texto, ya que las ventajas de estos framework, son propios de ellos y no del IDE utilizado.

6. De acuerdo a lo investigado, al subir a un entorno de producción una aplicación desarrollada en Django, se recomienda usar Apache como servidor Web, pero de preferencia con el módulo “mod_wsgi” o “mod_fsgi”, ya que si utilizamos

"mod_python" este presentará dificultades, especialmente cuando a más de usar Python se usan otros lenguajes como PHP.

7. Se recomienda utilizar este tipo de frameworks para mejorar la productividad en el desarrollo de aplicaciones web ya que ayudan, al brindarnos facilidades como son las de evitarnos programar las conexiones a Base de Datos, Autenticación, métodos de Ingreso, Actualización o Eliminación, etc.

8. Para proyectos en los que el tiempo de desarrollo es un factor importante se recomienda la utilización de la metodología SCRUM, ya que al combinarla con la utilización de un Framework como Ruby on Rails o Django, finalmente obtendremos una aplicación en un tiempo considerablemente menor, además de estar interactuando constantemente durante el desarrollo con los futuros usuarios del nuevo sistema o aplicación.

9. Se recomienda también la utilización de herramientas software libre, ya que no estamos limitados a un software en específico, sino que estamos en un ambiente en el que todos podemos colaborar con nuestros aportes, podemos compartir y mejorar lo que nos han dado y lo más importante sabremos el porqué de las cosas, ya que al tener la cultura de software libre siempre habrá personas dispuestas en colaborar con sus conocimientos hacia la comunidad.

RESUMEN

Análisis de Python con Django frente a Ruby on Rails para desarrollo ágil de aplicaciones web, caso práctico realizado en la Dirección Provincial de Educación de Chimborazo. Se realizó el análisis de las tecnologías de framework de software libre más populares en el mercado, con el propósito de determinar que framework permite mejorar la productividad de desarrollo ágil de aplicaciones web.

El análisis comparativo se baso en el método Científico General, con lo cual se efectuó la investigación. Ambas tecnologías fueron adaptadas a los mismos ambientes de desarrollo y pruebas, con técnicas de observación directa, de laboratorio y experimento, para alcanzar los objetivos se utilizo las siguientes herramientas: Django 1.3.1 y Ruby on Rails 3.1, Servidor Apache en modo FastCGI, PostgreSQL, Aptana Studio 3 como IDE de desarrollo y la construcción de prototipos de pruebas.

Los resultados cuantitativos que se obtuvieron mediante la comparación de parámetros como: Patrón MVC, Reutilización, Seguridad de Aplicación, Madurez de Producto e Instalación, permitieron evaluar la productividad de desarrollo, cuyo resultado fue: Django 91,82% (Muy Bueno) y Ruby on Rails con 84,61% (Muy Bueno).

En conclusión la tecnología que ofrece la mejor productividad en el desarrollo ágil de aplicaciones web es Django con un puntaje de 92,81%/100, que supera a su directo competidor Ruby on Rails.

Es recomendable que para proyectos donde el tiempo de desarrollo es un factor importante se debe utilizar de la metodología SCRUM, ya que al combinarla con la utilización de un Framework obtendremos una aplicación en un tiempo considerablemente menor

BIBLIOGRAFÍA

BIBLIOGRAFÍA DE LIBROS

[1] **FERNANDEZ O.,** The Rails Way., Edición 2.0., Editorial Pragmatic., Estados Unidos., 2010., 910p

[2] **GONZALEZ R.,** Python para Todos., Segunda Edición., Madrid España., 2010., 108p.

[3] **HOLOVATY A. y otros.,** El Libro de Django., Segunda Edición., Apres., 2007., 381p.

BIBLIOGRAFÍA DE ARTICULOS CIENTIFICOS

[4] **CACERES P. y otros.,** Procesos Agiles para el Desarrollo de Aplicaciones Web., Madrid España., Universidad Rey Juan Carlos., Departamento de Ciencias Experimentales e Ingeniería.

[5] **MÉNDEZ E. y otros.,** ¿Cómo se relacionan la Calidad Sistémica y la Productividad en el Proceso de Desarrollo de Software?., Caracas Venezuela., Universidad Simón Bolívar.

[6] **SOLANO R.,** Ruby on Rails, una forma rápida de hacer aplicaciones web., San José Costa Rica,. Universidad de Costa Rica., Escuela de Ciencias de la Computación e Informática.

BIBLIOGRAFÍA DE TESIS

[7] **GUEVARA J. y otros.,** Análisis del patrón modelo vista controlador implementado en lenguajes de software libre para el desarrollo de

aplicaciones web., Tesis (Ingeniero en Sistemas Informáticos)., Riobamba Ecuador., ESPOCH,. Facultad de Informática y Electrónica., 2010., 159h.

BIBLIOGRAFÍA DE INTERNET

[8] **BERKELEY SOFTWARE DISTRIBUTION**., Licencias de software Abril 30 de 2012., [en línea] from: http://www.bsd.org/

[9] **CONTI J.,** Filosofía del diseño de Python. Enero 21 de 2012 [en línea] from: http://www.juanjoconti.com.ar/2009/01/21/la-historia-de-python-filosofia-de-diseno/

[10] **DESARROLLO WEB CON PYTHON Y PYLONS**. Marzo 22 de 2012., [en línea] from: http://toporojo.es/blog/2012/02/11/desarrollo-web-con-python-pylons/

[11] **DJANGO ESPAÑOL.,** Django Framework. Abril 18 2012,. [en línea] from: http://django.es/

[12] **ESTADO I.,** Mini Framework para Ruby. Abril 13 de 2012., [en línea] from: http://www.estadobeta.net/2006/11/13/merb-mini-framework-para-ruby/index.html

[13] **GUTIERREZ J.,** ¿Qué es un Framework Web? Marzo 19 de 2012., [en línea] from: http://www.cssblog.es/guias/Framework.pdf

[14] **GROK COMUNITY**., Grok Framework.
Mayo 17 de 2012., [en línea] from:
http://grok.zope.org/

[15] **INICIATIVA ESPAÑOLA DE SOFTWARE Y SERVICIOS.,**
Productividad en el Desarrollo de Software.
Marzo 23 de 2012., [en línea] from:
http://www.ines.org.es/node/1485

[16] **MELE A.,** Taller de Django Betabeers., [videograbación], Barcelona España. Octubre 20 de 2011 from:
http://django.es/blog/

[17] **NOWAY.**, Tutorial de Instalación de Django.,
Abril 03 de 2012., [Tutorial en línea] from:
http://www.noway.es/django-instalacion-tutorial

[18] **PALCIO J.,** Gestión de Proyectos Ágil.
Abril 23 de 2012 [en línea], from:
http://www.navegapolis.net/files/s/NST-003_01.pdf

[19] **PRESTASHOP.,** El Patrón MVC.
Marzo 29 de 2012., [en línea] from:
http://prestashop5estrellas.wordpress.com/2010/03/29/el-patron-mvc-modelo-vista-controlador/

[20] **RIGADA A.,** Model View Controller.
Mayo 20 de 2012 [en línea] from:
http://blogdeaitor.wordpress.com/2008/10/20/model-view-controller/

[21] **RUBY LANG.,** Acerca de Ruby.
Mayo 5 de 2012., [en línea] from:

http://www.ruby-lang.org/es/about/

[22] **SAAVEDRA J.**, El Mundo Informático,. Lenguajes de Programación., Marzo 13 de 2012., [en línea] from: http://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/

[23] **SLAGEL M.,** Ruby User Guide. Marzo 04 de 2012., [en línea] from: http://www.rubyist.net/~slagell/ruby/index.html

[24] **STEWART B.,** An Interview with the Creator of Ruby. Marzo 29 de 2012., [en línea] from: http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html

[25] **SUGIHARA H.,** The desicive moment of the language name Ruby. Marzo 25 de 2012., [en línea] from: http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/88819

[26] **SWAROOP C H.,** A byte of Python Abril 15 de 2012., [Tutorial en línea] from: http://dev.laptop.org/~edsiper/byteofpython_spanish/ch01s02.html

[27] **WEB2PY ESPAÑOL.,** Framework Web2py. Abril 18 2012 [en línea] from: http://www.web2py.com.ar/examples/default/index

[28] **WIKIPEDIA.,** Django. Mayo 16 de 2012., [en línea], from: http://es.wikipedia.org/wiki/Django

[29] **WIKIPEDIA., Python**.

Abril 13 de 2012., [en línea] from:
http://es.wikipedia.org/wiki/Python

[30] **WIKIPEDIA**., Ramaze Framework.
Abril 19 de 2012., [en línea] from:
http://es.wikipedia.org/wiki/Ramaze

[31] **WIKIPEDIA.,** Ruby.
Mayo 20 de 2012 [en línea], from:
http://es.wikipedia.org/wiki/Ruby

[32] **WIKIPEDIA.,** Ruby on Rails.
Abril 25 de 2012., [en línea] from:
http://es.wikipedia.org/wiki/Ruby_on_Rails

[33] **WIKIPEDIA.**, Sinatra Framework.
Febrero 29 de 2012., [en línea] from:
http://es.wikipedia.org/wiki/Sinatra_(software)

[34] **WIKIPEDIA.,** TuboGears.
Junio 12 de 2012., [en línea] from:
http://es.wikipedia.org/wiki/TurboGears

ANEXOS

ANEXO 1

Todas las líneas de código programadas en la construcción de los módulos de prueba en los dos frameworks, Ruby on Rails así como Django.

Ruby on Rails Framework
ANEXO

Líneas de código del prototipo construido utilizando el Framework de desarrollo Web Ruby on Rails, incluye los modelos, vistas, controladores, archivos de configuración, etc.

MODELOS

institucion.rb

```
class Institucion < ActiveRecord::Base
has_many :contratos
validates :nombre, :ubicacion, presence: true
validates :nombre, uniqueness: true
end
```

docente.rb

```
class Docente < ActiveRecord::Base
has_many :contratos, dependent: :destroy
validates :nombres, :apellidos, :ci, :direccion, :telefono, :email,
presence: true
validates :ci, uniqueness: true
end
```

contrato.rb

```
class Contrato < ActiveRecord::Base
belongs_to :docente
belongs_to :institucion
end
```

user.rb

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :token_authenticatable, :encryptable, :confirmable, :lockable,
:timeoutable and :omniauthable
devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  # Setup accessible (or protected) attributes for your model
  attr_accessible :email, :username, :password,
:password_confirmation, :remember_me
end
```

VISTAS

CONTRATOS

_form.html.erb

```
<%= form_for(@contrato) do |f| %>

<% if @contrato.errors.any? %>

<div id="error_explanation">

<h2><%= pluralize(@contrato.errors.count, "error") %> prohibited this
contrato from being saved:</h2>



<ul>

<% @contrato.errors.full_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

<% end %>



<div class="field">

<%= f.label :docente_id %><br />

<%= f.number_field :docente_id %>

</div>

<div class="field">
```

```
<%= f.label :institucion_id %><br />
<%= f.number_field :institucion_id %>
</div>
<div class="field">
<%= f.label :observaciones %><br />
<%= f.text_field :observaciones %>
</div>
<div class="actions">
<%= f.submit %>
</div>
<% end %>
```

edit.html.erb

```
<h1>Editing contrato</h1>
<%= render 'form' %>
<%= link_to 'Show', @contrato %> |
<%= link_to 'Back', contratos_path %>
```

index.html.erb

```
<h1>Listing contratos</h1>
<table>
<tr>
<th>Docente</th>
<th>Institucion</th>
<th>Observaciones</th>
<th></th>
<th></th>
<th></th>
</tr>
<% @contratos.each do |contrato| %>
<tr>
```

```
<td><%= contrato.docente_id %></td>

<td><%= contrato.institucion_id %></td>

<td><%= contrato.observaciones %></td>

<td><%= link_to 'Show', contrato %></td>

<td><%= link_to 'Edit', edit_contrato_path(contrato) %></td>

<td><%= link_to 'Destroy', contrato, confirm: 'Are you sure?', method:
:delete %></td>

</tr>

<% end %>

</table>

<br />

<%= link_to 'New Contrato', new_contrato_path %>
```

new.html.erb

```
<h1>New contrato</h1>

<%= render 'form' %>

<%= link_to 'Back', contratos_path %>
```

Show.html.erb

```
<p id="notice"><%= notice %></p>

<p>

<b>Docente:</b>

<%= @contrato.docente_id %>

</p>

<p>

<b>Institucion:</b>

<%= @contrato.institucion_id %>

</p>

<p>

<b>Observaciones:</b>

<%= @contrato.observaciones %>
```

```
</p>

<%= link_to 'Edit', edit_contrato_path(@contrato) %> |

<%= link_to 'Back', contratos_path %>
```

DOCENTES

_form-html.erb

```
<%= form_for(@docente) do |f| %>

<% if @docente.errors.any? %>

<div id="error_explanation">

<h2><%= pluralize(@docente.errors.count, "error") %> prohibited this
docente from being saved:</h2>

<ul>

<% @docente.errors.full_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

<% end %>

<div class="field">

<%= f.label :nombres %><br />

<%= f.text_field :nombres %>

</div>

<div class="field">

<%= f.label :apellidos %><br />

<%= f.text_field :apellidos %>

</div>

<div class="field">

<%= f.label :ci %><br />

<%= f.text_field :ci %>

</div>
```

```
<div class="field">
<%= f.label :direccion %><br />
<%= f.text_field :direccion %>
</div>
<div class="field">
<%= f.label :telefono %><br />
<%= f.text_field :telefono %>
</div>
<div class="field">
<%= f.label :email %><br />
<%= f.text_field :email %>
</div>
<div class="actions">
<%= f.submit %>
</div>
<% end %>
```

edit.html.erb

```
<h1>Editing docente</h1>
<%= render 'form' %>
<%= link_to 'Show', @docente %> |
<%= link_to 'Back', docentes_path %>
```

index.html.erb

```
<h1>Listing docentes</h1>
<table>
<tr>
<th>Nombres</th>
<th>Apellidos</th>
<th>Ci</th>
<th>Direccion</th>
```

```
<th>Telefono</th>

<th>Email</th>

<th></th>

<th></th>

<th></th>

</tr>

<% @docentes.each do |docente| %>

<tr>

<td><%= docente.nombres %></td>

<td><%= docente.apellidos %></td>

<td><%= docente.ci %></td>

<td><%= docente.direccion %></td>

<td><%= docente.telefono %></td>

<td><%= docente.email %></td>

<td><%= link_to 'Show', docente %></td>

<td><%= link_to 'Edit', edit_docente_path(docente) %></td>

<td><%= link_to 'Destroy', docente, confirm: 'Are you sure?', method:
:delete %></td>

</tr>

<% end %>

</table>

<br />

<%= link_to 'New Docente', new_docente_path %>
```

new.html.erb

```
<h2>Nuevo docente</h2>

<%= render 'form' %>

<%= link_to 'Back', docentes_path %>
```

show.html.erb

```
<p id="notice"><%= notice %></p>
```

```
<p>
<b>Nombres:</b>
<%= @docente.nombres %>
</p>
<p>
<b>Apellidos:</b>
<%= @docente.apellidos %>
</p>
<p>
<b>Ci:</b>
<%= @docente.ci %>
</p>
<p>
<b>Direccion:</b>
<%= @docente.direccion %>
</p>
<p>
<b>Telefono:</b>
<%= @docente.telefono %>
</p>
<p>
<b>Email:</b>
<%= @docente.email %>
</p>
<%= link_to 'Edit', edit_docente_path(@docente) %> |
<%= link_to 'Back', docentes_path %>
```

INSTITUCIONES

_form.html.erb

```
<%= form_for(@institucion) do |f| %>
```

```
<% if @institucion.errors.any? %>

<div id="error_explanation">

<h2><%= pluralize(@institucion.errors.count, "error") %> prohibited
this institucion from being saved:</h2>

<ul>

<% @institucion.errors.full_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

<% end %>

<div class="field">

<%= f.label :nombre %><br />

<%= f.text_field :nombre %>

</div>

<div class="field">

<%= f.label :ubicacion %><br />

<%= f.text_field :ubicacion %>

</div>

<div class="actions">

<%= f.submit %>

</div>

<% end %>
```

edit.html.erb

```
<h1>Editing institucion</h1>

<%= render 'form' %>

<%= link_to 'Show', @institucion %> |

<%= link_to 'Back', institucions_path %>
```

index.html.erb

```
<h1>Listing institucions</h1>

<table>

<tr>

<th>Nombre</th>

<th>Ubicacion</th>

<th></th>

<th></th>

<th></th>

</tr>

<% @institucions.each do |institucion| %>

<tr>

<td><%= institucion.nombre %></td>

<td><%= institucion.ubicacion %></td>

<td><%= link_to 'Show', institucion %></td>

<td><%= link_to 'Edit', edit_institucion_path(institucion) %></td>

<td><%= link_to 'Destroy', institucion, confirm: 'Are you sure?',
method: :delete %></td>

</tr>

<% end %>

</table>

<br />

<%= link_to 'New Institucion', new_institucion_path %>
```

new.html.erb

```
<h2>Nueva institución</h2>

<%= render 'form' %>

<%= link_to 'Back', institucions_path %>
```

show.html.erb

```
<p id="notice"><%= notice %></p>

<p>
```

```
<b>Nombre:</b>
<%= @institucion.nombre %>
</p>
<p>
<b>Ubicacion:</b>
<%= @institucion.ubicacion %>
</p>
<%= link_to 'Edit', edit_institucion_path(@institucion) %> |
<%= link_to 'Back', institucions_path %>
```

PAGINAS ESTÁTICAS

acercade.html.erb

```
<% provide(:title, 'Acerca de')%>
<h1>Prototipo</h1>
<p>
      Esta es Acerca de para el Sistema de contratación docente
</p>
```

ayuda.html.erb

```
<% provide(:title, 'Ayuda')%>
<h1>Prototipo</h1>
<p>
      Esta es la ayuda del Sistema de contratación docente
</p>
```

consultas.html.erb

```
<% provide(:title, 'Ingresos')%>
<h1>Prototipo</h1>
<p>
```

```
      <li><%= link_to "Listado de docentes de docentes", docentes_path
%></li>

      <li><%= link_to "listado de instituciones", instituciones_path
%></li>

      <li>Listado de Contratos</li>

</p>
```

ingresos.html.erb

```
<% provide(:title, 'Ingresos')%>

<h1>Prototipo</h1>

<p>

      <li><%= link_to "Ingreso de docentes", ingreso_docentes_path
%></li>

      <li><%= link_to "Ingreso de instituciones",
ingreso_instituciones_path %></li>

      <li>Ingreso de Contratos</li>

</p>
```

inicio.html.erb

```
<% provide(:title, 'Inicio')%>

<h1>Prototipo</h1>

<p>

      Esta es la página de inicio para el Sistema de contratación
docente

</p>
```

USERS

Confirmations

new.html.erb

```
<h2>Resend confirmation instructions</h2>

<%= form_for(resource, :as => resource_name, :url =>
confirmation_path(resource_name), :html => { :method => :post }) do
|f| %>

<%= devise_error_messages! %>

<div><%= f.label :email %><br />

<%= f.email_field :email %></div>

<div><%= f.submit "Resend confirmation instructions" %></div>

<% end %>

<%= render "links" %>
```

MAILER

confirmation_instructions.html.erb

```
<p>Welcome <%= @resource.email %>!</p>

<p>You can confirm your account email through the link below:</p>

<p><%= link_to 'Confirm my account', confirmation_url(@resource,
:confirmation_token => @resource.confirmation_token) %></p>
```

reset_password_instructions.html.erb

```
<p>Hello <%= @resource.email %>!</p>

<p>Someone has requested a link to change your password, and you can
do this through the link below.</p>

<p><%= link_to 'Change my password', edit_password_url(@resource,
:reset_password_token => @resource.reset_password_token) %></p>

<p>If you didn't request this, please ignore this email.</p>

<p>Your password won't change until you access the link above and
create a new one.</p>
```

unlock_instructions.html.erb

```
<p>Hello <%= @resource.email %>!</p>

<p>Your account has been locked due to an excessive amount of
unsuccessful sign in attempts.</p>

<p>Click the link below to unlock your account:</p>

<p><%= link_to 'Unlock my account', unlock_url(@resource,
:unlock_token => @resource.unlock_token) %></p>
```

PASSWORDS

edit.html.erb

```
<h2>Change your password</h2>

<%= form_for(resource, :as => resource_name, :url =>
password_path(resource_name), :html => { :method => :put }) do |f| %>

<%= devise_error_messages! %>

<%= f.hidden_field :reset_password_token %>

<div><%= f.label :password, "New password" %><br />

<%= f.password_field :password %></div>

<div><%= f.label :password_confirmation, "Confirm new password" %><br
/>

<%= f.password_field :password_confirmation %></div>

<div><%= f.submit "Change my password" %></div>

<% end %>

<%= render "links" %>
```

new.html.erb

```
<h2>Forgot your password?</h2>

<%= form_for(resource, :as => resource_name, :url =>
password_path(resource_name), :html => { :method => :post }) do |f| %>

<%= devise_error_messages! %>

<div><%= f.label :email %><br />

<%= f.email_field :email %></div>

<div><%= f.submit "Send me reset password instructions" %></div>

<% end %>

<%= render "links" %>
```

REGISTRATIONS

edit.html.erb

```
<h2>Edit <%= resource_name.to_s.humanize %></h2>

<%= form_for(resource, :as => resource_name, :url =>
registration_path(resource_name), :html => { :method => :put }) do |f|
%>

<%= devise_error_messages! %>

<div><%= f.label :email %><br />

<%= f.email_field :email %></div>

<div><%= f.label :password %><i>(leave blank if you don't want to
change it)</i><br />

<%= f.password_field :password, :autocomplete => "off" %></div>

<div><%= f.label :password_confirmation %><br />

<%= f.password_field :password_confirmation %></div>

<div><%= f.label :current_password %><i>(we need your current password
to confirm your changes)</i><br />

<%= f.password_field :current_password %></div>

<div><%= f.submit "Update" %></div>

<% end %>

<h3>Cancel my account</h3>

<p>Unhappy? <%= link_to "Cancel my account",
registration_path(resource_name), :confirm => "Are you sure?", :method
=> :delete %>.</p>

<%= link_to "Back", :back %>
```

new.html.erb

```
<h2>Sign up</h2>

<%= form_for(resource, :as => resource_name, :url =>
registration_path(resource_name)) do |f| %>

<%= devise_error_messages! %>

<div><%= f.label :email %><br />

<%= f.email_field :email %></div>

<div><%= f.label :password %><br />
```

```
<%= f.password_field :password %></div>

<div><%= f.label :password_confirmation %><br />

<%= f.password_field :password_confirmation %></div>

<div><%= f.submit "Sign up" %></div>

<% end %>

<%= render "links" %>
```

SESSIONS

new.html.erb

```
<h2>Sign in</h2>

<%= form_for(resource, :as => resource_name, :url =>
session_path(resource_name)) do |f| %>

<div><%= f.label :email %><br />

<%= f.email_field :email %></div>

<div><%= f.label :password %><br />

<%= f.password_field :password %></div>

<% if devise_mapping.rememberable? -%>

<div><%= f.check_box :remember_me %><%= f.label :remember_me %></div>

<% end -%>

<div><%= f.submit "Sign in" %></div>

<% end %>

<%= render "links" %>
```

UNLOCKS

links.html.erb

```
<%- if controller_name != 'sessions' %>

<%= link_to "Sign in", new_session_path(resource_name) %><br />

<% end -%>

<%- if devise_mapping.registerable? && controller_name !=
'registrations' %>

<%= link_to "Sign up", new_registration_path(resource_name) %><br />

<% end -%>
```

```
<%- if devise_mapping.recoverable? && controller_name != 'passwords'
%>

<%= link_to "Forgot your password?", new_password_path(resource_name)
%><br />

<% end -%>

<%- if devise_mapping.confirmable? && controller_name !=
'confirmations' %>

<%= link_to "Didn't receive confirmation instructions?",
new_confirmation_path(resource_name) %><br />

<% end -%>

<%- if devise_mapping.lockable? &&
resource_class.unlock_strategy_enabled?(:email) && controller_name !=
'unlocks' %>

<%= link_to "Didn't receive unlock instructions?",
new_unlock_path(resource_name) %><br />

<% end -%>

<%- if devise_mapping.omniauthable? %>

<%- resource_class.omniauth_providers.each do |provider| %>

<%= link_to "Sign in with #{provider.to_s.titleize}",
omniauth_authorize_path(resource_name, provider) %><br />

<% end -%>

<% end -%>
```

CONTROLADORES

application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery
end
```

contratos_controller.rb

```
class ContratosController < ApplicationController
  before_filter :authenticate_user!
  # GET /contratos
  # GET /contratos.json
def index
    @contratos = Contrato.all
    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @contratos }
end
end
  # GET /contratos/1
  # GET /contratos/1.json
def show
    @contrato = Contrato.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @contrato }
end
end
  # GET /contratos/new
  # GET /contratos/new.json
def new
    @contrato = Contrato.new
```

```
    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @contrato }
end
end
  # GET /contratos/1/edit
def edit
    @contrato = Contrato.find(params[:id])
end
  # POST /contratos
  # POST /contratos.json
def create
    @contrato = Contrato.new(params[:contrato])
    respond_to do |format|
if @contrato.save
        format.html { redirect_to @contrato, notice: 'Contrato was
successfully created.' }
        format.json { render json: @contrato, status: :created,
location: @contrato }
else
        format.html { render action: "new" }
        format.json { render json: @contrato.errors, status:
:unprocessable_entity }
end
end
end
  # PUT /contratos/1
  # PUT /contratos/1.json
def update
    @contrato = Contrato.find(params[:id])
    respond_to do |format|
if @contrato.update_attributes(params[:contrato])
```

```
        format.html { redirect_to @contrato, notice: 'Contrato was
successfully updated.' }

        format.json { head :ok }

else

        format.html { render action: "edit" }

        format.json { render json: @contrato.errors, status:
:unprocessable_entity }

end

end

end

  # DELETE /contratos/1

  # DELETE /contratos/1.json

def destroy

    @contrato = Contrato.find(params[:id])

    @contrato.destroy

    respond_to do |format|

      format.html { redirect_to contratos_url }

      format.json { head :ok }

end

end

end
```

docentes_controller.rb

```
class DocentesController < ApplicationController

  before_filter :authenticate_user!

  # GET /docentes

  # GET /docentes.json

def index

    @docentes = Docente.all

    respond_to do |format|

      format.html # index.html.erb

      format.json { render json: @docentes }
```

```
end
end
  # GET /docentes/1
  # GET /docentes/1.json
def show
    @docente = Docente.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @docente }
end
end
  # GET /docentes/new
  # GET /docentes/new.json
def new
    @docente = Docente.new
    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @docente }
end
end
  # GET /docentes/1/edit
def edit
    @docente = Docente.find(params[:id])
end
  # POST /docentes
  # POST /docentes.json
def create
    @docente = Docente.new(params[:docente])
    respond_to do |format|
if @docente.save
        format.html { redirect_to @docente, notice: 'Docente was
successfully created.' }
```

```
        format.json { render json: @docente, status: :created,
location: @docente }

else

        format.html { render action: "new" }

        format.json { render json: @docente.errors, status:
:unprocessable_entity }

end

end

end

  # PUT /docentes/1

  # PUT /docentes/1.json

def update

    @docente = Docente.find(params[:id])

    respond_to do |format|

if @docente.update_attributes(params[:docente])

        format.html { redirect_to @docente, notice: 'Docente was
successfully updated.' }

        format.json { head :ok }

else

        format.html { render action: "edit" }

        format.json { render json: @docente.errors, status:
:unprocessable_entity }

end

end

end

  # DELETE /docentes/1

  # DELETE /docentes/1.json

def destroy

    @docente = Docente.find(params[:id])

    @docente.destroy

    respond_to do |format|

      format.html { redirect_to docentes_url }

      format.json { head :ok }
```

```
end
end
end
```

institucions_controller.rb

```
class InstitucionsController < ApplicationController
  before_filter :authenticate_user!
  # GET /institucions
  # GET /institucions.json
def index
    @institucions = Institucion.all
    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @institucions }
end
end
  # GET /institucions/1
  # GET /institucions/1.json
def show
    @institucion = Institucion.find(params[:id])
    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @institucion }
end
end
  # GET /institucions/new
  # GET /institucions/new.json
def new
    @institucion = Institucion.new
    respond_to do |format|
      format.html # new.html.erb
```

```
      format.json { render json: @institucion }
end
end
  # GET /institucions/1/edit
def edit
    @institucion = Institucion.find(params[:id])
end
  # POST /institucions
  # POST /institucions.json
def create
    @institucion = Institucion.new(params[:institucion])
    respond_to do |format|
if @institucion.save
        format.html { redirect_to @institucion, notice: 'Institucion
was successfully created.' }
        format.json { render json: @institucion, status: :created,
location: @institucion }
else
        format.html { render action: "new" }
        format.json { render json: @institucion.errors, status:
:unprocessable_entity }
end
end
end
  # PUT /institucions/1
  # PUT /institucions/1.json
def update
    @institucion = Institucion.find(params[:id])
    respond_to do |format|
if @institucion.update_attributes(params[:institucion])
        format.html { redirect_to @institucion, notice: 'Institucion
was successfully updated.' }
        format.json { head :ok }
```

```
else
        format.html { render action: "edit" }
        format.json { render json: @institucion.errors, status:
:unprocessable_entity }
end
end
end
  # DELETE /institucions/1
  # DELETE /institucions/1.json
def destroy
    @institucion = Institucion.find(params[:id])
    @institucion.destroy
    respond_to do |format|
      format.html { redirect_to institucions_url }
      format.json { head :ok }
end
end
end
```

paginas_controller.rb

```
class PaginasController < ApplicationController
before_filter :authenticate_user!
def inicio
end
def ayuda
end
def acercade
end
def ingresos
end
end
```

PRINCIPALES ARCHIVOS DE CONFIGURACIÓN

Gemfile

```
source 'http://rubygems.org'


gem 'rails', '3.1.3'
# Bundle edge Rails instead:
# gem 'rails',     :git => 'git://github.com/rails/rails.git'
gem 'sqlite3'
gem 'therubyracer'
gem 'execjs'
gem 'devise'
# Gems used only for assets and not required
# in production environments by default.
group :assets do
gem 'sass-rails',   '~> 3.1.5'
gem 'coffee-rails', '~> 3.1.1'
gem 'uglifier', '>= 1.0.3'
end
gem 'jquery-rails'
# To use ActiveModel has_secure_password
# gem 'bcrypt-ruby', '~> 3.0.0'
# Use unicorn as the web server
# gem 'unicorn'
# Deploy with Capistrano
# gem 'capistrano'
# To use debugger
# gem 'ruby-debug19', :require => 'ruby-debug'


group :test do
  # Pretty printed test output
```

```
gem 'turn', '0.8.2', :require => false
end
```

schema.rb

```
ActiveRecord::Schema.define(:version => 20120305212739) do
  create_table "contratos", :force => true do |t|
    t.integer  "docente_id"
    t.integer  "institucion_id"
    t.string   "observaciones"
    t.datetime "created_at"
    t.datetime "updated_at"
end

  create_table "docentes", :force => true do |t|
    t.string   "nombres"
    t.string   "apellidos"
    t.string   "ci"
    t.string   "direccion"
    t.string   "telefono"
    t.string   "email"
    t.datetime "created_at"
    t.datetime "updated_at"
end

  create_table "institucions", :force => true do |t|
    t.string   "nombre"
    t.string   "ubicacion"
    t.datetime "created_at"
    t.datetime "updated_at"
end

  create_table "users", :force => true do |t|
```

```
    t.string   "username"
    t.string   "email",                  :default => "", :null =>
false
    t.string   "encrypted_password",     :default => "", :null =>
false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          :default => 0
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
    t.datetime "created_at"
    t.datetime "updated_at"
end


  add_index "users", ["email"], :name => "index_users_on_email",
:unique => true
  add_index "users", ["reset_password_token"], :name =>
"index_users_on_reset_password_token", :unique => true
  add_index "users", ["username"], :name => "index_users_on_username"


end
```

routes.rb

```
Prototipo::Application.routes.draw do
resources :contratos
resources :institucions
resources :docentes
  #devise_for :users
devise_for :users
match '/inicio', to: 'paginas#inicio'
```

```
match '/ayuda', to: 'paginas#ayuda'

match '/acercade', to: 'paginas#acercade'

match '/ingresos', to: 'paginas#ingresos'

match '/consultas', to: 'paginas#consultas'

match '/docentes', to: 'docentes#index'

match '/ingreso_docentes', to: 'docentes#new'

match '/instituciones', to: 'institucions#index'

match '/ingreso_instituciones', to: 'institucions#new'

  # The priority is based upon order of creation:

  # first created -> highest priority.

  # Sample of regular route:

  #   match 'products/:id' => 'catalog#view'

  # Keep in mind you can assign values other than :controller and
:action

  # Sample of named route:

  #   match 'products/:id/purchase' => 'catalog#purchase', :as =>
:purchase

  # This route can be invoked with purchase_url(:id => product.id)

  # Sample resource route (maps HTTP verbs to controller actions
automatically):

  #   resources :products


  # Sample resource route with options:

  #   resources :products do

  #     member do

  #       get 'short'

  #       post 'toggle'

  #     end

  #

  #     collection do

  #       get 'sold'

  #     end

  #   end
```

```
  # Sample resource route with sub-resources:
  #   resources :products do
  #     resources :comments, :sales
  #     resource :seller
  #   end
  # Sample resource route with more complex sub-resources
  #   resources :products do
  #     resources :comments
  #     resources :sales do
  #       get 'recent', :on => :collection
  #     end
  #   end
  # Sample resource route within a namespace:
  #   namespace :admin do
  #     # Directs /admin/products/* to Admin::ProductsController
  #     # (app/controllers/admin/products_controller.rb)
  #     resources :products
  #   end
  # You can have the root of your site routed with "root"
  # just remember to delete public/index.html.
  # root :to => 'welcome#index'
root to: 'paginas#inicio'
  # See how all your routes lay out with "rake routes"
  # This is a legacy wild controller route that's not recommended for
RESTful applications.
  # Note: This route will make all actions in every controller
accessible via GET requests.
  # match ':controller(/:action(/:id(.:format)))'
end
```

Entorno de Producción

development.rb

```
Prototipo::Application.configure do

  # Settings specified here will take precedence over those in
config/application.rb

  # In the development environment your application's code is reloaded
on

  # every request.  This slows down response time but is perfect for
development

  # since you don't have to restart the web server when you make code
changes.

  config.cache_classes = false

  # Log error messages when you accidentally call methods on nil.

  config.whiny_nils = true

  # Show full error reports and disable caching

  config.consider_all_requests_local       = true

  config.action_controller.perform_caching = false

  # Don't care if the mailer can't send

  config.action_mailer.raise_delivery_errors = false

  # Print deprecation notices to the Rails logger

  config.active_support.deprecation = :log

  # Only use best-standards-support built into browsers

  config.action_dispatch.best_standards_support = :builtin

  # Do not compress assets

  config.assets.compress = false

  # Expands the lines which load the assets

  config.assets.debug = true

  config.action_mailer.default_url_options = { :host =>
'localhost:3000' }

end
```

NOTA: Los archivos de configuración de los entornos de desarrollo, pruebas y producción son similares, únicamente cambian sus parámetros, por ejemplo la base de datos.

Django Framework.

Configuración de la conexión Base de Datos en settings.py

```
# Django settings for WebAcademico project.

DEBUG = True
TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Your Name', 'your_email@example.com'),
)

MANAGERS = ADMINS

DATABASES = {
    'default': {
        'ENGINE': 'postgresql_psycopg2', # Add 'postgresql_psycopg2',
'postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'BDAcademico',                      # Or path to
database file if using sqlite3.
        'USER': 'postgres',                      # Not used with
sqlite3.
        'PASSWORD': 'pauldav1d',                  # Not used with
sqlite3.
        'HOST': '127.0.0.1',                      # Set to empty
string for localhost. Not used with sqlite3.
        'PORT': '5432',                      # Set to empty string for
default. Not used with sqlite3.
    }
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
# If running in a Windows environment this must be set to the same as
your
# system time zone.
TIME_ZONE = 'America/Guayaquil'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'es-ec'

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as
not
# to load the internationalization machinery.
USE_I18N = True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale
USE_L10N = True
```

```
# Absolute filesystem path to the directory that will hold user-
uploaded files.
# Example: "/home/media/media.lawrence.com/media/"
MEDIA_ROOT = ''

# URL that handles the media served from MEDIA_ROOT. Make sure to use
a
# trailing slash.
# Examples: "http://media.lawrence.com/media/",
"http://example.com/media/"
MEDIA_URL = ''

# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static
files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT = ''

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL = '/static/'

# URL prefix for admin static files -- CSS, JavaScript and images.
# Make sure to use a trailing slash.
# Examples: "http://foo.com/static/admin/", "/static/admin/".
ADMIN_MEDIA_PREFIX = '/static/admin/'

# Additional locations of static files
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or
"C:/www/django/static".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

# List of finder classes that know how to find static files in
# various locations.
STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
#    'django.contrib.staticfiles.finders.DefaultStorageFinder',
)

# Make this unique, and don't share it with anybody.
SECRET_KEY = '4=s1=#%%*bj#&_3u3-pst_lo#ehrrmf4rwz#+smni!l)oc%!^w'

# List of callables that know how to import templates from various
sources.
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
#     'django.template.loaders.eggs.Loader',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.csrf.CsrfResponseMiddleware',
)

ROOT_URLCONF = 'WebAcademico.urls'


TEMPLATE_DIRS = (
                 'C:\Users\Paul David\Proyectos
Django\WebAcademico\Plantillas'
    # Put strings here, like "/home/html/django_templates" or
"C:/www/django/templates".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'WebAcademico.matricula',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error.
# See http://docs.djangoproject.com/en/dev/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
    }
}
```

Especificación de las URL en urls.py

```
from django.conf.urls.defaults import patterns, include, url
from WebAcademico.matricula.views import *#hora_actual, buscar_form,
buscar,contact, ingresar_estudiante
from django.contrib.auth.views import login, logout

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'WebAcademico.views.home', name='home'),
    # url(r'^WebAcademico/', include('WebAcademico.foo.urls')),

    # Uncomment the admin/doc line below to enable admin
documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$','WebAcademico.matricula.views.inicio'),
    url(r'^busqueda/$', buscar_form),
    url(r'^buscar/$', buscar),
    url(r'^ingreso_institucion/$', ingresar_institucion),
    url(r'^ingreso_docente/$', ingresar_docente),
    url(r'^accounts/login/$','django.contrib.auth.views.login',
{'template_name': 'WebAcademicologin.html'}),
    url(r'^accounts/logout/$', logout),
)
```

Creación de los modelos de la aplicación en models.py MODELO

```
from django.db import models

# Create your models here.
class Institucion(models.Model):
    nombre = models.CharField(max_length = 30)

    def __str__(self):
        return self.nombre

class Docente(models.Model):
    nombre = models.CharField(max_length = 30)
    apellido = models.CharField(max_length = 30)
    direccion = models.CharField(max_length = 50)
    telefono = models.CharField(max_length = 9)
    fecha_nacimiento = models.DateField()
    email = models.EmailField()

    def __str__(self):
        return self.nombre
```

Creación de los formularios de la aplicación en forms.py

```
from django import forms
```

```
from models import Estudiante, Docente, Materia, Matricula
from django.contrib.auth.models import User


class InstitucionForm(forms.ModelForm):
    class Meta:
        model = Estudiante

class DocenteForm(forms.ModelForm):
    class Meta:
        model = Docente
```

Especificación de los elementos para el administrador de Django en admin.py

```
from django.contrib import admin
from WebAcademico.matricula.models import
Estudiante,Docente,Materia,Matricula

admin.site.register(Estudiante)
admin.site.register(Docente)
admin.site.register(Materia)
admin.site.register(Matricula)
```

Creación de los métodos en la capa del controlador en views.py CONTROLADOR

```
from django.http import HttpResponse
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from django.template import Template, Context, RequestContext, loader
from datetime import datetime
from django.contrib.auth import authenticate, login
from WebAcademico.matricula.forms import *#EstudianteForm, UserForm,
ContactForm
from WebAcademico.matricula.models import Estudiante
# Create your views here.


def hora_actual(request):
    now = datetime.now()
    return render_to_response('home.html',{'hora':now})

def info_vista(request):
    return HttpResponse("Bienvenido a la pagina %s"%request.path)


def buscar_form(request):
    return render_to_response('busca_form.html')

def buscar(request):
    errors = [30]
    if 'q' in request.GET:
        q = request.GET['q']
        if not q:
            errors.append('Ingrese termino de busqueda')
        elif len(q)>20:
            errors.append('Ingrese menos de 20 caracteres')
        else:
```

```
            estudiantes =
Estudiante.objects.filter(nombre__icontains=q)
            return
render_to_response('resultado_busqueda.html',{'estudiantes':estudiante
s,'query':q})
    return render_to_response('busca_form.html',{'errors':errors})

def ingresar_institucion(request):
     form = InstitucionForm(request.POST or None)
     if request.POST and form.is_valid():
         form.save()
         return HttpResponseRedirect('/estudiante/')
     return
render_to_response('add_institucion.html',{'form':form},context_instan
ce=RequestContext(request))

def ingresar_docente(request):
     form = DocenteForm(request.POST or None)
     if request.POST and form.is_valid():
         form.save()
         return HttpResponseRedirect('/gracias/')
     return
render_to_response('add_docente.html',{'form':form},context_instance=R
equestContext(request))

def inicio(request):
     now = datetime.now()
     return render_to_response('index.html',{'hora':now},
context_instance=RequestContext(request))
```

Creación de las interfaces web de la aplicación VISTA

base.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Web Academico</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link       href="{{STATIC_URL}}css/style.css"       rel="stylesheet"
type="text/css" />
<script      type="text/javascript"      src="{{STATIC_URL}}js/cufon-
yui.js"></script>
<script                                          type="text/javascript"
src="{{STATIC_URL}}js/arial.js"></script>
<script                                          type="text/javascript"
src="{{STATIC_URL}}js/cuf_run.js"></script>
{%block head%}
{%endblock%}
</head>
<body>
<div class="main">
  <div class="header">
    <div class="header_resize">
      <div class="logo">
        <h1><a href=""><span>Web</span>Academico<br />
```

```
          <small>Sistema Academico Institucional</small></a></h1>
          <p>Ecuador {{hora}}</p>
        </div>
        <div class="searchform">
          <form    id="formsearch"    name="formsearch"    method="post"
action="#">
          <input                                    name="button_search"
src="{{STATIC_URL}}images/search_btn.gif"         class="button_search"
type="image" />
          <span>
          <input      name="editbox_search"      class="editbox_search"
id="editbox_search" maxlength="80" value="" type="text" />
          </span>
          </form>
        </div>
        <div class="clr"></div>
        <div class="menu">
          <ul>
          <li                                              class="active"><a
href="/inicio"><span>Inicio</span></a></li>
          <li><a href="/admin "><span>Administrar</span></a></li>
          <li><a href="/add_inst"><span>Consultas</span></a></li>
          <li><a href="/add_doc"><span>Blog</span></a></li>
          <li><a href="/contacto"><span>Contacto</span></a></li>
          </ul>
        </div>
        <div class="clr"></div>
      </div>
    </div>
    <div class="clr"></div>
    <div class="hbg">
      <div class="hbg_resize">
        <p><img    src="{{STATIC_URL}}images/readmore.gif"    width="112"
height="33" alt="" border="0" align="right"/></p>
        <p><a    href="#"><img    src="{{STATIC_URL}}images/readmore.gif"
width="112" height="33" alt="" border="0" /></a></p>
      </div>
    </div>
    <div class="content">
      <div class="content_resize">
        <div class="mainbar">
            <h2 class="star"></h2></div>
                    <div class="gadget">
                   {%block content%}
                 {%endblock%}
              </div>
              <div class="col c3">
        </div>
        </div>
        <div class="clr"></div>
      </div>
    </div>
    <div class="fbg">
      <div class="fbg_resize">
        <div class="clr"></div>
      </div>
    </div>
    <div class="footer">
      <div class="footer_resize">
```

```
    </div>
    <div class="clr"></div>
  </div>
</div>
</body>
</html>
```

Plantillas heredadas de la plantilla base.

index.html

```
{% extends "base.html" %}
{%block content%}
<fieldset style="border: 1px solid #C0C0C0">
      <div align="center" id="seleccion">
      <p align="center"><h2>Sistema de Contratacion Docente</h2></p>
      <img src="{{STATIC_URL}}images/arquitectura.jpg" height="500"
width="700" />
      </div>
</fieldset>
<h3><b>Sistema de Contratacion Docente SISCODOCH</b></h3>
<div align="justify">
      <p>
            <h3>
                  Descripcion del modulo de pruebas
            </h3>
      </p>
</div>
```

inicio.html

```
<span>Hola,
    <span id="nombre_usuario">
            {{ user.username }}
    </span>. Gracias por iniciar sesión.
</span>

<div id="ultimo_login">Tu último ingreso fue el {{ user.last_login
}}</div>
<p><a href="/logout/">Logout</a></p>
```

entrar.html

```
{% if form.errors%}
    <p class="error">Error al loguear, los datos no coinciden...</p>
{% endif %}
    <div id="div_login_logo">
        <div id="div_form">
            <form action="{% url django.contrib.auth.views.login %}"
method="post">
                        {% csrf_token %}
                    <fieldset style="border: 1px solid #C0C0C0"
width:"100px" height:"200px">
                        <legend><h3>Autenticacion</h3></legend>
                        <label for="username">Usuario:</label>
                        <br/>
```

```
                        <input type="text" name="username" value=""
id="username">
                        <br/>
                        <label for="password">Contraseña:</label>
                        <br/>
                        <input type="password" name="password" value=""
id="password">
                        </br>
                        <input type="submit" value="Ingresar" />
                        <input type="hidden" name="next" value="{{
request.get_full_path }}">
                    </fieldset>
                </form>
        </div>
    </div>
```

add_docente.html

```
{% extends base.html %}
{%block content%}
<h1>Ingreso de Docentes</h1>
<form method=”post” action””>
<table>
  {{form.as_table}}
</table>
<input type = “submit” value = “Guardar”/>
</form>
{%endblock%}
```

add_institucion.html

```
{% extends base.html %}
{%block content%}
<h1>Ingreso de Instituciones</h1>
<form method=”post” action””>
<table>
  {{form.as_table}}
</table>
<input type = “submit” value = “Guardar”/>
</form>
{%endblock%}
```

Estilos Utilizados.

style.css

```
@charset "utf-8";
body    {    margin:0;    padding:0;    width:100%;    background:#fff
url(http://127.0.0.1:8000/static/images/main_bg.jpg)     top     center
repeat-x;  color:#5f5f5f;  font:normal  12px/1.8em  Arial,  Helvetica,
sans-serif;}
html, .main { padding:0; margin:0; background-color:#e9e9e9;}
.clr  {  clear:both;  padding:0;  margin:0;  width:100%;  font-size:0px;
line-height:0px;}
h1  {  margin:0;  padding:29px  0;  color:#fff;  font:normal  40px/1.2em
Arial, Helvetica, sans-serif;}
```

```
h1 a, h1 a:hover { color:#fff; text-decoration:none;}
h1 span { color:#009933;}
h1 small { padding-left:38px; font:normal 13px/1.2em Arial, Helvetica,
sans-serif;}
h2 { font:bold 24px Arial, Helvetica, sans-serif; color:#323a3f;
padding:8px 0; margin:8px 0;}
p { margin:8px 0; padding:0 0 8px 0; font:normal 12px/1.8em Arial,
Helvetica, sans-serif;}
a { color:#0066FF; text-decoration:none;}

.header, .hbg, .content, .fbg, .footer { margin:0; padding:0;}
.hbg_resize, .content_resize, .fbg_resize { margin:0 auto;
padding:24px 40px; width:900px; background-color:#fff; -moz-border-
radius:10px;}

/* header */
.header {
background:url(http://127.0.0.1:8000/static/images/header_bg.gif)
repeat-x top;}
.header_resize { margin:0 auto; padding:0; width:900px;}
.header .logo { width:auto; float:left;}
.header .menu ul { margin:1px 0 0 0; padding:0; float:right;
width:auto; list-style:none;}
.header .menu ul li { margin:0; float:left;}
.header .menu ul li a { margin:0; padding:12px 10px; color:#5f5f5f;
text-decoration:none; line-height:36px;}
.header .menu ul li a span { margin:0; padding:12px 0 12px 10px;}
.header .menu ul li a:hover, .header .menu ul li.active a {
color:#fff;
background:url(http://127.0.0.1:8000/static/images/menu_r.gif) no-
repeat right top;}
.header .menu ul li a:hover span, .header .menu ul li.active a span {
background:url(http://127.0.0.1:8000/static/images/menu.gif) repeat-x
top;}

/* search */
#formsearch { margin:0; padding:36px 0 0 0;}
#formsearch span { display:block; margin:7px 0 6px 0; padding:0 8px 0
11px !important; padding:0 4px 0 6px; float:right; width:170px;
background:#fff url(http://127.0.0.1:8000/static/images/search.gif)
no-repeat top left;}
#formsearch input.editbox_search { margin:0; padding:6px 0;
float:left; width:170px; height:12px; border:none; background:none;
font:normal 13px/16px Arial, Helvetica, sans-serif; color:#a8acb2;}
#formsearch input.button_search { margin:7px 0 0 0; padding:0;
border:none; float:right;}

/* hbg */
.hbg { padding:16px 0;}
.hbg_resize { padding-right:486px; width:454px; height:291px;
background:#fff url(http://127.0.0.1:8000/static/images/hbg_img.jpg)
no-repeat top right;}

/* content */
.content { padding-bottom:16px;}
.content .mainbar { margin:0; padding:0; float:left; width:620px;}
.content .mainbar .article, .content .sidebar .gadget { margin:0;
padding:0 0 16px 0;}
.content .sidebar { margin:0; padding:0; float:right; width:220px;}
```

```
ul.sb_menu, ul.ex_menu { margin:0; padding:0; list-style:none;}
ul.sb_menu li, ul.ex_menu li { margin:0; padding:0; border-bottom:1px
dashed #e9e5e5;}
ul.ex_menu li { color:#959595;}
ul.sb_menu li a, ul.ex_menu li a { margin:0; padding-left:24px;
color:#5f5f5f;                                    text-decoration:none;
background:url(http://127.0.0.1:8000/images/li.gif) no-repeat 0 0;
line-height:1.8em;}
ul.sb_menu li a:hover, ul.ex_menu li a:hover { color:#0066FF; font-
weight:bold;
background:url(http://127.0.0.1:8000/images/li_active.gif) no-repeat 0
0;}
ul.sb_menu li { padding:4px 0;}
ul.ex_menu li { padding:8px 0;}
ul.sb_menu li a:hover { text-decoration:underline;}
ul.ex_menu li a:hover { text-decoration:none;}

/* subpages */
.content .mainbar .comment { margin:0; padding:16px 0 0 0;}
.content .mainbar .comment img.userpic { border:1px solid #dedede;
margin:10px 16px 0 0; padding:0; float:left;}

/* fbg */
.fbg { margin:0; padding:0;}
.fbg_resize img { border:1px solid #dedede;}
.fbg_resize .col { margin:0; float:left;}
.fbg_resize .c1 { padding:0 16px 0 0; width:290px;}
.fbg_resize .c2 { padding:0 16px; width:290px;}
.fbg_resize .c3 { padding:0 0 0 16px; width:240px;}
.fbg_resize .c1 img { margin:8px 16px 16px 0; padding:0; float:left;}
.fbg_resize .c3 img { margin:4px; padding:0;}

/* footer */
.footer { margin:0; padding:0;}
.footer_resize { margin:0 auto; padding:8px 32px; width:962px;}
.footer_resize p.lf { float:left; width:auto;}
ul.fmenu { margin:8px 0; padding:0; list-style:none; float:right;
width:auto;}
ul.fmenu li { margin:0; padding:0 8px; float:left;}
ul.fmenu li a { color:#5f5f5f; text-decoration:none;}
ul.fmenu li a:hover, ul.fmenu li.active a { color:#0066FF;}
ul.fmenu li a:hover { text-decoration:underline;}

/* form */
form { margin:0; padding:0;}
ol { margin:0; padding:0; list-style:none;}
ol li { margin:0; padding:0; display:block; clear:both;}
ol li label { display:block; margin:0; padding:16px 0 0 0;}
ol li input.text { width:480px; border:1px solid #c0c0c0; margin:2px
0; padding:5px 2px; height:16px; background:#fff;}
ol li textarea { width:480px; border:1px solid #c0c0c0; margin:2px 0;
padding:2px; background:#fff;}
ol li .send { margin:16px 0 0 0;}
```

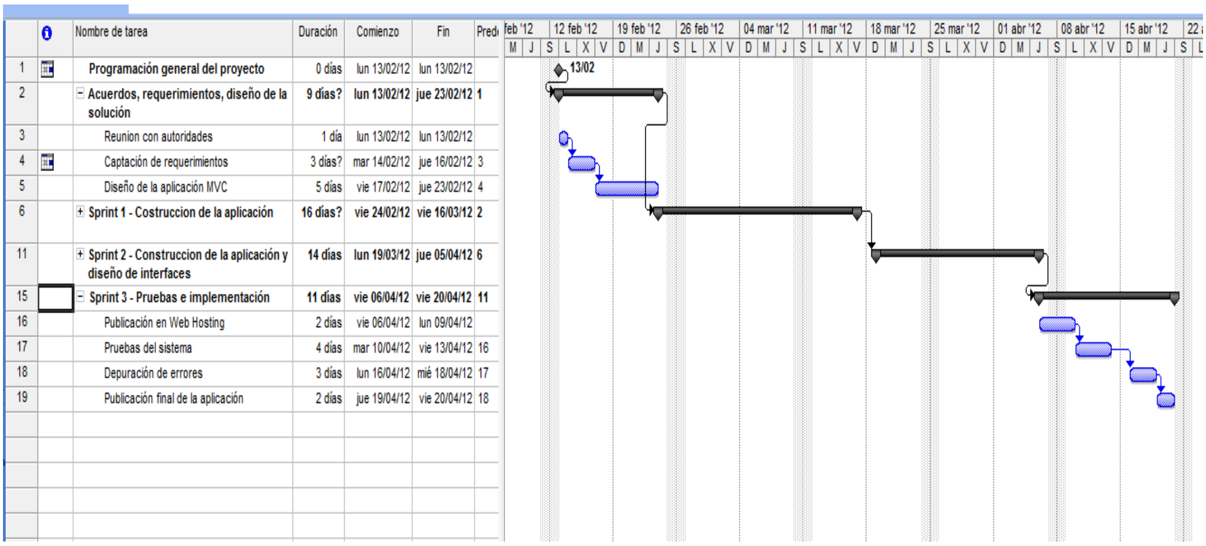
estilo.css

```
body {
       padding-left: 11em;
```

```
  font-family: Georgia, "Times New Roman",
        Times, serif;
  color: #FFFFFF;
  background-color: #0000FF }
  ul.navbar {
  list-style-type: none;
  padding: 0;
  margin: 0;
  position: absolute;
  top: 2em;
  left: 1em;
  width: 9em }
h1 {
  font-family: Helvetica, Geneva, Arial,
        SunSans-Regular, sans-serif }
  ul.navbar li {
  background: white;
  margin: 0.5em 0;
  padding: 0.3em;
  border-right: 1em solid black }
ul.navbar a {
  text-decoration: none }
a:link {
  color: blue }
a:visited {
  color: purple }
address {
  margin-top: 1em;
  padding-top: 1em;
  border-top: thin dotted }
```

ANEXO 2

Planificación y Agenda del proyecto según SCRUM



Agenda del desarrollo de las actividades de los SPRINTS

PRODUCT BACKLOG - SISCONDECH.

ID	Tarea
1	**SPRINT 1**
1.1	Inicio del proyecto
1.1.1	Creación del nuevo proyecto
1.2	Construcción de la aplicación MVC
1.2.1	Construcción del módulo de Acceso (login)
1.2.2	Construcción del módulo de ingresos
1.2.2	Construcción del módulo de consultas
1.2.2	Construcción del módulo de contenido estático
2	**SPRINT 2**
2.1	Depuración del producto entregado en el Sprint 1
2.2	Continuación de la construcción de la aplicación MVC
2.2.1	Implementación de método buscar para docentes e instituciones para la contratación
2.2.2	Implementación autocompletado de docentes e instituciones para la contratación

ID	Tarea
2.2.2	Mejora de las Restricciones en el modelo
2.3	Diseño de las interfaces
3	**SPRINT 3**
3.1	Depuración de la aplicación
3.2	Publicación en WebHosting
3.3	Pruebas del Sistema
3.4	Corrección de posibles errores
3.5	Publicación final

Sprint Backlog 1

SPRINT BACKLOG 1 - Construcción de la aplicación

ID	Tarea	Tipo	Estado	Responsable
1	**Inicio del proyecto**			
1.1	**Creación del nuevo proyecto**	Desarrollo	Finalizado	Paul Cumba
1.2	Conexión con Base de Datos	Base de datos / Desarrollo	Finalizado	Paul Cumba
1.2.1	Crear Base de Datos Postgresql	Base de datos	Finalizado	Byron Barreno
1.2.2	Crear usuario para la base de datos	Base de datos	Finalizado	Byron Barreno
2	**Construcción de la aplicación MVC**		Finalizado	
2.1	**Construcción**		Finalizado	

ID	Tarea	Tipo	Estado	Responsable
	del módulo de Acceso (login)			
2.1.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.1.2	Agregación al modelo de la clase Users	Desarrollo	Finalizado	Paul Cumba
2.1.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.1.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.1.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo de ingresos**		Finalizado	
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Agregación al modelo para Docentes, Instituciones, Contratos, y demás tablas para la BD	Desarrollo	Finalizado	Paul Cumba
2.2.3	Creación de	Desarrollo	Finalizado	Paul Cumba

ID	Tarea	Tipo	Estado	Responsable
	vistas			
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo de consultas**		Finalizado	
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Agregación al modelo según las clases para las consultas	Desarrollo	Finalizado	Paul Cumba
2.2.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Construcción del módulo de contenido estático**		Finalizado	
2.2.1	Configuración del	Desarrollo	Finalizado	Paul Cumba

ID	Tarea	Tipo	Estado	Responsable
	controlador			
2.2.2	Edición del modelo para el contenido estático (Home, help, about_us).	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2.2.3	Creación de vistas	Desarrollo	Finalizado	Paul Cumba
2.2.4	Generación de Formularios	Desarrollo	Finalizado	Byron Barreno
2.2.5	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno

Sprint Backlog 2

SPRINT BACKLOG 2 Construcción de la aplicación y diseño de interfaces

ID	Tarea	Tipo	Estado	Responsable
1	**Depuración del producto entregado en el Sprint 1**	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2	**Construcción de la aplicación MVC**			
2.1	**Implementación de un buscar para docentes e instituciones para la contratación**		Finalizado	
2.1.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.1.2	Edición del modelo	Desarrollo	Finalizado	Paul Cumba

ID	Tarea	Tipo	Estado	Responsable
2.1.3	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.2	**Implementación autocompletado de docentes e instituciones para la contratación**			
2.2.1	Configuración del controlador	Desarrollo	Finalizado	Paul Cumba
2.2.2	Edición del modelo	Desarrollo	Finalizado	Paul Cumba
2.2.3	Validación de formularios en el modelo	Desarrollo	Finalizado	Byron Barreno
2.3	Mejora de las Restricciones en el modelo			
2.3.1	Edición del modelo	Desarrollo	Finalizado	Paul Cumba
3	**Diseño de las interfaces**			
3.1	Implementación de hojas de estilo	Diseño de Interfaz	Finalizado	Byron Barreno
3.2	Creación de la plantilla base	Diseño de Interfaz	Finalizado	Byron Barreno
3.3	Integración de la plantilla HTML y sus estilos, con la aplicación	Diseño de Interfaz	Finalizado	Byron Barreno
3.4	Mejora de detalles en la interfaz (Renderización, banner, imágenes, etc)	Diseño de Interfaz	Finalizado	Byron Barreno

Sprint Backlog 3

SPRINT BACKLOG 3 Pruebas e implementación de la aplicación

ID	Tarea	Tipo	Estado	Responsable
1	**Depuración de la aplicación (Sprint 2)**			

ID	Tarea	Tipo	Estado	Responsable
1.1	Personalización de formularios	Desarrollo	Finalizado	Paul Cumba
1.2	Restricciones de acceso a la aplicación	Desarrollo	Finalizado	Paul Cumba
1.3	Inserción de registros reales (Docentes, Instituciones, Contratos, etc.)	Desarrollo	Finalizado	Paul Cumba – Byron Barreno
2	**Publicación en WebHosting**			
2.1	Elección del proveedor Hosting	Redes	Finalizado	Byron Barreno
2.2	Configuración del servidor de BD y servidor Web.	Sistema Operativo	Finalizado	Byron Barreno
2.3	Respaldo de la BD	Base de Datos	Finalizado	Paul Cumba
2.4	Restauración de la BD en el servidor remoto	Base de Datos	Finalizado	Byron Barreno
2.5	Respaldo de la Aplicación.	Desarrollo	Finalizado	Paul Cumba
2.6	Carga de la aplicación hacia el servidor remoto	Desarrollo	Finalizado	Byron Barreno
2.7	Configuración de parámetros en la aplicación, para el funcionamiento en el Hosting	Desarrollo	Finalizado	Paul Cumba
2.8	Adquisición de dominio	Redes	Finalizado	Byron Barreno
3	**Pruebas del Sistema**			
3.1	Carga masiva de registros	Utilización del Sistema	En curso	Usuarios del sistema

ID	Tarea	Tipo	Estado	Responsable
3.2	Edición y eliminación de registros	Utilización del Sistema	En curso	Usuarios del sistema
3.3	Generación de Reportes	Utilización del Sistema	En curso	Usuarios del sistema
4	**Corrección de posibles errores**	Desarrollo	Finalizado	Byron Barreno
5	**Publicación final**	Desarrollo	Finalizado	Byron Barreno

ANEXO 3

Manual de Usuario del Sistema de Contratación Docente

SISTEMA DE SISTEMA DE CONTRATACION DOCENTE

SISCONDECH

MANUAL DE USUARIO

Índice General

Índice de Figuras

1. FUNCIONAMIENTO BASICO DE SISCONDECH

1.1. Introducción al Sistema de Contratación Docente SISCONDECH

EL Sistema de Contratación Docente, es una aplicación web en conjunto con normas e instructivos para el uso de la misma, y así dar facilidad de información y registro de docentes y sus contratos.

La función principal de la aplicación web será registrar los contratos realizados por la DECH a los docentes, y demás funciones que se derivan de esta, como registro de docentes, instituciones, etc. Todo esto con una interfaz amigable, con las debidas restricciones de seguridad y privacidad.

SISCONDECH cuenta con una arquitectura en base al patrón de diseño Modelo Vista Controlador.

1.2. Requerimientos

1.2.1. Requerimientos mínimos del sistema

- ✓ **Sistema Operativo:** Windows XP, Vista, Seven, Cualquier Distribución de Linux.
- ✓ **Navegador de Internet:** IExplorer 7 o superior, Mozilla Firefox, o Google Chrome.
- ✓ **Conexión a Internet:** 256/128 kbps
- ✓ **Procesador:** Intel Pentium4 400MHz
- ✓ **Memoria:** 256 MB mínimo
- ✓ **Video:** mínimo 64 MB
- ✓ **Sonido:** Tarjeta compatible
- ✓ **Hardware Adicional:** Lector de Discos CD-ROM/CD-RW/DVD-ROM/DVD-RW, Impresora, Parlantes.

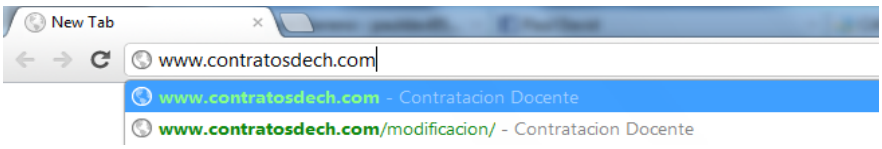
1.2.2. Requerimientos de la plataforma

- ✓ **Servidor Web:** Apache 2 modo FastGCI.
- ✓ **Servidor DNS:** Servicio de nombres de Dominio (Punto NET).
- ✓ **Lenguaje de Programación:** Python 2.5 o Superior.
- ✓ **Framework:** Django 1.3 o superior.
- ✓ **IDE:** Aptana Studio 3.

1.3. Conociendo la interfaz

SISCONDECH presenta una interfaz amigable para el usuario, la misma que maneja un orden en su diseño en la cual se podrá realizar las funciones que se necesiten de una manera fácil y eficiente.

A través de un navegador de internet (iexplorer, firefox o google chrome) situamos la dirección **http://www.contratosdech.com**



URL, dirección del Sistema de Contratación Docente

1.3.1. Componentes de la interfaz

Figura 1: Pantalla de inicio del sistema

Despues de escribir la direccion de la aplicación web en el browser el navegador de internet nos graficara la pagina de inicio del sistema.

Figura 2: Interfaz Principal del Sistema

Elementos:

1. **Browser del navegador de internet:** Es utilizado par ubicar la dirección url de cualquier aplicación web, es la petición hacia el servidor web.
2. **Banner de Presentación:** Identifica mediante una animación Flash la presentación de la institución, se requiere Flash Player para visualizarlo.
3. **Formulario de Autenticación:** Formulario donde los usuarios autorizados podrán acceder al sistema.
4. **Descripción del sistema:** Describe mediante un grafico el sistema de contratación docente.
5. **Enlaces:** Sitio de enlaces de comunicados de la Dirección de Estudios y Sitios de Interés.

1.4. Autentificación y autorización del usuario.

Solamente los usuarios que tengan autorización podrán acceder al sistema. Estos usuarios por son creados a través de las políticas y administración del sitio que de la División de Talento Humano de la DECH. Por motivos de seguridad cada usuario que utilizara SISCONDECH tendrá su nombre de inicio de sección y su contraseña con sus respectivos roles en la manipulación de la información del sistema.

Tipos de Usuarios en el sistema:

- Administrador del Sitio Django.
- Operadores de Contratación Docente

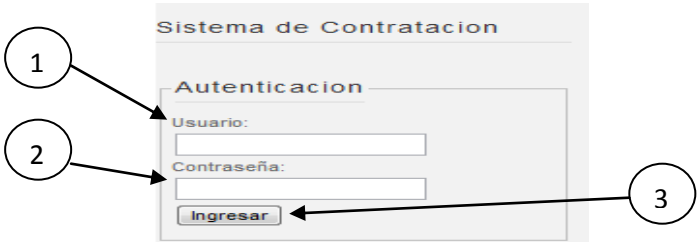


Figura 3: Autenticación de usuarios

1. Nombre de usuario.
2. Contraseña de usuario.
3. Botón de Ingreso.

En caso de que el nombre de inicio de sección o la contraseña sean incorrectos se informara al usuario del error.

Figura 4: Error de inicio de sesión

2. Funcionalidades del Sistema de Contratación Docente SISCONDECH

SISCONDECH consta con un menú principal interactivo mediante el cual el usuario podrá navegar con mejor manera el sitio.

Figura 5: Menú Principal

2.1.Gestor de Ingresos

Administra los ingresos necesarios para realizar un contrato, el cual consta de:

1. Ingreso de Docentes.
2. Ingreso de Instituciones.
3. Especialidad del Docente.

Figura 6: Gestor de Ingresos

2.1.1. Ingreso de Docentes

El ingreso de Docentes permite añadir la información de un docente nuevo el mismo que puede ser contratado.

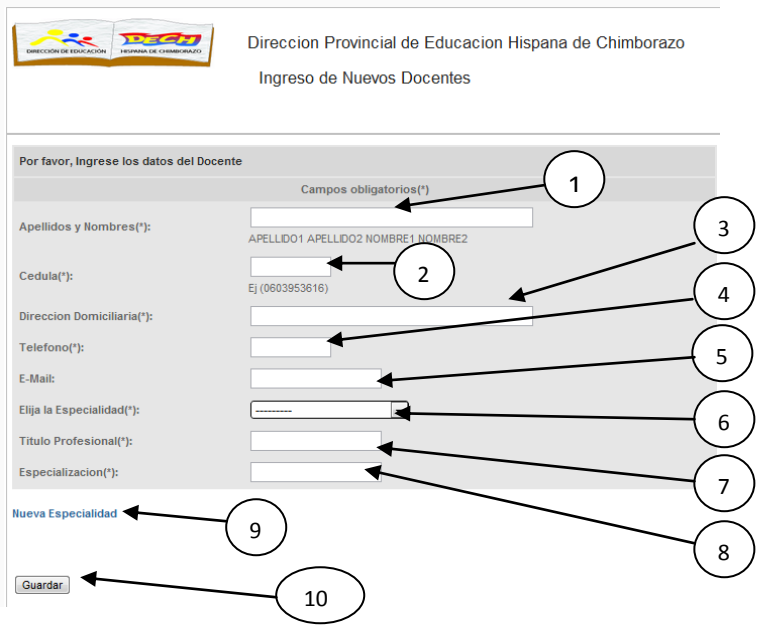


Figura 7: Ingreso de nuevos docentes

En esta pantalla de ingresos de docente se indicara la información personal del docente el cual será contratado por la DECH.

1. Ingreso de los Apellidos y nombres de un docente ej. (CUMBA ARMIJOS PAUL DAVID)
2. Cedula del docente
3. Dirección Domiciliaria.
4. Ubicación del teléfono del docente
5. Ubicación del correo electrónico en caso de que lo tuviese
6. Selección de la especialidad del docente.
7. Indicar el Titulo profesional del Docente.
8. La especialización profesional del docente.
9. Enlace que permite añadir una nueva especialidad (véase 2.1.3)
10. Botón que permite almacenar la información del docente proporcionada por el usuario.

Todos los campos marcados con (*) son obligatorios, en caso de no ubicar alguno el sistema indicara su obligatoriedad.

2.1.2. Ingreso de Instituciones.

Formulario que permite agregar las instituciones que pertenecen a la jurisdicción de la Dirección Provincial de Educación de Chimborazo.

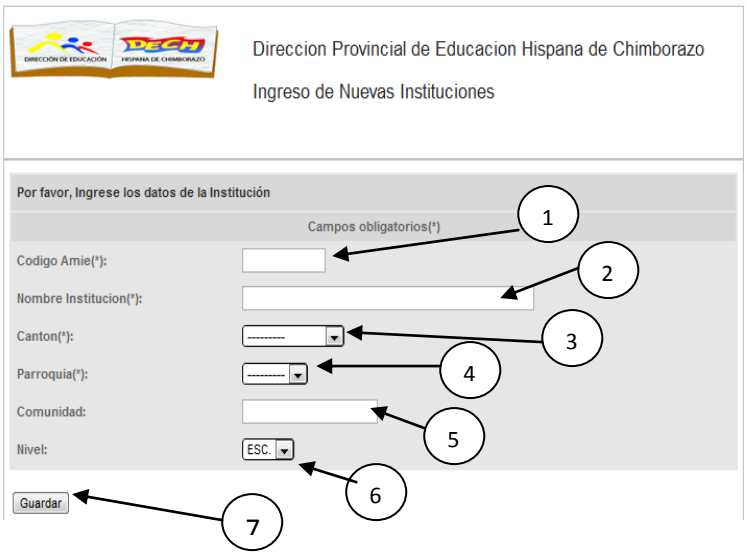


Figura 8: Ingreso de instituciones

1. Código AMIE de la escuela proporcionado por el departamento de estadística.
2. Nombre de la institución.

3. Cantón donde se ubica.
4. Parroquia.
5. Comunidad, en caso de que se encuentre en ella.
6. Nivel de instrucción, este puede ser: (ESC: Escuela, CEB: Centro de Educación Básica, JAR: Jardín, UE: Unidad Educativa).
7. Botón para almacenar la información proporcionada.

2.1.3. Especialidad del Docente.

Son las especialidades en las cuales se desenvolverá el docente en el periodo lectivo para el cual será contratado de acuerdo a sus aptitudes y requerimientos de la DECH.

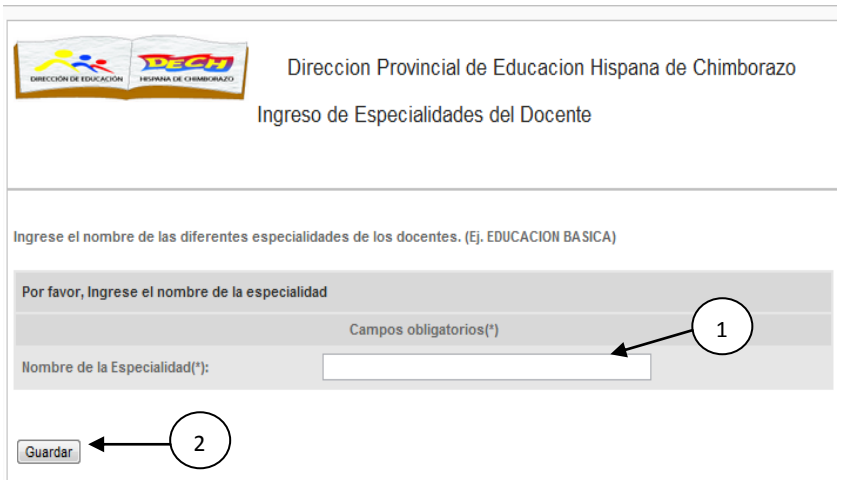


Figura 9: Ingreso de Especialidad

1. Ingreso de la especialidad del docente, esta puede ser (EDUCACION BASICA, MATEMATICAS, CIENCIAS NATURALES, ETC)
2. Botón para almacenar la información proporcionada.

2.2. Gestor de Contratación.

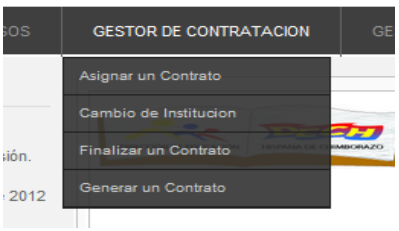


Figura 10: Gestor de Contratación

Administra todo el referente a la contratación de los docentes:

1. Asignar un Contrato
2. Cambio de Institución
3. Finalizar un contrato.
4. Generar un Contrato.

2.2.1. Asignar un Contrato.

Permite a los operadores de contratación asignar los contratos a los docentes de acuerdo a sus requerimientos.

Para asignar un contrato a un docente se debe tener las siguientes consideraciones:

- Los docentes e instituciones ya deben estar registrados previamente en la base de datos del sistema.
- No se puede asignar un número de contrato que ya haya sido asignado.
- Los docentes pueden ser contratados una sola vez siempre y cuando no tenga vigente otro contrato.
- Un docente puede ser contratado en una institución educativa.

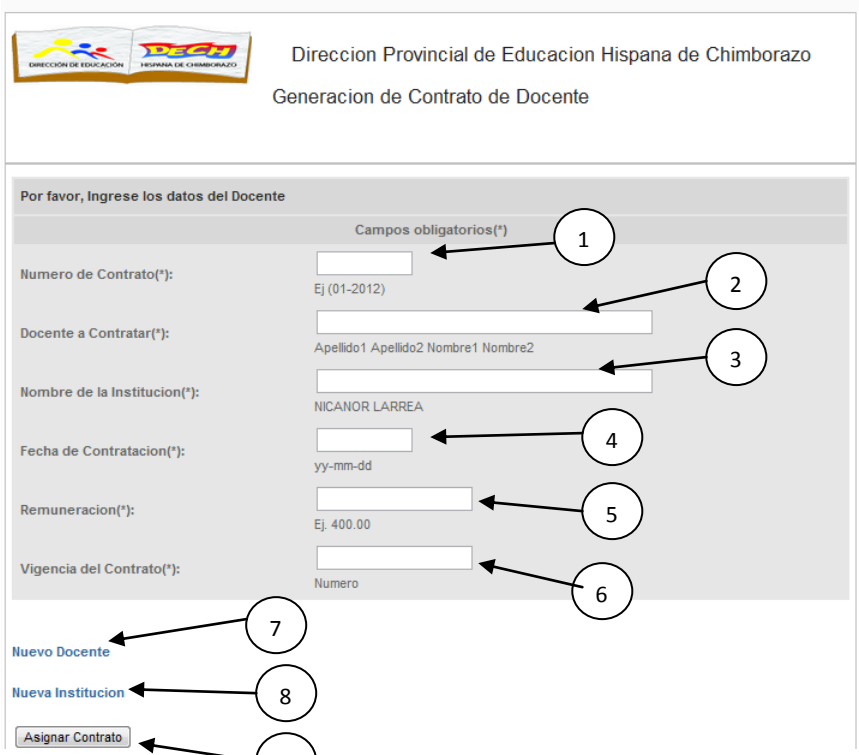


Figura [illegible]: Asignación de un nuevo contrato

1. **Numero del Contrato:** Se debe ingresar en base a la numeración establecida por la dirección de estudios. Ej. (01-2012)
2. **Docente a Contratar:** Indicara el docente que será contratado, el cuadro de texto posee ayuda de búsqueda mediante autocompletado.

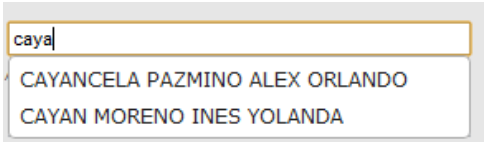


Figura 12: Búsqueda de docente por auto completado

3. **Nombre de la Institución:** Se especifica la institución educativa en la cual trabajara el docente. El cuadro de texto indicara las instituciones existentes mediante búsqueda por autocompletado.

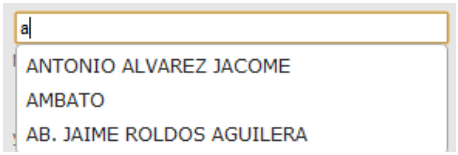


Figura 13: Búsqueda de institución por autocompletado

4. **Fecha de Contratación:** Se indicara la fecha a partir de la cual tiene vigencia el contrato. La selección de la fecha es mediante un datepicker.

Figura 14: Datepicker. Fecha de contratación

5. **Remuneración:** Establece el sueldo que percibirá el docente de acuerdo a su categoría.
6. **Vigencia del Contrato:** El número de años que posee de vigencia el contrato.
7. **Nuevo Docente:** (véase 2.1.1.)
8. **Nueva Institución:** (véase 2.1.2.)
9. **Asignar Contrato:** Botón para asigna el contrato.

2.2.2. Cambio de Institución.

El docente mediante pedido del departamento administrativo puede ser reubicado en otra institución educativa. Este proceso se lo efectúa mediante la búsqueda del docente o el contrato a modificar su institución.

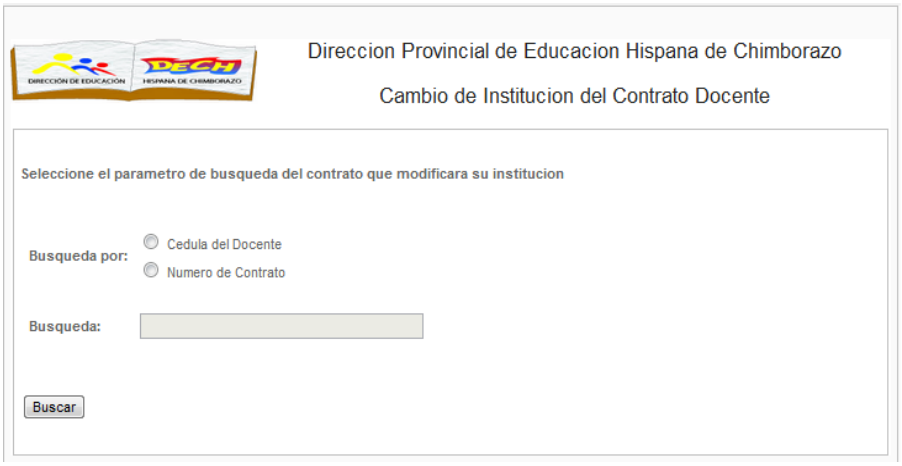


Figura 15: Búsqueda del contrato a modificar

Se deberá seleccionar el parámetro de búsqueda del contrato que se desea modificar, estos parámetros de búsqueda pueden ser por:

- Cedula del Docente
- Número de Contrato

Después de elegir e ingresar el parámetro de búsqueda, procedemos a buscar el contrato a modificar.

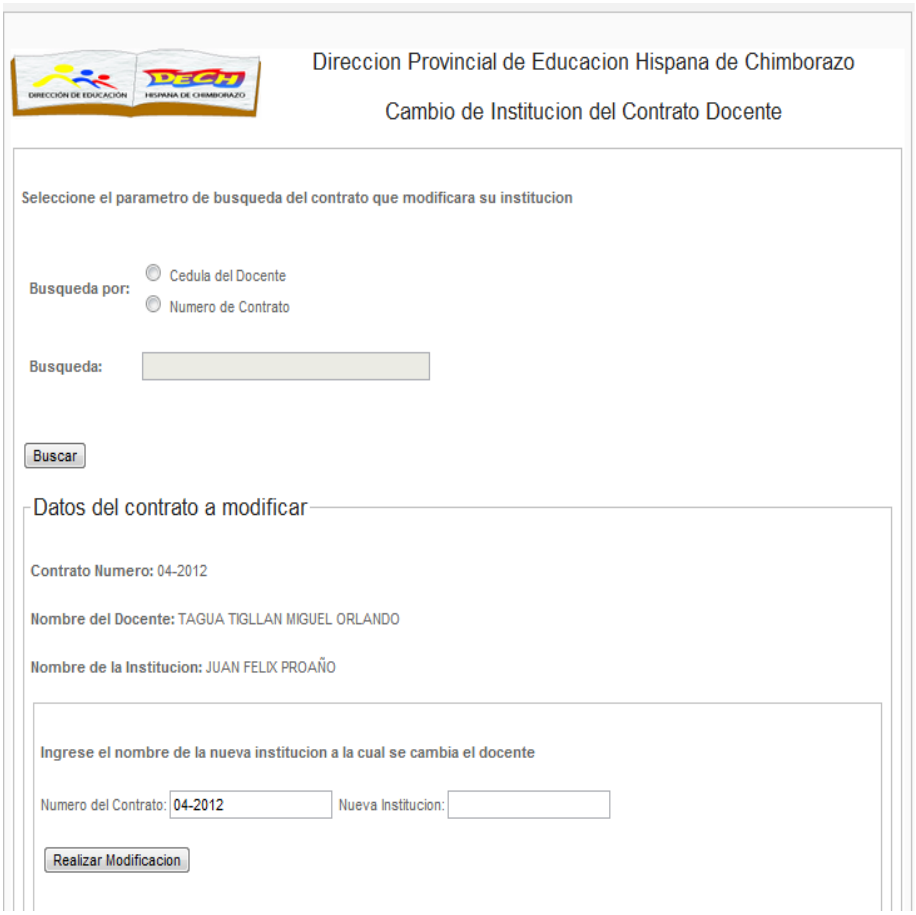


Figura 16: Cambio de institución en el contrato

Después de encontrar el contrato a modificar, procedemos a realizar el cambio, ubicando la nueva institución para luego << Realizar Modificación >>

2.2.3. Finalizar el contrato.

La finalización del contrato se la efectúa de acuerdo a las estipulaciones de la DECH.

Para finalizar un contrato al igual que para el cambio de institución se debe realizar la búsqueda del contrato a finalizar en este caso.

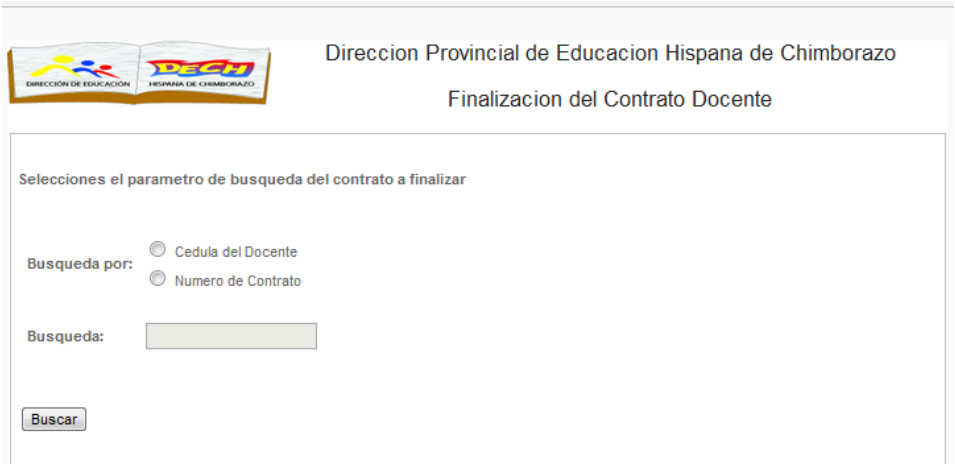


Figura 17: Búsqueda del contrato a finalizar.

Al encontrar el contrato que se desee finalizar se deberá indicar el motivo que finaliza el contrato, para luego proceder a finalizar el contrato del docente.

Figura 18: Finalización del contrato

2.2.4. Generar un contrato.

Después de efectuar la asignación de los contratos el operador de contratación puede generar un documento de contrato físico para la constancia de la contratación.

Para generar un contrato se deberá indicar el nombre del docente del cual se quiere generar el contrato. El docente se buscara mediante autocompletado.

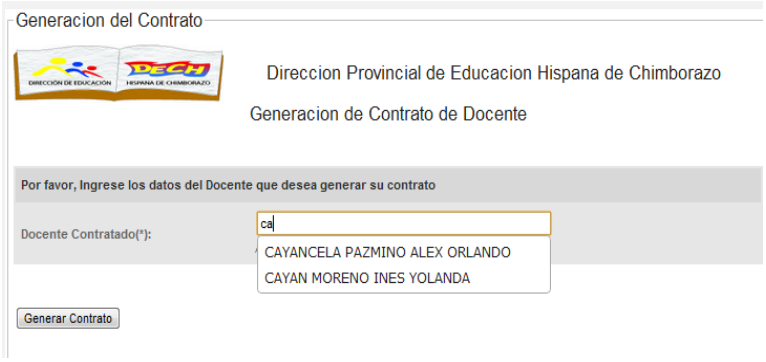


Figura 19: Generación del contrato.

Posteriormente de establecer el docente del cual se desea generar el contrato, procedemos a <<Generar Contrato>>

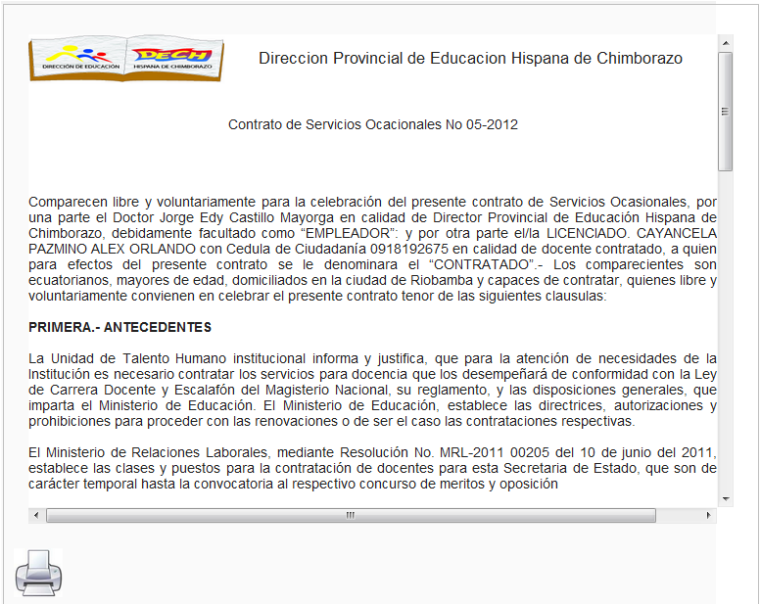


Figura 20: Contrato Generado.

El contrato generado se lo puede imprimir.

2.3. Gestor de Consultas.

La generación de los Reportes se la realizara mediante consultas establecidas por los usuarios operarios de contratación.

Las consultas están clasificadas de acuerdo los requerimientos de la División de Recursos Humanos.

Figura 21: Gestor de Consultas

Clasificación de las consultas:

- **Indicadores de Docentes**
  - ✓ Información de Docente
  - ✓ Docentes Contratados por año
  - ✓ Docentes con contratos activos
  - ✓ Docentes con contratos inactivos
- **Indicadores de Instituciones**
  - ✓ Información de Instituciones
  - ✓ Listado de Instituciones
  - ✓ Instituciones por ubicación
- **Indicadores de Contratación**
  - ✓ Información de contrato
  - ✓ Contratos Activos
  - ✓ Contratos Inactivos
  - ✓ Contratos Finalizados

2.3.1. Presentación de las consultas.

Todas las consultas están disponibles para su impresión, y los indicadores son establecidos de acuerdo a los requerimientos de la DECH.

NOMBRE DE DOCENTE	NUMERO DE CONTRATO	FECHA DE CONTRATACION	ESTADO DEL CONTRATO
PUCHA YEPEZ FAUSTO VINICIO	01-2012	21 de mayo de 2012	ACTIVO
DURAN MARTINEZ LEONARDO RAMIRO	02-2012	21 de mayo de 2012	ACTIVO
TAGUA TIGLLAN MIGUEL ORLANDO	04-2012	21 de mayo de 2012	ACTIVO
CAYANCELA PAZMINO ALEX ORLANDO	05-2012	21 de mayo de 2012	ACTIVO
GAIBOR PEÑA BLANCA ROSA	06-2012	21 de mayo de 2012	ACTIVO
LLERENA VELOZ ROSA MARINA	07-2012	21 de mayo de 2012	ACTIVO
NUÑEZ ROLDAN ARELY GABRIELA	10-2012	21 de mayo de 2012	ACTIVO
ZAMBRANO GUADALUPE FABIAN ERNESTO	08-2012	21 de mayo de 2012	ACTIVO
PAILIACHO ARMIJOS DIEGO PATRICIO	09-2012	21 de mayo de 2012	ACTIVO
DELGADO NOBOA LORENA PATRICIA	12-2012	21 de mayo de 2012	ACTIVO
DUCHI SISA MARIA GRICELDA	13-2012	21 de mayo de 2012	ACTIVO
DOMINGUEZ CABEZAS MARTHA DEL ROCIO	14-2012	21 de mayo de 2012	ACTIVO
DUCHI PATARON LUIS RUPERTO	16-2012	21 de mayo de 2012	ACTIVO
FLORES ADRIANO MARIA TERESA	21-2012	21 de mayo de 2012	ACTIVO

Figura 22: Reporte consultado, docentes contratados por año.

3. Administración del Sitio.

El sitio de administración es un consola en la cual el administrador es el encargado de gestionar toda la información que se requiere y se proporciona por la División de talento Humano de la DECH.

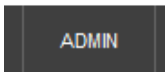


Figura 23: Menú Administración

Al acceder al sitio de administración de SISCONDECH, solo los usuarios con permisos de administrador podrán acceder a la consola de administración.

Figura 24: Autenticación del Administrador

Figura 25: Consola Administrativa

4. Acerca de

Indica los créditos del Sistema de Contratación Docente SISCONDECH

Figura 26: Créditos del sistema

COPYRIGHT

5. Mapa del Sitio

El usuario puede guiarse por medio de un mapa de enlaces que le permitir navegar correctamente por la aplicación.

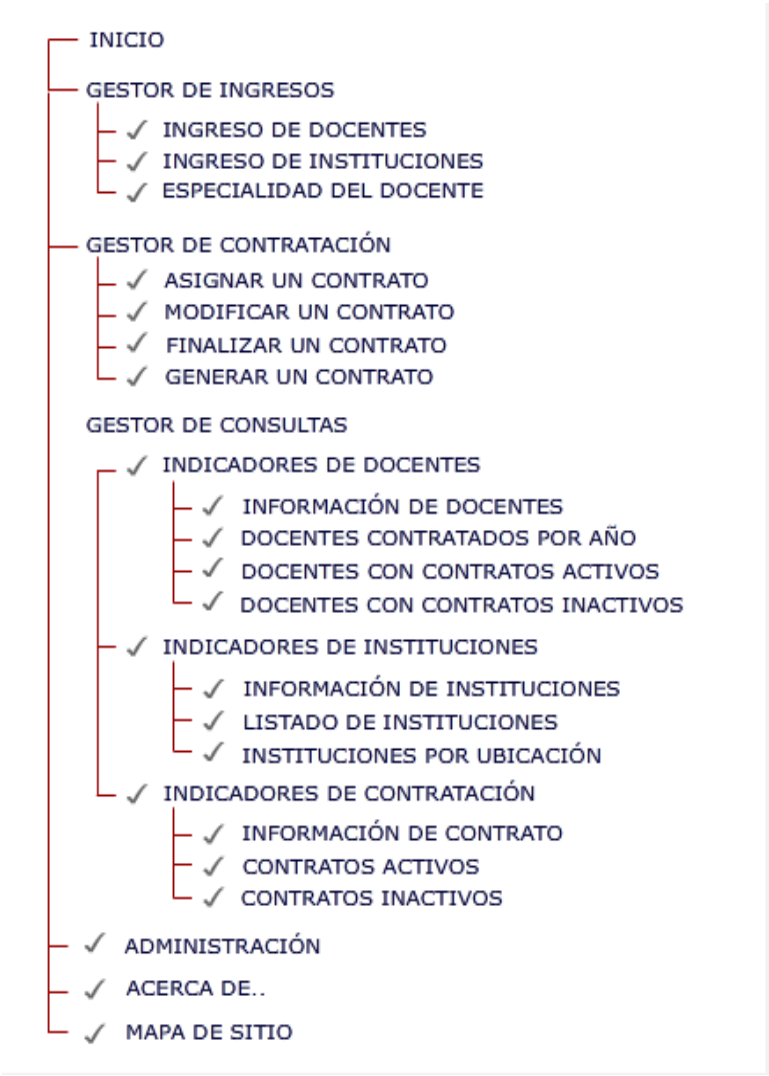


Figura 27: Mapa del sitio

6. Recomendaciones de uso

- ✓ En el caso de que usted no sea un usuario experto se recomienda que lea este manual
- ✓ Así mismo es recomendable que utilice los botones de navegación.
- ✓ Si tiene problemas en el manejo, no funciona correctamente el software diríjase a la página acerca de para que conozca el nombre del autor y pueda buscarlo.