UNIVERSIDAD INTERNACIONAL DE LA RIOJA

**Universidad Internacional de La Rioja (UNIR)**

## Escuela de Ingeniería

**Máster universitario en Dirección e Ingeniería de Sitios Web**

**RAMA PROFESIONAL**

# DESARROLLO DE UNA INFRAESTRUCTURA ORIENTADA A LA WEB PARA GENERACIÓN DE APLICACIONES DE ARRANQUE PERSONALIZADAS EN TECNOLOGÍA RUBYONRAILS

**Trabajo Fin de Máster**

**Presentado por:** Peña Azar, Rafael Eduardo

**Director:** Dr. Cobo Martin, Manuel Jesús

Ciudad: Santa Marta, Colombia.
Fecha: Febrero 8 de 2016.

# Resumen

Hoy en día, los desarrolladores de aplicaciones web que implementan el framework RubyOnRails tienen acceso a múltiples funciones, las cuales les permiten un sinnúmero de configuraciones adaptables al proyecto que estén llevando a cabo. Sin embargo, al culminar un proyecto e iniciar otro, ninguna de éstas funciones pueden ser migradas, deben ser invocadas nuevamente y vueltas a configurar, adaptándolas al nuevo proyecto a elaborar. La infraestructura orientada a la web propuesta contiene múltiples configuraciones que un desarrollador RoR puede integrar a sus aplicaciones, escoger el servidor de desarrollo/test/producción a utilizar, gestores de correo electrónico, frameworks para desarrollo front-end, entre otros. Estas configuraciones pueden ser guardadas y reutilizadas tantas veces como el número de aplicaciones que desee empezar a desarrollar el usuario, con lo cual se cumple el objetivo de personalizar la generación de aplicaciones de arranque y enfocar al usuario en el desarrollo de su proyecto.

**Palabras Clave:** RubyOnRails, Ruby, Rails, Aplicaciones, Arranque, Configuración.

# Abstract

Nowadays, web apps developers that implement the RubyOnRails framework has access to multiple functions, which allow them to a countless configurations adaptable to the project being carried out. However, on completion of a project and start another, none of these functions can be migrated, should be invoked again and re-configured, adapting it to the new project to elaborate. The web-oriented infrastructure proposed contains multiple configurations that RoR developers can integrate to their applications, choosing the development/test/production server to use, email managers, front-end development open-source frameworks, among others. These settings can be saved and reused as many times as the number of applications that the user have to elaborate, whereby the target that is customize the generation of starting applications and focus the users on their projects, is met.

**Keywords:** RubyOnRails, Ruby, Rails, StarterApps, Configuration.

# TABLA DE CONTENIDO

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# CAPITULO 1
# INTRODUCCIÓN

Desde hace algunos años, la web ha sido uno de los medios de información más utilizados para un sinnúmero de objetivos. La manera en que ha ido evolucionando ha sido directamente proporcional a la cantidad de información que ha logrado facilitarle a los millones de usuarios que día a día la han usado. Sitios web hay de muchas clases, siendo el medio donde se propaga la mayor cantidad de información de la mayor variedad posible. Esta evolución constante, va de la mano de muchas personas que se encargan de hacer de la web, tal medio de información, y con la cual acercan cada día a más y más usuarios alrededor del mundo con respecto a alguna temática en específico. Los desarrolladores tienen un papel muy importante en éste ámbito, ya que son quienes hacen posible el uso de éste medio de información masiva, evolucionando junto con la web, conforme ésta así lo requiera.

Específicamente, los desarrolladores RubyOnRails han ido adaptándose a los cambios que la web ha ido presentando con el paso del tiempo, y a razón de los mismos, han ido introduciendo diferentes configuraciones adaptables a dicho framework de desarrollo, para intentar propagar la información que desean, acerca de cualquier temática que se esté tratando, de la mejor manera posible, a través de la web. Hoy en día es muy importante tener un sitio web con diseño adaptativo (responsive), elegante, veloz, que interactúe con el usuario a través de muchas maneras posibles (como por ejemplo, notificarle alguna noticia a través de correo electrónico), entre otras muchas maneras de “atrapar” a los usuarios a dicho sitio.

Hay muchas configuraciones adaptables a frameworks como RubyOnRails, que facilitan el desarrollo y el mantenimiento de aplicaciones web de éste estilo, tales como, entornos de código abierto para desarrollo front-end, motores de autenticación y de bases de datos, tipos

de servidores para alojar dichas aplicaciones, y gemas escritas en Ruby, especiales para las construcciones en Rails.

Debido a la dificultad que puede llegar a presentar cada configuración, y considerando que éstas, no pueden ser migradas hacia otros desarrollos, lo que implica que cada una debe ser reescrita y adaptada a cada nueva aplicación que se desee elaborar, y teniendo en cuenta que cada nuevo desarrollo tiene una configuración general única, lo que hace mas compleja éstas adaptaciones; surgieron algunas soluciones mediante las cuales se pueden llegar a automatizar dichas configuraciones, pero ninguna ofrece la facilidad suficiente para que un desarrollador de aplicaciones en RoR evite los múltiples inconvenientes que éstas configuraciones pueden llegar a presentar, y se centre en la funcionalidad que pretende de éstas, para la solución final del proyecto en el que se encuentra.

En el presente trabajo de fin de Máster, se pretende desarrollar un prototipo de infraestructura web, mediante la cual, desarrolladores de aplicaciones bajo tecnología RubyOnRails, tengan acceso a una gran cantidad de funciones, las cuales sean seleccionadas y puestas en marcha de la manera más sencilla posible, en pro de facilitarles la tarea de configurar el framework y adaptarlo a la aplicación que se desea desarrollar.

## 1.1. PLANTEAMIENTO DEL TRABAJO

Actualmente, una de las metodologías que promueve el desarrollo de aplicaciones en RubyOnRails es el DRY! (Don't Repeat Yourself – ¡No repitas!), la cual llevada a cabo, hace que el proceso de desarrollo como tal se vuelva menos tedioso, ya que no hay que repetir fragmentos de código y se promueve constantemente su reutilización.

Al momento de iniciar un proyecto RoR se deben tener en cuenta muchos aspectos, ya que el comando que produce la nueva aplicación crea un prototipo genérico. En éste punto, se debe hacer un número de configuraciones importantes para el posterior desarrollo a realizar: Que tipo de servidor utilizar para las fases de desarrollo, test y producción (Passenger, Puma, Unicorn, WEBrick, entre otros), que motor de bases de datos utilizar (MySQL, Postgres, SQLite), que framework front-end poner en acción (Bootstrap, Zurb, CSS Simple), que sistema de envío de correo electrónico configurar para dicha aplicación (GMail, Mandrill, SMTP), opciones de autenticación (de llegar a utilizarse - Devise, Sorcery, OmniAuth), utilizar alguna que otra gema importante para la construcción de la aplicación (Savon - para llamar todo tipo de webservices, SimpleForm - para la construcción automatizada de formularios, entre otras), etc.

Existe un sinnúmero de configuraciones, las cuales varían dependiendo del tipo de aplicación que se desea construir, sin embargo existen muchas similitudes, sobretodo en el momento de inicio de los proyectos, en donde se repiten muchas de éstas configuraciones y en la cual no existe un método de reutilización para los desarrolladores, lo que implica que cada vez que una persona va a iniciar un proyecto RoR necesariamente inicia desde cero y con el prototipo genérico. Además, existen múltiples dificultades y a primera instancia se tiende demasiado a cometer errores en la estructuración de las funciones de dichas configuraciones.

Este proyecto intenta atacar, con una infraestructura sencilla en el que el desarrollador personalice toda la configuración inicial que requiere (tipo de servidor, motor de base de

datos, sistema de correo, etc.) e incluso, con cuáles gemas desea contar en su aplicación, desde el inicio, siendo la configuración inicial por excelencia para desarrolladores RubyOnRails que desde el principio de un proyecto apliquen el famoso método DRY, utilizado únicamente con el desarrollo ya puesto en marcha. Se pretende con éste proyecto que cada vez que un desarrollador inicie una nueva aplicación, tenga personalizadas todas las configuraciones que desee, y con las cuales poder aplicar con éstas, otro conjunto de métodos reconocible en el ámbito de la programación, como lo es el CRUD (crear, leer, actualizar y borrar).

## 1.2. ESTRUCTURA DE LA MEMORIA

La presente estructuración de la memoria está compuesta por capítulos que se detallan a continuación:

- **Capítulo 1. Introducción:** En este capítulo se detalla la idea del proyecto, la lectura de la introducción pretende dar una idea muy clara de lo que se quiere elaborar, basada en las motivaciones principales y el planteamiento del trabajo, desglosando los objetivos asociados a la investigación.

- **Capítulo 2. Contexto y estado del arte:** En este capítulo se muestra un resumen específico del conocimiento existente en el campo de trabajo y de las problemáticas abordadas. Aquí se citan otras tecnologías con relación a la planteada en el proyecto actual, además de la materia conceptual del proyecto y las bases teóricas del tipo de metodología de desarrollo de software planteada para la consecución de la resolución de la problemática. Otros estudios elaborados y patentados por otros autores citando referencias relacionadas con el tema escogido.

- **Capítulo 3. Objetivos y metodología de trabajo:** Se describe la meta general y los pasos específicos a ir cumpliendo a medida que el proyecto se realiza, los cuales van ligados a la solución del mismo, y por ende, ligada al título que recibe el proyecto. Los objetivos conllevan a la realización de actividades y tareas y se especifica una metodología asociada al tipo de investigación.

- **Capítulo 4. Aspectos generales del desarrollo de aplicaciones web en RubyOnRails:** Esta es la fundamentación teórica del proyecto. En éste capítulo son descritas todas las configuraciones iniciales existentes en el proyecto y cada una de las opciones en las cuales se dividen. Conlleva una descripción detallada de cada configuración y de su adaptación al desarrollo realizado bajo tecnología RubyOnRails.

- **Capítulo 5. Desarrollo de la contribución:** Se lleva a cabo la identificación de requisitos y todo el estudio previo realizado para guiar el desarrollo del sitio. Además, se hace una descripción general y detallada de la aplicación. Se aportan detalles del proceso de desarrollo y capturas de pantalla que permiten al lector entender el

funcionamiento de la herramienta. Por último, se hace una evaluación del sitio web que alberga la aplicación desarrollada, siguiendo unas métricas y parámetros que determinan el nivel de usabilidad y funcionalidad de la misma.

- **Capítulo 6. Conclusiones y trabajos futuros:** Se describen las conclusiones generadas de la investigación y de cómo ha sido abordado y de porqué dicha solución sería válida, además de la relación de las contribuciones y resultados obtenidos con los objetivos planteados. Además, se incluyen líneas de trabajo futuro, con el fin de aportar (próximamente) valor añadido al trabajo realizado (ninguna aplicación termina de estar desarrollada, siempre se le pueden plantear mejoras que colaboren aún más con la problemática resuelta, o actualizaciones – un caso podría ser, actualizar la versión del Framework utilizado). Se debe justificar de qué modo se puede utilizar el aporte que se ha desarrollado y en qué campos.

# CAPITULO 2
# CONTEXTO Y ESTADO DEL ARTE

Para lograr los objetivos planteados, es necesario un conocimiento, y sobretodo, un entendimiento amplio del contexto asociado a la línea de trabajo seleccionada. Si bien éste proyecto presenta una solución novedosa para realizar las configuraciones iniciales a través de interfaces gráficas, en cuanto a la temática como tal, es complementado por estudios y publicaciones previas de otros autores asociados a la configuración de aplicaciones de arranque en desarrollos basados en RubyOnRails.

## ∅ STARTERKIT

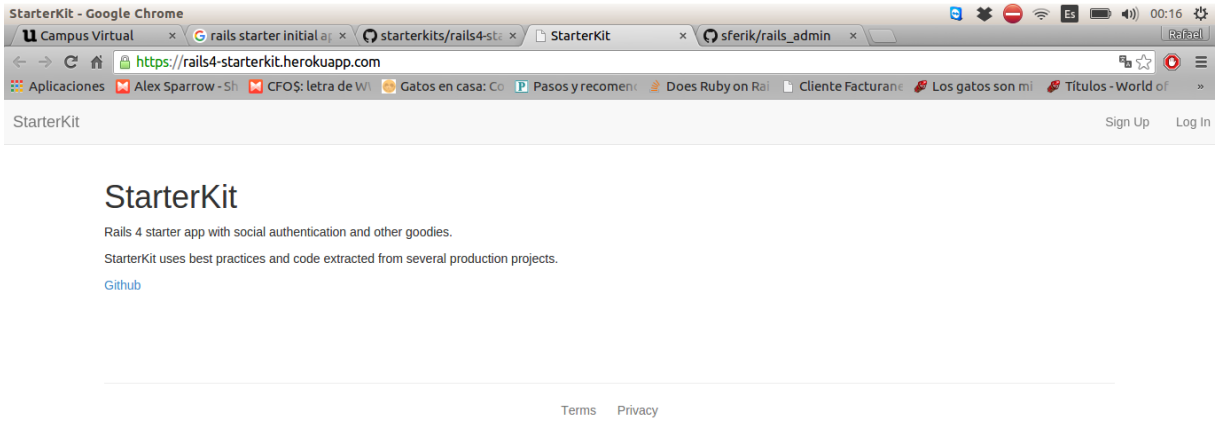


**Figura 1. Página de Presentación del Proyecto StarterKit en Heroku**

StarterKit (Johnston, 2014), es un aplicación de arranque configurada para desarrollar directamente a producción que contiene funciones de búsqueda y selección de gemas en Ruby por medio de una terminal de comandos, para ser aplicadas en el proyecto a desarrollar en Rails. Este proyecto fue desarrollado hace más de 2 años por Joe Johnston,

siendo uno de los pioneros en desarrollo de aplicaciones de arranque predeterminadas para el cambio a la cuarta versión del framework.

Este proyecto maneja una integración con tecnologías como Bootstrap 3, Foundation 5, CoffeeScript, SASS, Devise Authentication, entre otras, para generar plantillas y realizar, por medio de la inclusión de gemas, configuraciones iniciales previas a la primera generación de la aplicación presta a realizar.

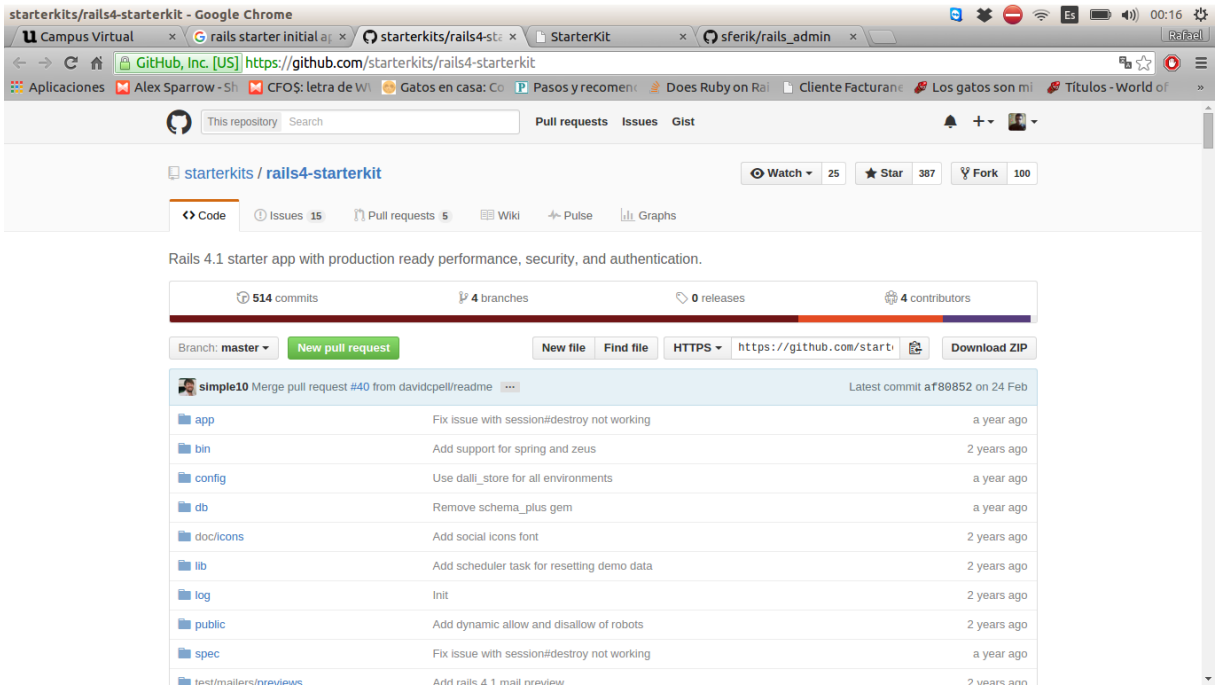


Figura 2. Repositorio del Proyecto StarterKit en GitHub

La manera de aplicar éste proyecto es mediante una consola o terminal de comandos, en la que se clone el repositorio directamente de GitHub (ver Figura 3) y luego aplicar los comandos necesarios para generar las aplicaciones, los cuales se encuentran detallados dentro del manual del repositorio online.

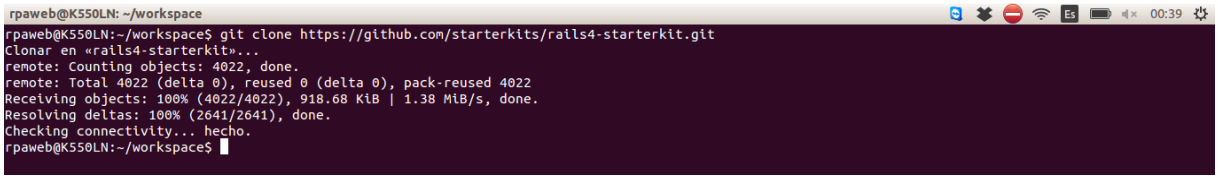


Figura 3. Clonación del Proyecto StarterKit desde GitHub

## ∅ COMPOSER

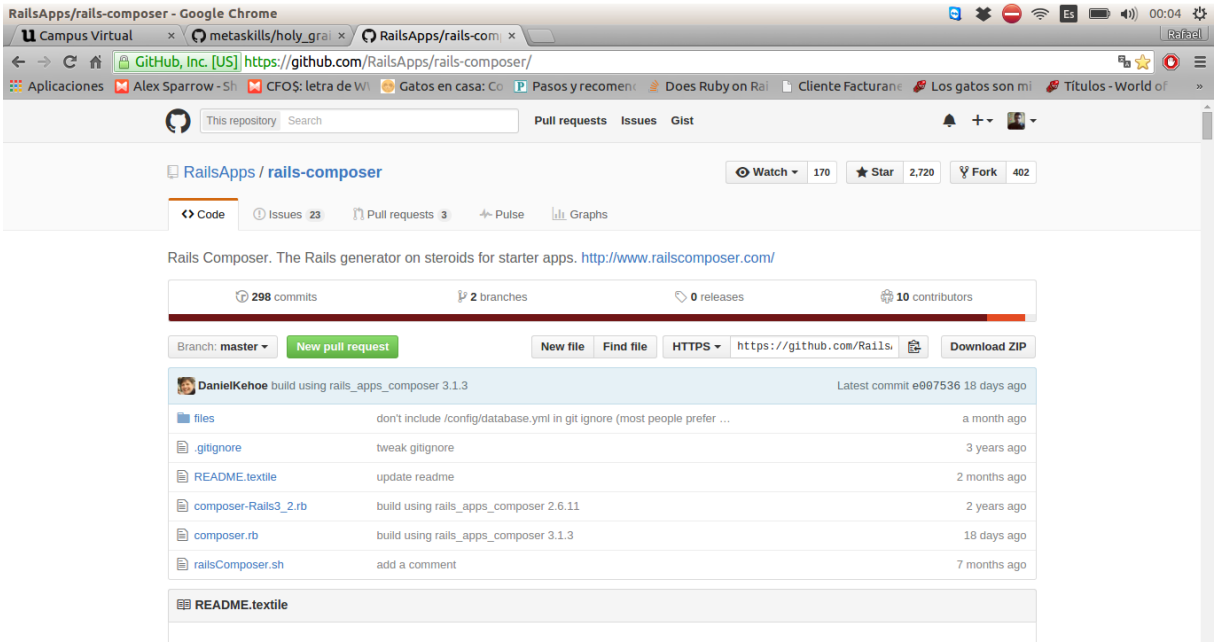


Figura 4. Repositorio del Proyecto Composer en GitHub

Composer (RailsApps, 2012), es una plantilla desarrollada en Ruby para la generación de aplicaciones de arranque en Rails. Llamado a sí mismo “generador con esteroides”, ofrece una gama de configuraciones iniciales para desplegar a partir de éstas, un nuevo proyecto. Este proyecto fue desarrollado por Daniel Kehoe en el año 2012 para las versiones 3.x de RubyOnRails, sin embargo se ha ido actualizando conforme avanza el framework y ya presta soporte para las versiones 4.x.

El método de uso del Composer es a través de una consola o terminal de comandos, y consiste en completar una serie de pasos por medio de los cuales va siendo configurada la nueva aplicación, sin abandonar la terminal.


Figura 5. Generación de una nueva aplicación en RoR a través del Composer

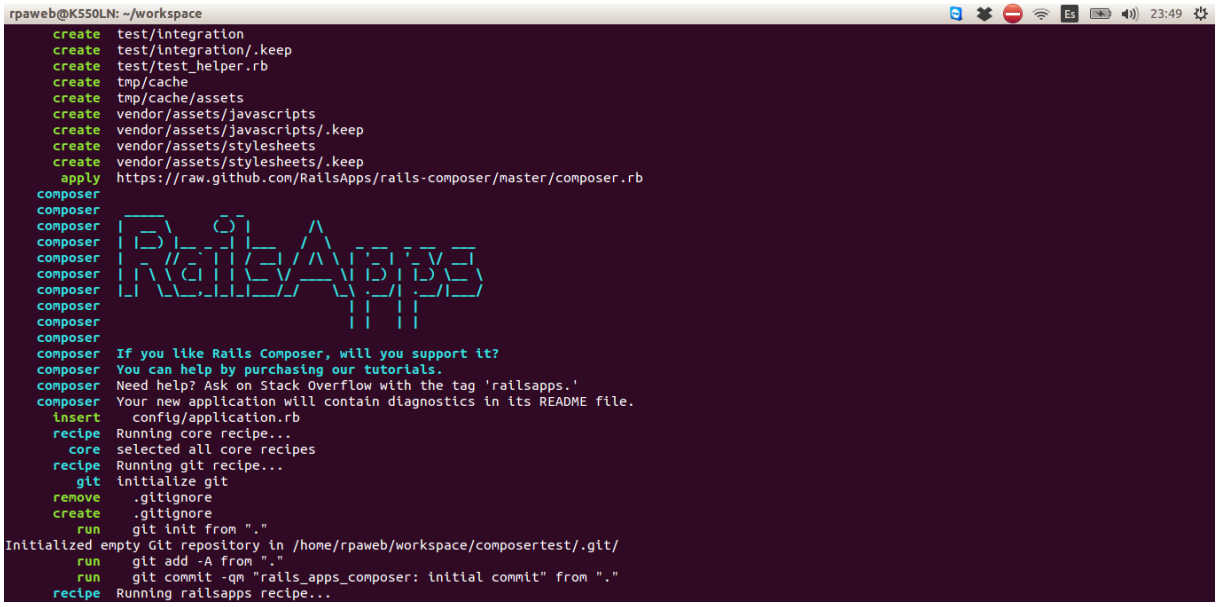


Figura 6. Proceso de generación de la aplicación en base al Composer

Al finalizar la selección de plantillas de configuración (ver Figura 7) para generar la nueva aplicación a través del Composer, se denota la estructuración del archivo de gemas (ver Figura 8) que se encuentran en funcionamiento, las cuales dan paso a las configuraciones iniciales que dicha aplicación posee.

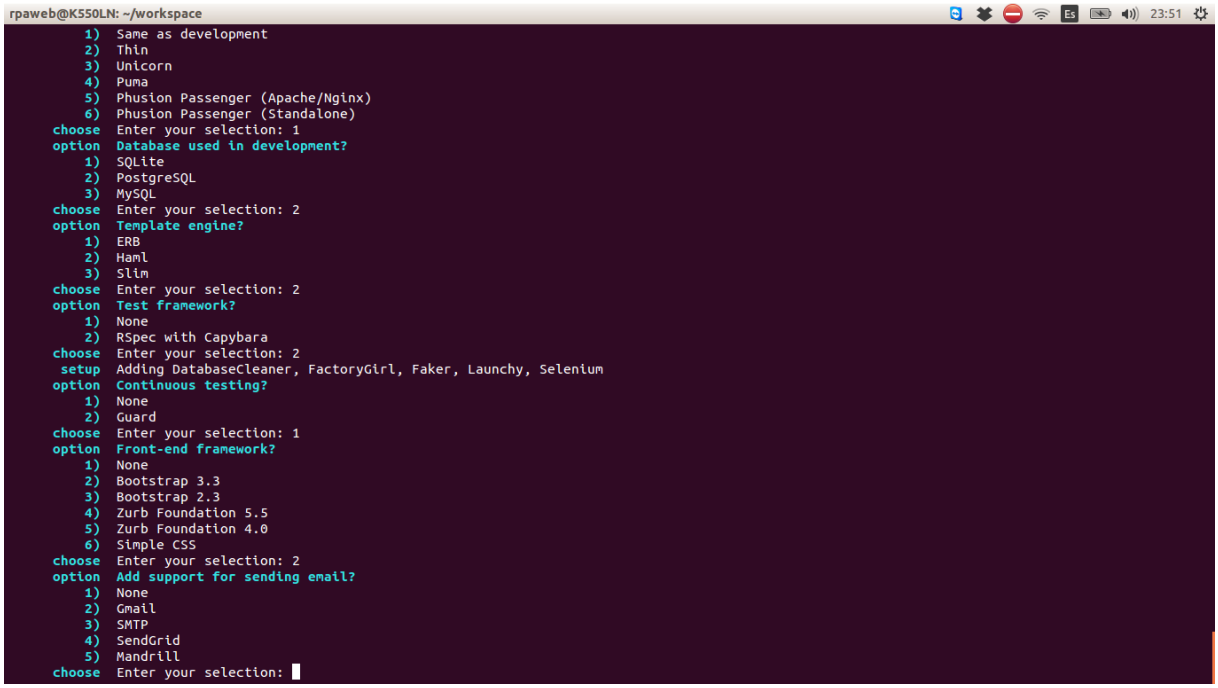


Figura 7. Formato que presenta el Composer para la selección de plantillas

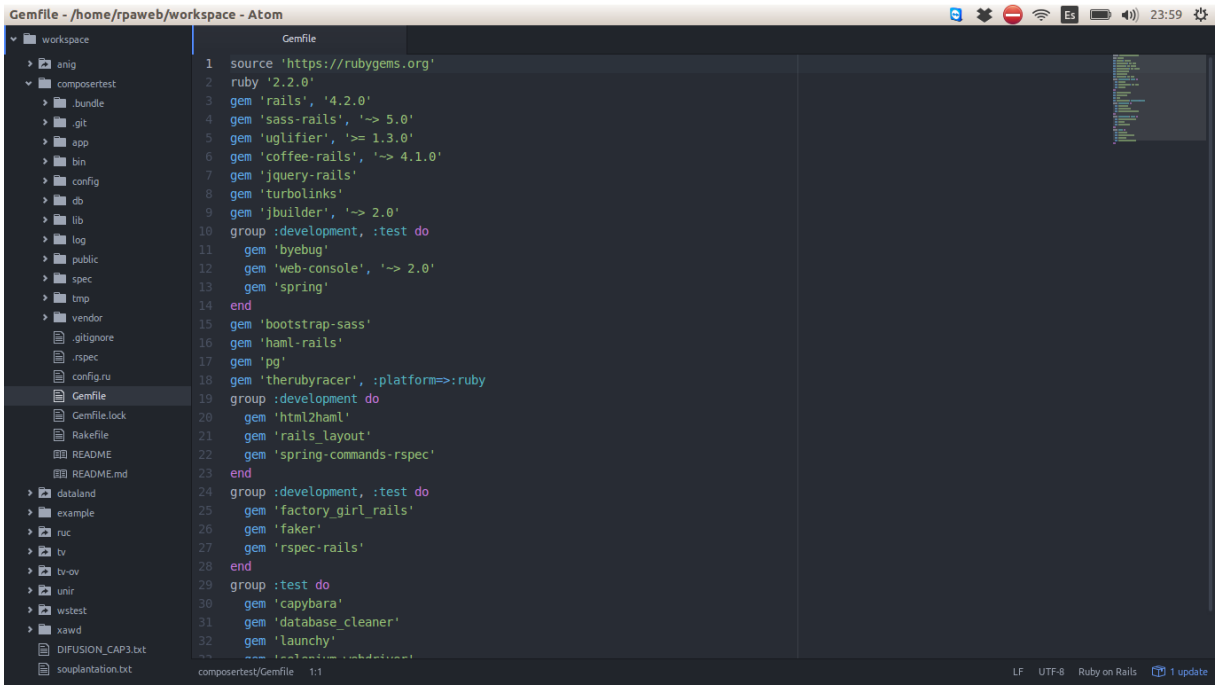


**Figura 8. Estructura del Gemfile luego de la generación de la nueva app**

### ∅ HOLY GRAIL HARNESS

Holy Grail Harness (Collins, 2012), es un prototipo de aplicación conservadora en RubyOnRails, que se centra en los patrones de prueba simples para Ruby y JavaScript. Este proyecto fue creado por Ken Collins en el año 2012.

A diferencia de las plantillas modernas generadoras de aplicaciones como el Composer, Holy Grail Harness es una aplicación básica que puede ser considerada un prototipo y personalizada a través de un simple script de instalación. Además, promueve opciones de prueba sencillas y potentes y se centra en el uso del lenguaje Ruby (versiones 1.9 en adelante), MiniTest::Spec, Capybara (Gema de RubyOnRails), Poltergeist/PhantomJS, Konacha y genera plantillas de Bootstrap (para el diseño web de la app).

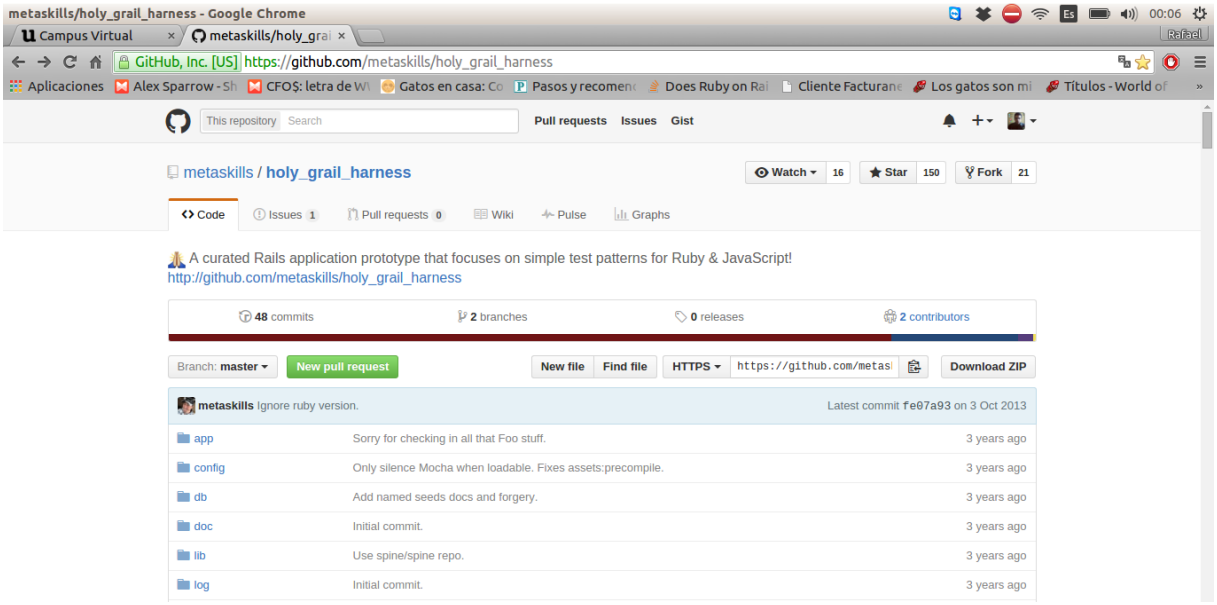


Figura 9. Repositorio del Proyecto HolyGrailHarness en GitHub

Este proyecto si bien ha servido como complemento para otros proyectos de generación de aplicaciones de arranque en Rails, ha sido descontinuado en cuanto a soporte y tiene muchos inconvenientes en el uso para las versiones superiores a las 3.x.

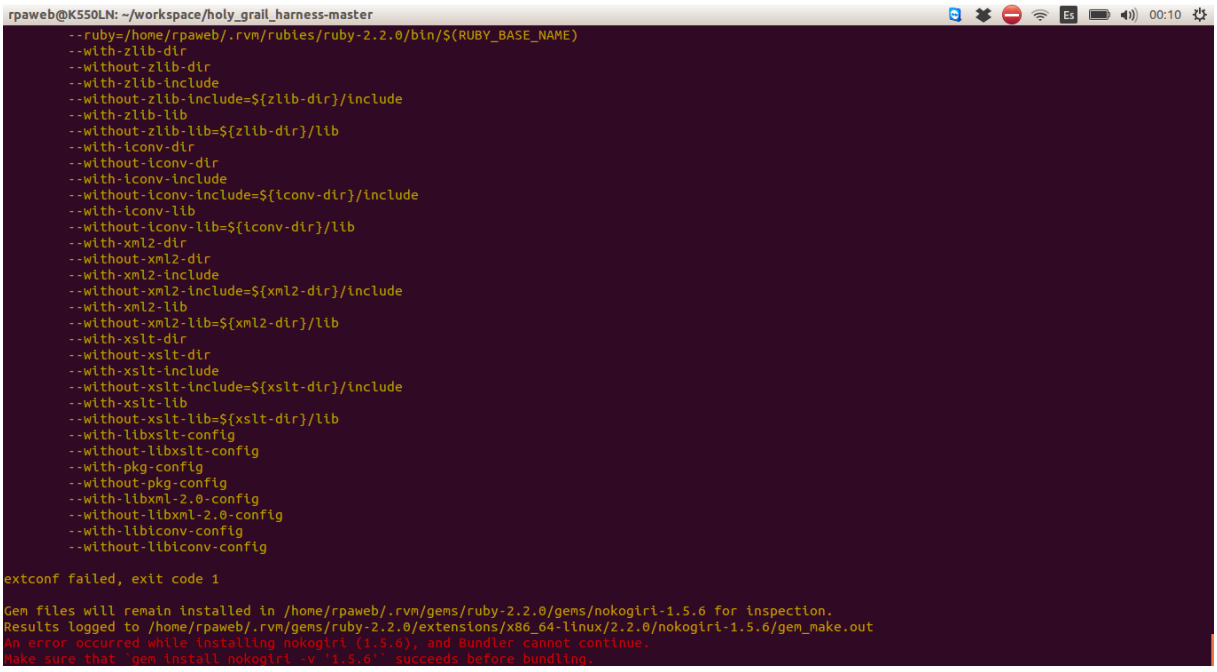


Figura 10. Errores en el despliegue de HolyGrailHarness

- **RAILS WIZARD**

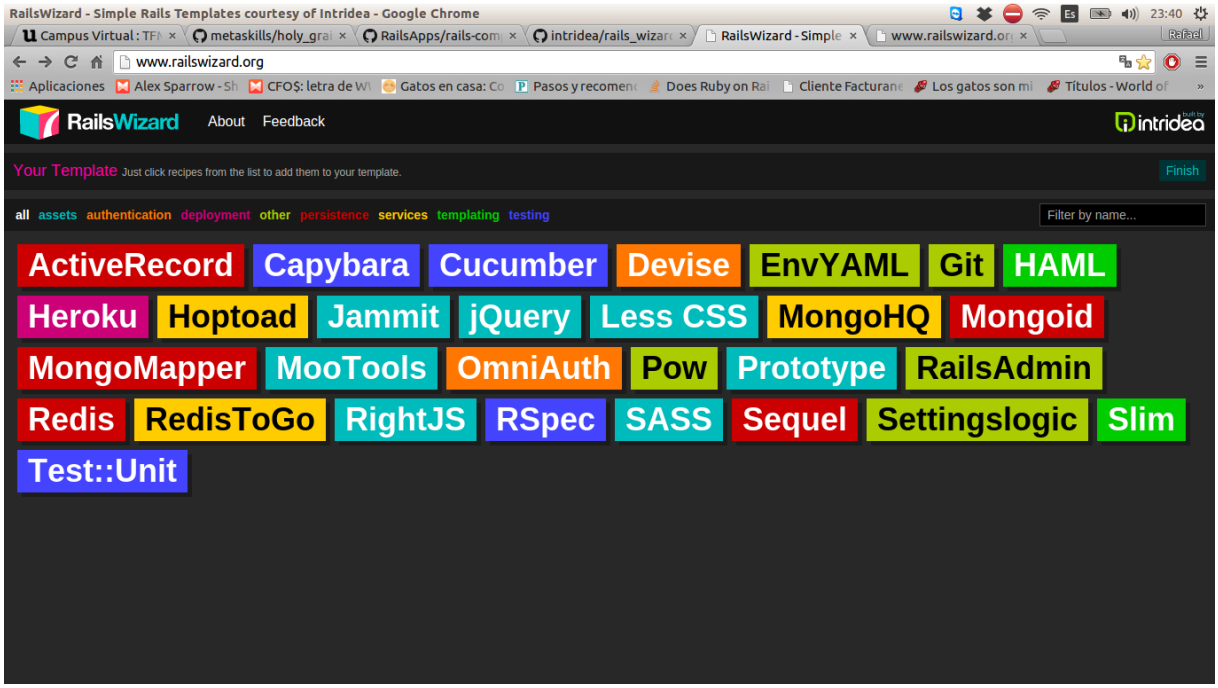


Figura 11. Interfaz del generador RailsWizard

Rails Wizard (Intridea, 2011), es una herramienta generadora de recetas (lista de configuraciones), por medio de la cual los usuarios escogen las gemas de Ruby que requieren para su aplicación. Luego, el generador le provee una línea de comando que debe ser escrita por medio de una consola o terminal para llevar a cabo el proceso de la generación de la nueva aplicación.

La diferencia entre éste proyecto y el propuesto en éste trabajo radica en una mejora sustancial de la interfaz gráfica, la cual en RailsWizard no es modificable (solo se permite escoger las gemas mas no configurarlas). RW genera una lista de configuraciones las cuales son puestas en marcha una vez se introduzca la línea de comandos generada vía terminal.

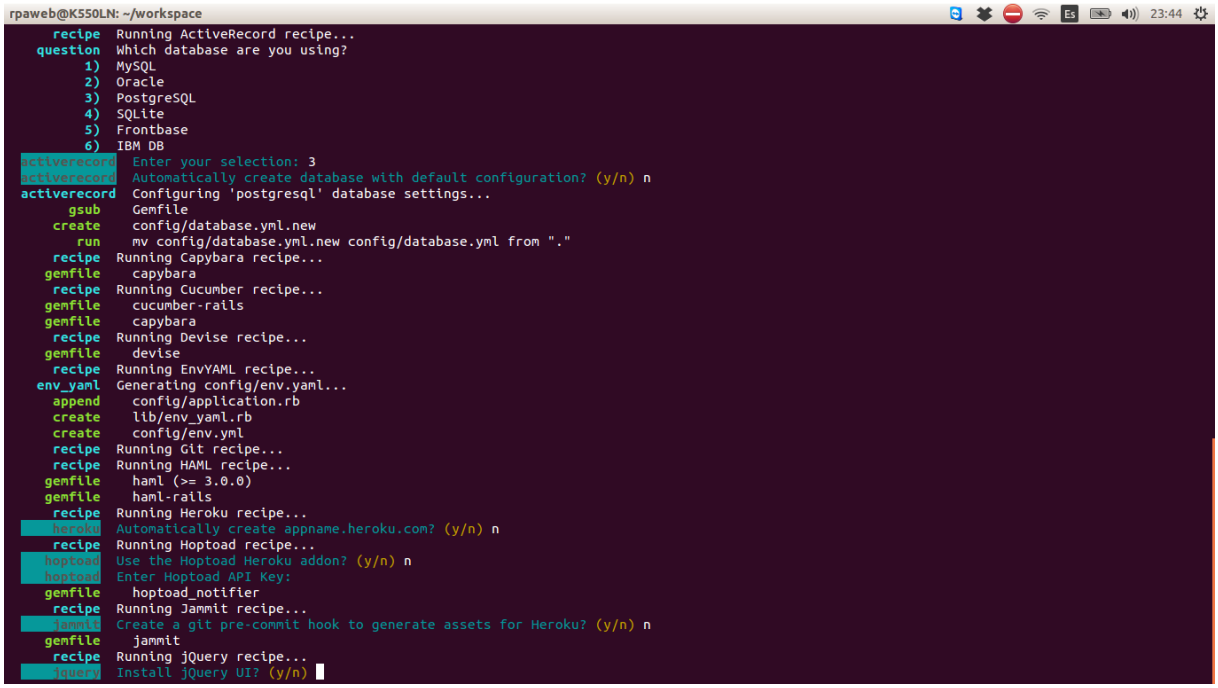


**Figura 12. Generación de la nueva aplicación vía terminal**

Rails Wizard ha servido como complemento para varios generadores de aplicaciones de arranque en RubyOnRails, como por ejemplo, los proyectos Composer y StarterKit (mencionadas anteriormente en éste capítulo), ya que su interfaz provee el mecanismo para la construcción de una lista de recetas (configuraciones – ver Figura 13), las cuales son implementadas a medida que el usuario va creando su aplicación a través de una terminal de comandos.

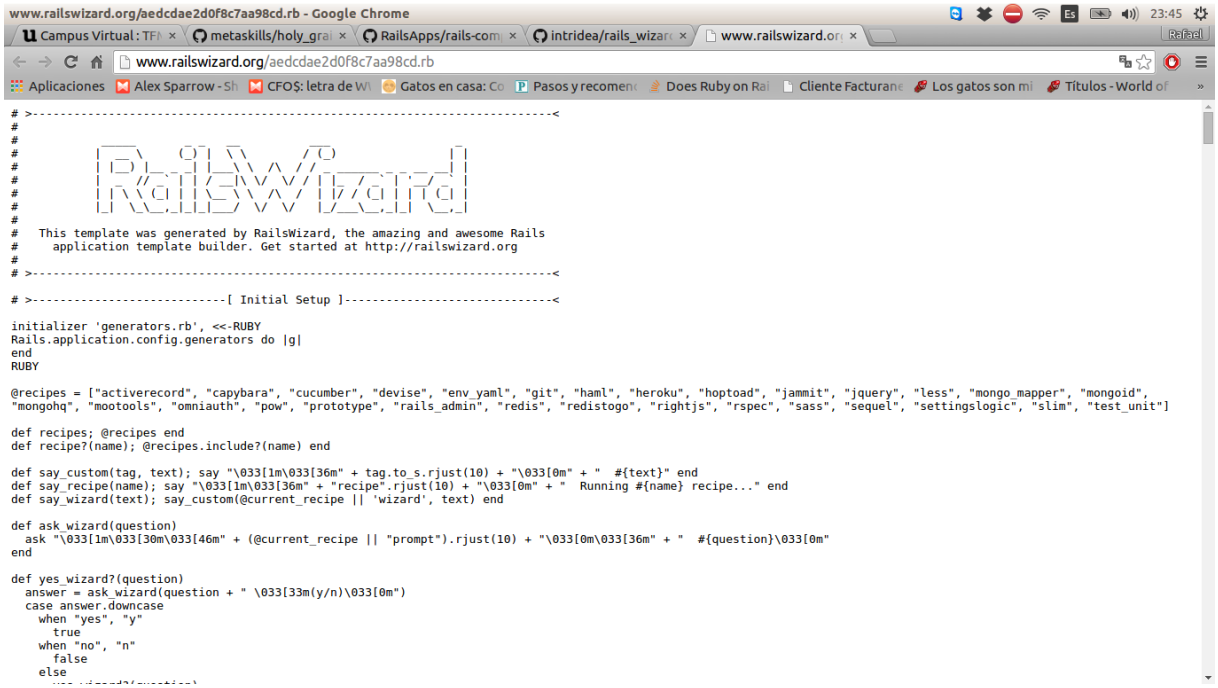


Figura 13. Lista de recetas generada por RailsWizard

Uno de los problemas con los que un desarrollador se puede encontrar mediante el uso de éste generador, radica en la imposibilidad de atacar cualquier error mediante la instalación de las gemas listadas en la receta generada (ver Figura 14), haciendo necesaria una reimplementación de la herramienta (iniciar todo el proceso de nuevo) o bien, abandonando su uso en el caso de no poder corregir nunca el error presentado.

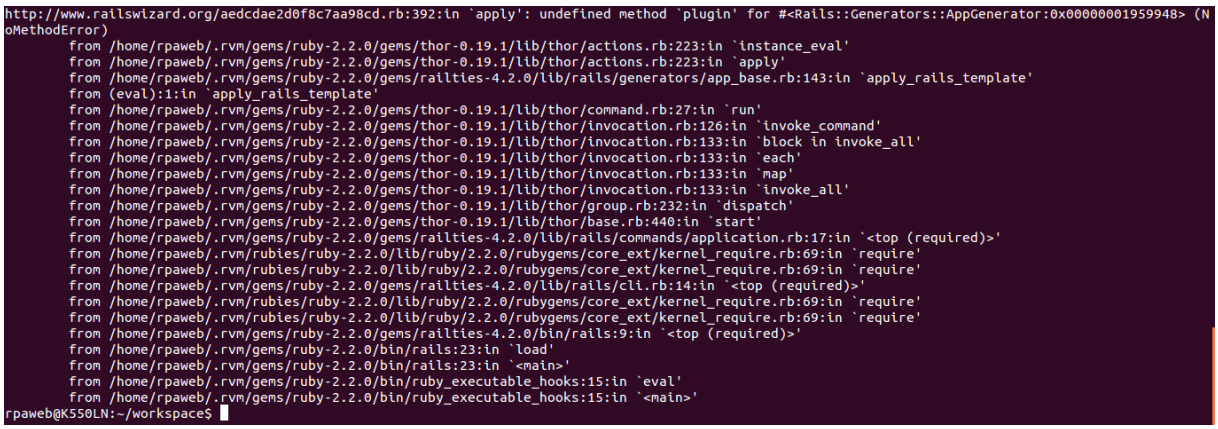


Figura 14. Error común en la generación de aplicaciones con RailsWizard

# CAPITULO 3
# OBJETIVOS Y METODOLOGÍA DE TRABAJO

## 3.1. OBJETIVO GENERAL

Desarrollar una infraestructura para la generación de aplicaciones de arranque personalizadas bajo tecnología RubyOnRails, y que ésta sea 100% orientada a la web.

## 3.2. OBJETIVOS ESPECIFICOS

- ∅ Hacer un estudio exhaustivo de las configuraciones de arranque posibles, tales como selección de servidores de prueba, sistemas de gestión de bases de datos, administradores de tareas periódicas, entre otras.
- ∅ Elaborar los métodos que se implementarán al prototipo para la carga de cada una de las configuraciones.
- ∅ Desarrollar la estructuración de la base de datos de la aplicación, y la construcción de los módulos de Sesiones y el componente de “personalización” de las configuraciones iniciales que el usuario decida guardar.
- ∅ Elaborar la interfaz de usuario, por medio de la cual se hará uso de los métodos a utilizar en el prototipo, de acuerdo a las configuraciones iniciales a las que el usuario decida hacer uso, además cabe resaltar, que deberá ser lo suficientemente simple para hacer al sistema, usable.
- ∅ Hacer las pruebas correspondientes al prototipo funcional.

## 3.3. METODOLOGIA DE TRABAJO

Se aplicará el proceso metodológico de Ingeniería de Software Rational Unified Process, mejor conocido como RUP. La estructura dinámica que presenta ésta metodología, permite un proceso de desarrollo fundamentalmente iterativo en el cual se ven inmersas 4 fases las cuales son: Iniciación, Elaboración, Implementación y Transición.

En la primera fase procedemos a definir el alcance del proyecto, se propone una visión muy general de la arquitectura, se llevan a cabo los estudios pertinentes acerca del tema a tratar, se realiza ingeniería de requisitos y se identifican los riesgos asociados al proyecto.

En la segunda fase se define la arquitectura base del sistema y se presenta una solución preliminar a la problemática planteada. Esta es una fase más avanzada de diseño en la cual se presenta documentación altamente explicativa a nivel de desarrollo, como pueden ser casos de uso, diagramas de clases, modelos de entidad-relación (de ser requeridos), etc. Además, se inicia el desarrollo del proyecto como tal, generando las funcionalidades más importantes.

En la tercera fase se completa la funcionalidad del sistema, se clarifican los requisitos pendientes, se administran cambios de acuerdo a evaluaciones periódicas y se realizan mejoras para el proyecto. Para proyectos RoR en ésta fase suele alternarse tanto Desarrollo Back-End como Front-End, para generar todas las vistas acordes a las funcionalidades ya programadas.

Por último, en la transición, se da disponibilidad del software a usuarios finales y se realizan pruebas de aceptación, se ajustan los errores (bugs) que sean encontrados y al final se verifica que el producto cumpla con las especificaciones iniciales requeridas en el proyecto.

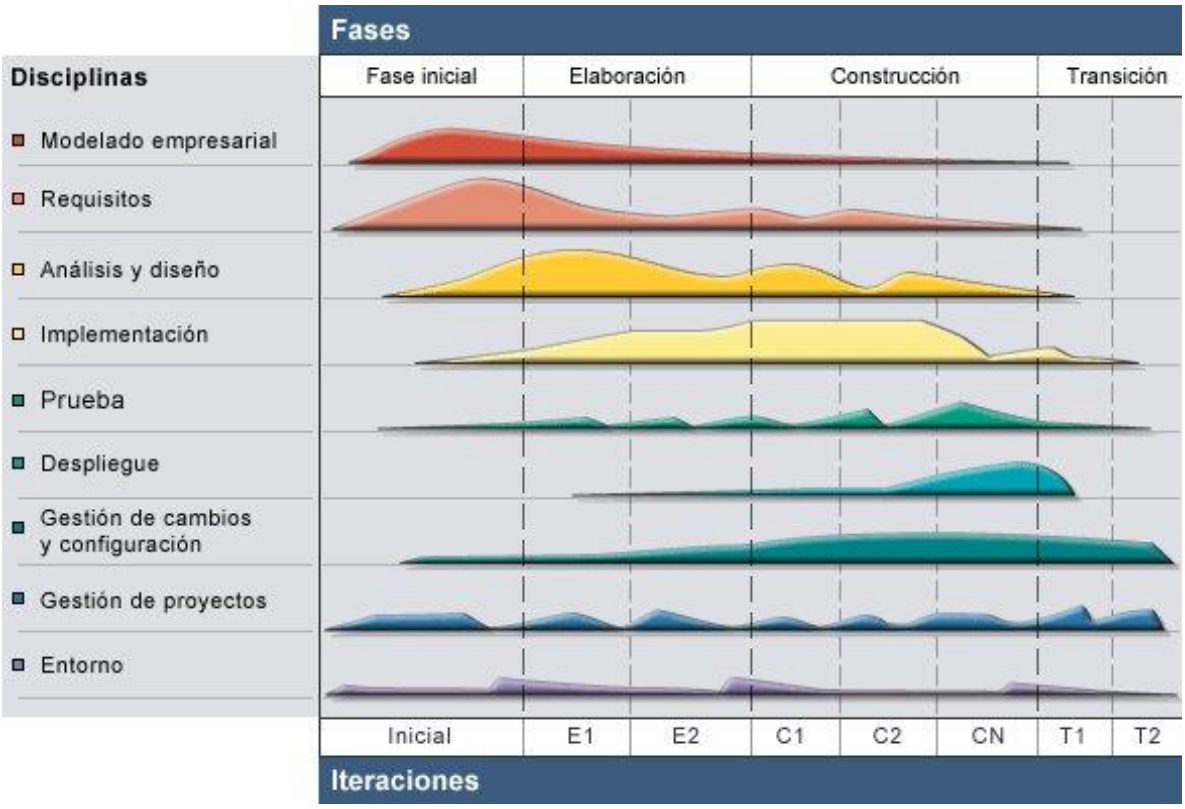


Figura 15. Fases y Disciplinas de la Metodología RUP

# CAPITULO 4
# ASPECTOS GENERALES DEL DESARROLLO DE APLICACIONES WEB EN RUBY ON RAILS

Las aplicaciones web son desarrolladas con una base lógica que es traducida al lenguaje y el entorno en el que se piensa implementar, lo cual le otorga la funcionalidad requerida. Pero hay muchas otras connotaciones que hacen que una aplicación desarrollada, funcione de manera correcta en la web. Se deben tener en cuenta muchos aspectos en el paso-a-paso y en la implementación final de dicha aplicación.

Una de las cosas a tener en cuenta principalmente es el engranaje que se va a utilizar para el desarrollo. Para el caso de éste Trabajo de Fin de Máster, la solución está enfocada en los desarrolladores de aplicaciones web bajo RubyOnRails, el cual conlleva escritura en lenguaje Ruby para establecer la funcionalidad de la aplicación y HTML5, CSS3/SASS y JavaScript/CoffeeScript para el desarrollo de las interfaces de usuario e interacciones.

Pero, ¿Cuáles son las configuraciones necesarias para llevar a cabo el desarrollo de una aplicación web en Ruby y bajo el entorno RubyOnRails?

## 4.1. SERVIDOR WEB

Los servidores web en el desarrollo de aplicaciones web mediante el framework RubyOnRails se usan en diferentes etapas para no involucrar desde el inicio las bases de datos que van a ser adheridas a la aplicación. Son 3 fases en las que está caracterizado el desarrollo de la aplicación y en las que van añadidas a cada una de estas, servidores web: Fase de desarrollo, de pruebas y de producción. En las dos primeras se trabajan de manera

local, y la última vía internet con la aplicación ya funcionando para todos los usuarios que puedan acceder a ella. Los servidores web más usados en el desarrollo de aplicaciones web en RubyOnRails son Apache y Nginx. Además, para fases de desarrollo se suelen usar servidores como Puma, Unicorn y WEBrick (éste último siendo el genérico de RubyOnRails).

### 4.1.1. WEBrick

WEBrick es un conjunto de herramientas de servidor HTTP que puede ser configurado como un servidor HTTPS, un servidor proxy y un servidor virtual - host. WEBrick cuenta con completo acceso a ambas operaciones del servidor, además de acceso HTTP. Además, soporta tanto autenticación básica como digest, además de algoritmos fuera del RFC 2617. Un servidor WEBrick puede estar compuesto de múltiples servidores WEBrick o servlets a proporcionar diferentes comportamientos en función por host o por link. WEBrick incluye servlets para el manejo de scripts CGI, páginas ERB (plantillas para embeber Ruby en HTML), bloques de rubíes y listados de directorios. WEBrick también incluye herramientas para demonizar un proceso e iniciar un proceso con nivel de privilegio superior y/o hacer caer permisos. (Takahashi & Gotou, 2000)

### 4.1.2. Unicorn

Unicorn es un servidor HTTP para aplicaciones de rack diseñado para servir únicamente en clientes rápidos en baja latencia, con buenas conexiones de banda ancha, para aprovechar las características de Unix y su kernel. Clientes menos rápidos deben únicamente ser funcionales mediante la colocación de un proxy inverso capaz de hacer buffering totalmente en ambas, tanto la petición y la respuesta entre clientes Unicorn. (Show, 2007)

### 4.1.3. Puma

Puma es un simple, rápido, hilado, y altamente concurrente servidor HTTP 1.1 para aplicaciones de Ruby/RACK. Se puede utilizar con cualquier aplicación compatible con

RACK, y se considera el reemplazo para WEBrick. Fue diseñado para ser la salida al servidor para Rubinius, pero también funciona bien con JRuby y la MRI. Puma es para uso tanto en entornos de desarrollo y producción. (Phoenix, 2014)

### 4.1.4. Passenger

Phusion Passenger, es un servidor web y de aplicaciones, diseñado para ser rápido, robusto y altamente escalable. Añade potentes funciones de nivel empresarial que son útiles en la producción, y hace de la administración algo mucho más fácil y menos complejo. Passenger soporta Ruby, Python, Node.js y Meteor, y está siendo utilizado por empresas de alto perfil, tales como Apple, Pixar, New York Times, AirBnB, Juniper, etc., así como más de 350.000 sitios web. Lo que lo hace tan rápido y fiable es su núcleo C++, su arquitectura zero-copy, su sistema de vigilancia y su diseño hibrido, multihilo y multiproceso. Soporta Apache y Nginx entre otras y cuenta con una versión Stand-Alone. (B.V., 2012)

## 4.2. GESTOR DE BASES DE DATOS

Algunos gestores o motores de bases de datos destacados son Postgres, MySQL, MongoDB, Redis, entre otros. En Rails, es muy común el uso de alguno de los dos primeros mencionados. Además, es muy usado a nivel de aprendizaje SQLite, el cual es el genérico de RubyOnRails, con el que solo se suelen hacer pruebas cuando el desarrollador es novato y está iniciando en el framework. Las interacciones de Postgres y MySQL con Rails se ejecutan a través de la instalación de las gemas correspondientes, PG para Postgres y Mysql2 para MySQL.

## 4.3. GESTOR DE PLANTILLAS

Conjunto de métodos mediante el cual se logra “embeber” lenguaje Ruby en vistas escritas en HTML. Existen dos tipos muy usados en Rails, los cuales son: ERB y HAML. Éste último, modifica la escritura de las etiquetas HTML, manejándolas de diferente forma, pero con mismo resultado. Además, el navegador, luego de mostrada la vista, transforma el código, mostrando en el “source” código HTML.

### 4.3.1. ERB

Ruby embebido (**E**mbedded **R**u**B**y) es una característica de Ruby que le permite generar convenientemente cualquier tipo de texto, en cualquier cantidad, a partir de plantillas. Las plantillas mismas combinan texto plano con código Ruby para la sustitución de variables y de control de flujo, lo que hace que sean fáciles de escribir y mantener. Aunque ERB frecuentemente es usado para la generación de páginas web, también se utiliza para producir documentos XML, RSS, código fuente, y otras formas de archivo de texto estructurado. Puede ser muy valioso cuando se necesitan crear archivos que incluyen muchas repeticiones de un patrón estándar, tales como bancos de pruebas por unidad. El principal componente de ERB es una biblioteca que se puede llamar dentro de sus aplicaciones Ruby y tareas Rake. Esta biblioteca acepta cualquier cadena como plantilla, y no impone limitaciones a la fuente de la misma. Se puede definir una plantilla totalmente dentro de su código, o almacenarlo en una ubicación externa y la carga como sea necesario. Esto significa que se pueden mantener las plantillas de archivos, bases de datos SQL, o cualquier otro tipo de almacenamiento que se desee utilizar. Distribuciones de Ruby también incluyen una utilidad de línea de comandos que le permite procesar las plantillas que se celebran en los archivos sin necesidad de escribir código adicional. (Ellis, 2015)

```
Hello, <%= @name %>.
Today is <%= Time.now.strftime('%A') %>.
```

**Figura 16. Estructura embebida de Ruby en HTML mediante una plantilla ERB**

### 4.3.2. HAML

HAML es un motor de plantillas para HTML. Está diseñado para que sea más fácil y más agradable para escribir documentos HTML, eliminando la redundancia, lo que refleja la estructura subyacente de que el documento representa, y proporcionando una elegante sintaxis que es a la vez potente y fácil de entender. (Catlin & Weizenbaum, 2006)

```
.row
  .col-lg-12
    %h1.page-header
      = @app.name
      %small
        %kbd= @dbengine
        %kbd.bg-primary= @status
        = @app.desc.humanize
.row
  .col-lg-12
    = render 'layouts/messages'
```

Figura 17. Estructura HAML obtenida del proyecto desarrollado

## 4.4. FRAMEWORK DE DESARROLLO FRONT-END

Los frameworks o entornos de desarrollo a nivel front-end, son un conjunto de herramientas de código abierto que son utilizadas para elaborar el diseño de los sitios y de las aplicaciones web. Al ser de carácter libre el código, poseen plantillas predeterminadas de diseño para todo tipo de etiquetas HTML bajo CSS3, como formularios, iframes, paneles, menús de navegación, etc., y extensiones de JavaScript para establecer la funcionalidad de dichas etiquetas, convirtiendo la web en un entorno ideal para la experiencia del usuario. Correspondiente a RubyOnRails, el framework de desarrollo front-end más utilizado es Bootstrap.

## 4.5. SOPORTE DE CORREO ELECTRONICO

Herramientas que permiten realizar conexiones entre las aplicaciones desarrolladas y los medios para envíos masivos de correo electrónico. Estas herramientas datan atributos específicos para realizar dichas conexiones, para que las mismas aplicaciones sean capaces por medio de llamados de funciones, enviar correos electrónicos masivos de manera automática. Entre algunos ejemplos podemos destacar el SMTP, que es la herramienta básica de envío de correos por parte del motor de RubyOnRails, las herramientas de Google (GMail) para envío masivo de correos y una de las más populares herramientas de conexiones a servidores de correo conocidas como Mandrill.

### 4.5.1. SMTP

Es el protocolo básico de generación de transferencias simples de correo electrónico. Actualmente es un estándar en internet al igual que en el método de desarrollo de transferencia de mensajes vía correos electrónicos masivos en RubyOnRails.

Ilustra (Kehoe, 2012) en su artículo, la manera en la que se configura el ActionMailer bajo la estipulación SMTP básica como método de emisión de correos electrónicos. (Ver Figura 18)

```
# ActionMailer Config
config.action_mailer.default_url_options = { :host => 'localhost:3000' }
config.action_mailer.delivery_method = :smtp
# change to true to allow email to be sent during development
config.action_mailer.perform_deliveries = false
config.action_mailer.raise_delivery_errors = true
config.action_mailer.default :charset => "utf-8"
```

**Figura 18. Estructura de configuración SMTP**

### 4.5.2. GMail

Al igual que SMTP básico, es un estándar para transferencia de correos electrónicos de manera masiva, creada e implementada por Google. En RubyOnRails, es usada para experimentación debido a su nivel de simpleza.

En su artículo (Kehoe, 2012) menciona la manera en la que se configura el ActionMailer con GMail bajo una estructura estándar smtp. (Ver Figura 19)

```
config.action_mailer.smtp_settings = {
  address: "smtp.gmail.com",
  port: 587,
  domain: "example.com",
  authentication: "plain",
  enable_starttls_auto: true,
  user_name: ENV["GMAIL_USERNAME"],
  password: ENV["GMAIL_PASSWORD"]
}
```

Figura 19. Estructura de configuración GMail

### 4.5.3. Mandrill

Mandrill es una API de envío de correos electrónicos masivos transaccionales entre sitios y aplicaciones web, altamente escalable y segura. Ideal para las transacciones de correos basados en datos, comercio electrónico y mensajes personalizados uno-a-uno.

En su artículo (Kehoe, 2012) menciona la manera en la que se configura el ActionMailer con Mandrill bajo una estructura estándar smtp. (Ver Figura 20)

```
config.action_mailer.smtp_settings = {
  :address   => "smtp.mandrillapp.com",
  :port      => 25,
  :user_name => ENV["MANDRILL_USERNAME"],
  :password  => ENV["MANDRILL_API_KEY"]
}
```

Figura 20. Estructura de configuración en Mandrill

## 4.6. AUTENTICACIONES

Las autenticaciones son los medios (funciones) mediante las cuales el software desarrollado lleva control de los usuarios que lo utilizan. Suele ser una parte bastante atacada debido a la seguridad que ésta misma plantea. Son muy pocas las aplicaciones que no conllevan un registro de usuarios y una petición de autenticación para poder acceder a su contenido. En RubyOnRails se trabaja bien sea de forma manual, o de forma automática, invocando gemas (plugins) del Rails para llevar a cabo las funciones. Cada gema tiene sus diferentes funciones y métodos de uso. Sorcery y Devise, son dos de las más conocidas y usadas en la gran mayoría de aplicaciones desarrolladas bajo éste framework.

### 4.6.1. Sorcery

Sorcery es una biblioteca de autenticación escueta y extremadamente robusta, con la cual se pueden escribir los flujos de autenticación necesarios para una aplicación web en RubyOnRails. (Ben-Ari, Shatrov, & Witek, 2010). Esta gema contiene varios módulos de configuración de autenticaciones entre los cuales se encuentran, la activación de usuarios, el reseteo de las contraseñas, la función “recordar”, el “timeout” de cada sesión, la protección de fuerza bruta, autenticaciones HTTP básicas, actividades de “logging”, autenticaciones bajo aplicaciones externas (redes sociales), entre otras.

### 4.6.2. Devise

Devise es una solución flexible para la realización de autenticaciones en RubyOnRails basada en Warden (framework de autenticación generalizada en base a Rack creado por Daniel Neighman) (Plataformatec, 2009). Contiene varios módulos de configuración en base a autenticaciones para las aplicaciones RubyOnRails, entre los cuales se encuentran, autenticación salvada en base de datos, autenticación propia, confirmación de autenticación, recuperación de cuentas de usuario, registros, la función “recordar”, seguimiento de autenticaciones, funciones “timeout”, validaciones y bloqueos de cuentas de usuario, entre otras.

## 4.7. ASIGNACION DE ROLES

Función mediante la cual se establecen los roles que va a manejar la aplicación desarrollada y en la cual le es asignado cada uno de éstos roles a cada uno de los usuarios registrados en dicha aplicación. Esto con el fin de establecer los tipos de autorizaciones que van a tener cada uno de los usuarios con respecto al rol que les es preestablecido. Rolify, es la gema que permite configurar la determinación de los roles y sus respectivas asignaciones. Se integra con módulos de autorizaciones para completar un medio de seguridad en el desarrollo.

## 4.8. AUTORIZACIONES

Según los tipos de roles que son determinados para su respectivo uso en cada uno de los usuarios registrados en la aplicación desarrollada, se establecen permisos para accesos a vistas y funciones de dicha aplicación a cada usuario dependiendo de su rol en el sistema. Estas autorizaciones son otro filtro más de seguridad en el desarrollo, para establecer las restricciones necesarias en el sistema. Existen gemas como Pundit o CanCanCan, que

permiten configurar las autorizaciones en la aplicación, siendo el complemento ideal de la gema Rolify en el establecimiento de los roles.

### 4.8.1. CanCanCan

CanCanCan es una continuación de la gema de control de autorizaciones CanCan para las versiones 3.x de Rails. CanCanCan, es una biblioteca de autorización para RubyOnRails que limita los recursos que se le permiten a un usuario. Todos los permisos se definen en un solo lugar (una clase denominada Ability) y no duplicado entre controladores, vistas y consultas de bases de datos. (Bates, Rite, Shelley, & Ermolovich, 2011)

```
class Ability
  include CanCan::Ability

  def initialize(user)
  end
end
```

Figura 21. Estructura inicial del Ability

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

Figura 22. CanCanCan embebido

### 4.8.2. Pundit

Pundit proporciona un conjunto de asistentes que guían en el aprovechamiento de las clases regulares de Ruby y en los patrones de diseño orientados a objetos, para construir un sistema de autorización simple, robusto y escalable. (Nicklas, 2012)

## 4.9. CONSTRUCTOR DE FORMULARIOS

Uno de los ítems más importantes en la generación automática de formularios es el constructor. En RubyOnRails suelen ser muy usados debido a la facilidad que prestan al desarrollador en vincular los formularios y sus respectivos permalinks (configurados en el resource) con los modelos y sus respectivos salvamentos de datos, sin violar la seguridad de la aplicación. El más importante hoy en día es el que presta la gema SimpleForm, la cual configura de manera muy flexible y con fuerte composición, la creación de formularios.

## 4.10. API DE CONSUMO DE SERVICIOS WEB

En temas de seguridad un tópico muy importante es el de mantener la integridad de los datos por encima de la aplicación que los manipula, y en muchas ocasiones, la solución radica en consumir servicios web en el que la petición se haga desde la aplicación desarrollada, y el servicio responda con los datos estrictamente necesarios (la aplicación nunca manejaría todos los datos y solo se ven involucrados estrictamente los requeridos). En RubyOnRails, se manejan 3 tipos de Servicios Web: SOAP, WSDL y REST. Gemas como HTTParty o SAVON, consumen los servicios web y dan las respuestas a la aplicación en JSON. Ésta última puede consumir sin inconvenientes los 3 tipos de servicios web, incluso a la vez, asignando una variable a cada servicio consumido.

### 4.10.1. HTTParty

HTTParty es una gema de Ruby que se centra en el consumo de servicios web y APIs. Cuenta con una conexión DSL muy legible y con un soporte out-of-the-box (funciona inmediatamente después de instalado, no necesita configuración ni modificaciones previas para poder hacer uso de sus funciones) para JSON, HTML y XML. (Nunemaker, 2008)

### 4.10.2. Savon

Savon es un cliente SOAP muy robusto, desarrollado para RubyOnRails en el cual pueden ser consumidos X cantidad de servicios web de diferentes tipos. Realiza conexiones rápidas y posee cortos tiempos de respuesta. La estructura tanto de emisión como de recepción es JSON, aunque al tratarse de Ruby, el lenguaje en el que se desenvuelve, puede realizar las peticiones y las recepciones a modo de HASH. (Harrington, 2010)

## 4.11. GESTOR DE VARIABLES DE ENTORNO

Muchas aplicaciones requieren una configuración de ajustes tales como credenciales de la cuenta de correo electrónico o claves API para servicios externos. Es posible pasar parámetros de configuración a una aplicación utilizando variables de entorno. Los sistemas operativos (Linux, MacOSX, Windows) proporcionan mecanismos para establecer las variables de entorno locales, al igual que sistemas en la nube como Heroku y otras plataformas de despliegue. RubyOnRails cuenta con sistemas para configurar estas variables de entorno y de esta manera, proteger las credenciales, las cuales sea necesario incluir para determinados servicios (como por ejemplo, la conexión a una base de datos)

### 4.11.1. Figaro

Las configuraciones a menudo incluyen información sensible. Figaro, por medio de un archivo YAML en la aplicación, el cual es ignorado en todo tipo de “commits” (subidas y actualizaciones de repositorios en la nube), y carga sus valores ENV (variables de entorno guardadas siempre a nivel local). De esta manera, se protegen las credenciales de una manera sencilla y sin poner en riesgo la seguridad de la aplicación en ningún momento. Figaro está basada en la aplicación TwelveFactor, la cual configura variables de entorno en sistemas en la nube como Heroku. (Richert, 2012)

## 4.12. GENERADOR DE PDF

Este es uno de los ítems más opcionales que posee una configuración inicial de aplicaciones en Rails, pero no deja de ser importante. En muchas aplicaciones se suelen hacer generaciones de reportes que conllevan a la impresión de un PDF con los datos requeridos. Existen varias gemas para Rails que construyen éstos PDFs a partir de parámetros y algunas variables.

### 4.12.1. Prawn

Prawn es una biblioteca de generación de PDF pura escrita en Ruby que proporciona una gran cantidad de funcionalidades, permaneciendo simple y razonable. A diferencia de muchos generadores por variable, ésta se centra en el diseño del PDF a partir de bloques. (Brown, 2014)

```ruby
require "prawn"

Prawn::Document.generate("hello.pdf") do
  text "Hello World!"
end
```

Figura 23. Configuración inicial de PrawnPDF

### 4.12.2. Wicked PDF

Wicked PDF utiliza la función de shell, **wkhtmltopdf**, para generarle a un archivo PDF a un usuario a partir de HTML. En lugar de establecer por medio de una conexión DSL o de algún otro tipo, la generación de un PDF, el usuario sólo debe escribir una vista HTML de manera común y corriente, y luego Wicked PDF se encarga de convertir toda la vista realizada en un documento PDF. (Sterrett, 2011) Rails genera unos formatos de respuesta en los controladores donde son requeridos; generalmente son HTML y JSON. Wicked PDF genera un tercer formato de respuesta (PDF) el cual renderiza el contenido de la vista en HTML en un nuevo archivo de extensión PDF.

# CAPITULO 5
# DESARROLLO DE LA CONTRIBUCIÓN

## 5.1. PREÁMBULO

El objetivo del presente capítulo es realizar una inducción al proyecto como tal, y describir el proceso mediante el cual fue llevado a cabo el desarrollo de la contribución realizada, que data de una infraestructura orientada a la web para optimizar el proceso de creación de aplicaciones con configuraciones iniciales personalizadas en tecnología RubyOnRails.

### 5.1.1. Aspectos Generales de la Tecnología Utilizada

RubyOnRails (conocido también como Rails o RoR) es un entorno de desarrollo de aplicaciones web, creado por David Heinemeier Hansson en 2004, de código abierto, escrito y adaptado totalmente al lenguaje Ruby, el cual sigue el paradigma de la arquitectura Modelo–Vista–Controlador.

Según Puente (2014) se define a Rails como un framework web full-stack optimizado para la felicidad y productividad sostenible a la hora de programar, que alienta la escritura de buen código usando la **convención contra la configuración**.

Está diseñado para hacer que las aplicaciones web sean más sencillas de desarrollar, construyendo supuestos acerca de lo que necesita cada desarrollador para empezar (denominado receta inicial o predeterminada). Permite escribir mucho menos código a diferencia de muchos otros lenguajes y entornos de desarrollo (frameworks). (RailsCoreTeam, 2013)

#### 5.1.1.1. Generalidades

Ruby como lenguaje permite la metaprogramación, de la que el framework RubyOnRails hace uso, resultando en una sintaxis que muchos usuarios/desarrolladores encuentran muy legible. Además, el framework se distribuye a través de RubyGems, el formato oficial de distribución de bibliotecas y aplicaciones Ruby para embeber en las aplicaciones Rails.

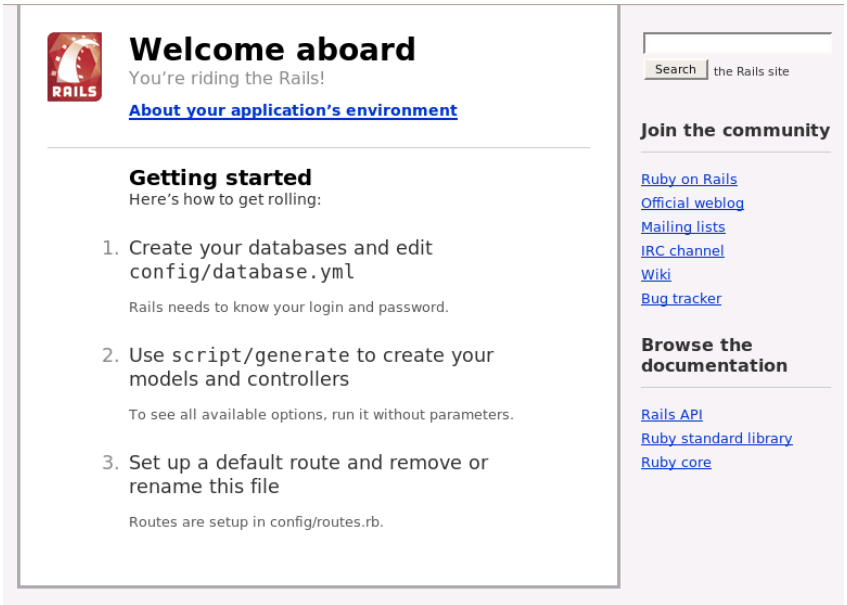


Figura 24. Plantilla inicial de una aplicación Rails

Según (RailsCoreTeam, 2013) la filosofía Rails incluye 2 grandes principios:

- Don't Repeat Yourself (DRY – ¡No repitas!): es un principio de desarrollo de software que establece que "cada pieza de conocimiento debe tener una representación única, autorizada y sin ambigüedades dentro de un sistema". No escribiendo la misma información una y otra vez, el código es más fácil de mantener, más extensible, y con menos errores.
- Convención sobre la Configuración: La comunidad Rails tiene diversas opiniones sobre la mejor manera de hacer muchas cosas en una aplicación web. "Es preferible establecer por defecto un conjunto de convenciones, en vez de que especificar cada uno de los pasos que se dan a través de archivos de configuración demasiado

largos". Esto significa que el programador sólo necesita definir aquella configuración que no es convencional.

#### 5.1.1.2. Modelo–Vista–Controlador

De acuerdo con (RailsCoreTeam, 2013), la arquitectura MVC en RubyOnRails está estructurada de la siguiente manera:

- ∅ Modelo: Son las clases que representan a cada una de las tablas de la base de datos fusionada con la aplicación. Estas clases son manejadas por ActiveRecord (una implementación del patrón denominado con el mismo nombre el cual describe el sistema de mapeo de objetos relacionales). El modelo representa las tablas de la base de datos, las migraciones (cambios en las bases de datos), observadores y emigraciones.
- ∅ Vista: Es la manera en la que se muestran los datos que provee el controlador. Llamada la lógica de la visualización. Haciendo énfasis en la filosofía DRY!, las vistas suelen ser compuestas por muy poco código, bien sea HTML, o con Ruby embebido dependiendo del tipo de gestor de plantillas utilizado (ERB, HAML, etc.)
- ∅ Controlador: Establecen la funcionalidad de la aplicación como tal. Responde a la interacción de los usuarios y proveen de datos a las vistas para su posterior muestra. A su vez, manipula los datos que le son provistos por los modelos. En Rails, es gestionado por ActionPack que es quien genera la clase padre "ApplicationController". Todos los controladores de la aplicación heredan de esta clase y cada acción la definen como un método.

#### 5.1.1.3. ActiveRecord

Según Fowler (2003) en Active Record los objetos llevan tanto los datos como la conducta que opera en esos datos persistentes. Active Record garantiza la lógica de acceso a datos como parte del objeto e instruye a los usuarios de ese objeto en la forma de escribir y leer directamente de la base de datos.

#### 5.1.1.4. Gemas

Según Puente (2014) una gema es, básicamente, la manera en que Ruby permite distribuir programas, módulos o librerías que extienden funcionalidad, casi siempre específica, y que aplican la filosofía de Rails, la cual es **no tener que repetirnos** (*Don't Repeat Yourself*) volviendo más fácil nuestro flujo de desarrollo. Existen gemas para Rails que se encargan de la autenticación de usuarios como es el caso de Devise, hasta gemas como Carrierwave para realizar "uploads" de archivos. Para casi cada acción que se realiza repetitivamente a la hora de programar, existe una gema que lo vuelve todo más fácil. La gema más famosa de Ruby es precisamente el framework de desarrollo full-stack **RubyOnRails**.

```
source 'https://rubygems.org'


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.0'
# Use mysql as the database for Active Record
gem 'mysql2'
# Using HAML instead of ERB
gem "haml-rails", "~> 0.9"
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.1.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtimes
gem 'therubyracer', platforms: :ruby
# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more: https:/
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

```

**Figura 25. Estructura de un Gemfile en Rails**

#### 5.1.1.5. Entornos de Trabajo adaptados a Rails

Existen muchos tipos de entornos de trabajo para adaptar a RubyOnRails para efectuar los desarrollos requeridos en ésta tecnología. Algunos de los más utilizados son los siguientes:

- Atom
- Sublime Text
- Aptana Studio
- TextMate
- GMate
- NetBeans

### 5.1.2. Ingeniería de Requisitos

Para el desarrollo de la infraestructura se efectuó un análisis por medio del cual fueron determinadas las diferentes opciones de configuración a realizar por parte de los usuarios/desarrolladores.

Existen 4 tipos de configuraciones que son obligatorias al momento de construir una aplicación en RubyOnRails.

- ∅ Servidor en Fase de Desarrollo
- ∅ Servidor en Fase de Producción
- ∅ Gestor de Base de Datos
- ∅ Gestor de Plantillas

El carácter obligatorio lo presentan debido a que ninguna aplicación puede funcionar bajo ningún entorno (tanto desarrollo, como pruebas y de producción), por lo cual es fundamental realizar su configuración y adaptación previa. Un ejemplo de ello lo muestra el tipo de configuración llevada a cabo para la conexión a una base de datos. (Ver Figura 24). En ella se puede ver que dependiendo del tipo de motor (adapter) la configuración tiende a variar, y cada una de las fases (en el caso de la imagen, de desarrollo) depende del tipo de configuración por defecto incluida (el adaptador o motor, suele ir en la configuración por defecto).

```
# MySQL. Version-5.5!
# Database Configurations.
#
default: &default
  adapter: mysql2
  encoding: utf8
  pool: 5
  host: 127.0.0.1
  port: 3306
  username: root
  password: cajamag
  socket: /var/lib/mysql/mysql.sock
#
development:
  <<: *default
```

Figura 26. Parte de un archivo de configuración de una BD

En el caso del gestor de plantillas, se sabe que ERB es añadida por Rails de manera automática, sin embargo no deja de ser de carácter obligatorio, ya que el usuario/desarrollador debe decidir el tipo de plantilla inicial.

Otro caso es el de la configuración del servidor. Es necesario incluirlo simplemente porque sin un servidor la aplicación no podría funcionar. No podría ponerse en marcha. El servidor en fase de desarrollo es necesario para ir comprobando que lo que se está escribiendo en código, está siendo visualizado en el navegador, verificando que no haya ningún tipo de errores. Mientras que el servidor de producción, es necesario para alojar la aplicación ya terminada en el host, y ponerla en marcha para todos los usuarios que deseen (o tengan acceso) a utilizarla.

En este ítem también se establece una diferencia clara entre una aplicación en fase de desarrollo y de producción, ya que el tipo de configuración también tiende a cambiar dependiendo del servidor estipulado. (Ver Figuras 27 y 28)

```
NameVirtualHost zebes:80

<VirtualHost zebes:80>
  ServerAdmin rafael.pena@cajamag.com.co
  DocumentRoot /var/www/html
  ServerName zebes
  <Directory /var/www/html>
    AllowOverride all
    Options -Multiviews
  </Directory>
</VirtualHost>

<VirtualHost zebes:80>
  ServerAdmin rafael.pena@cajamag.com.co
  DocumentRoot /var/www/apps/oculus/public
  ServerName oculus.cajamag.com.co
  RackEnv development
  ErrorLog /var/log/httpd/oculus/error.log
  CustomLog /var/log/httpd/oculus/access.log common
  <Directory /var/www/apps/oculus/public>
    AllowOverride all
    Options -Multiviews
  </Directory>
</VirtualHost>
```

Figura 27. Configuración de Virtual Hosts de un Servidor

```
LoadModule passenger_module /usr/local/rvm/gems/ruby-2.2.0/gems/passenger-5.0.2/buildout/apache2/mod_passenger.so
PassengerRoot /usr/local/rvm/gems/ruby-2.2.0/gems/passenger-5.0.2
PassengerRuby /usr/local/rvm/gems/ruby-2.2.0/wrappers/ruby
```

Figura 28. Método de Estipulación del Servidor Passenger

Existen otros tipos de configuraciones posibles a realizar para el desarrollo de una aplicación web en Rails, las cuales tienen el carácter de “opcionales”, ya que no son requeridas por el núcleo para que dicha aplicación pueda tener funcionamiento. Sin embargo, el uso masivo de estas opciones a nivel general, y su uso repetido en muchas aplicaciones, además de la ideología que posee la creación de ésta infraestructura, la cual es “no repetir”, conlleva a su inclusión dentro de este proyecto, como opciones para aquellos usuarios/desarrolladores que deseen ponerlas en práctica en cada una de sus elaboraciones. Estas son:

- ∅ Framework para Desarrollo Front-End de Código Abierto
- ∅ Soporte para el envío masivo de Correos Electrónicos
- ∅ Gestión de Autenticaciones
- ∅ Asignación de Roles por Usuario
- ∅ Gestión de Autorizaciones (factor dependiente de la asignación de roles)
- ∅ Constructores de Formularios

- ∅ APIs de Consumo de Servicios Web
- ∅ Gestor de Variables de Entorno
- ∅ Generadores de PDF

### 5.1.3. Marco Legal

Para hablar de marco legal hay que mencionar que este proporciona las bases sobre las cuales las creaciones determinan el alcance y naturaleza de estas. En el marco legal regularmente se encuentran en un buen número de provisiones regulatorias y normas interrelacionadas entre sí con el fin de colocar límites legales a estas instituciones.

En cuanto al marco legal, tenemos que referirnos a todos y cada uno de los puntos los cuales traten sobre la forma de implementar cada una de las configuraciones posibles mencionadas en el subapartado inmediatamente anterior, de una manera correcta y legal, sin incurrir en ningún tipo específico de malversación en cuanto a cada uno de los proyectos.

Para esto es necesario indagar detalladamente en cuáles son las normas o reglas que las rigen para evitar utilizarlas de una manera incorrecta y poder incursionar a más personas en el uso legal de estas, con el fin de una mejor experiencia a la hora de utilizarlas.

Una de las características clave para la inclusión de todas las opciones detalladas anteriormente en ésta infraestructura, fue cada uno de sus términos de licencia. Se verificó en cada uno de estos, que no hubiese limitaciones ni restricciones en cuanto al uso, copia, modificación, unificación, publicación, distribución, replica, fusión, términos de sublicencia e incluso posibilidad de venta.

Todas éstas características son posibles debido a que han sido catalogadas bajo la Licencia MIT (Ver Figura 29). Es menester resaltar que estas se caracterizan por permitir su

modificación, publicación y distribución de manera gratuita. Entre otras características permite reutilizar el software así licenciado, para que pueda utilizarse en las creaciones de otro software ya sea libre o no libre, pero teniendo en cuenta no liberar los cambios realizados al programa original.

```
Copyright (c) <año> <titular del copyright>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

**Figura 29. Texto Oficial de la Licencia MIT**

Para este tipo de licencias, se les aplican unos términos legales, los cuales tiene como finalidad la proporción y los alcances que tendrá dicha aplicación con Licencia MIT.

Estos conceptos abarcan las condiciones, las cuales, que la nota de copyright y la parte de los derechos se incluya en todas las copias o partes sustanciales del Software. Cabe resaltar que en el caso de que esta condición no se llevara a cabalidad, invalidaría la licencia. Los derechos, son un punto de gran importancia, pues de aquí se derivan los usos y la forma de distribución, modificación, etc. de estas creaciones.

Para las Licencias MIT, los derechos no tienen restricciones de ningún tipo incluyendo usar, copiar, modificar, integrar con otro Software, publicar, sublicenciar o vender copias del

Software, y además permitir a las personas a las que se les entregue el Software hacer lo mismo, muy parecidas a las características del Software libre.

Por ende, el uso de ésta licencia no implica un problema legal con los derechos de autor, o derechos de la propiedad intelectual puesto que, precisamente son de calidad libre para el uso de nuevos proyectos con el fin de generar nuevas tecnologías.

## 5.2. RORBOT

Este apartado está enfocado en describir de manera general y detallada la aplicación presentada como contribución. Además de detallar el proceso llevado a cabo para el desarrollo de la misma, su funcionalidad, y la manera en la que los usuarios pueden desenvolverse dentro de ésta.

### 5.2.1. Descripción General del Proyecto

En términos generales, **RORBOT** (**R**uby**O**n**R**ails Ro**bot**) como así fue bautizada, es una herramienta que genera cualquier combinación de configuraciones iniciales en una aplicación web. El usuario/desarrollador (rol que toma cada usuario en el sistema) crea su aplicación dentro de dicha herramienta (la cual se guarda momentáneamente dentro del servidor) y escoge la combinación de configuraciones iniciales que requiera su proyecto. RORBOT añade éstas configuraciones a la nueva aplicación, realizando de manera intuitiva los cambios respectivos al proyecto en código. Luego de esto, el usuario puede pasar a **extraer** su aplicación, pasando todo el proyecto a su ordenador. Una vez extraído el código, RORBOT procede a eliminarlo del servidor (estableciendo una práctica segura, sostenible y transparente).

Cabe resaltar, que la herramienta una vez extraído el proyecto, sólo guardará los datos suministrados por el usuario/desarrollador para entablar una estadística, a la cual tendrá acceso siempre que lo desee. Esta estadística arrojará información acerca de la combinación de configuraciones seleccionada en cada uno de los proyectos que realice dentro de la herramienta y una lista de las configuraciones más usadas. Además, el usuario/desarrollador puede catalogar alguna combinación de configuraciones iniciales como “favorita”, esto con el fin, de mantener el proyecto en el servidor, hasta que el usuario mismo desee eliminarlo, lo que le da la opción de editar dichas configuraciones cada vez que lo desee.

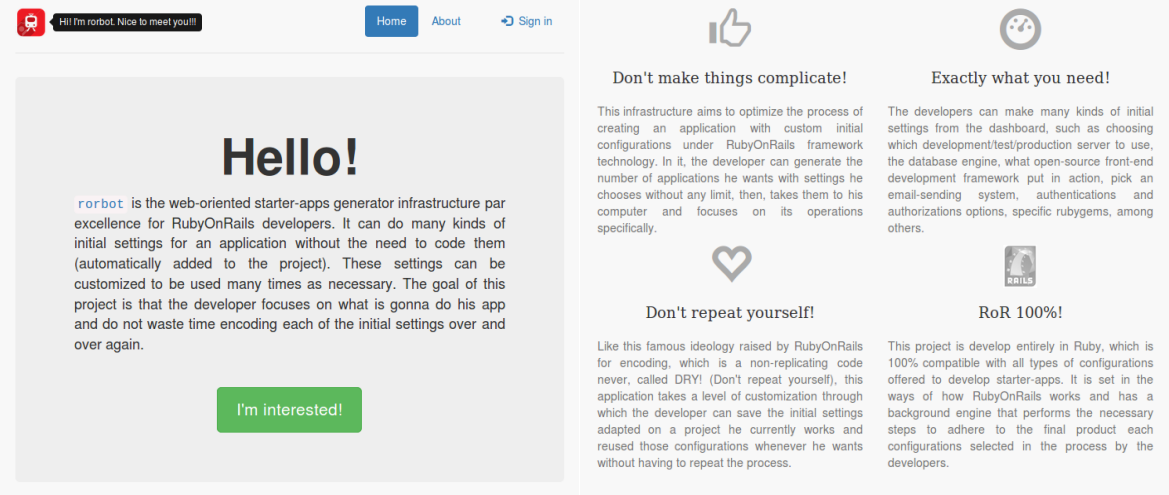


**Figura 30. Vista Principal RORBOT**

### 5.2.2. Descripción Detallada del Proyecto

Partiendo de la novedad descrita en capítulos anteriores, la cual está enfocada en orientar el proceso de adaptación de configuraciones iniciales en una aplicación web que está siendo desarrollada, se pueden detallar varios aspectos importantes de la contribución.

RORBOT presenta una forma de desarrollar aplicaciones con configuraciones iniciales diferente a las convencionales, las cuales emplean un archivo de configuración llamado “recipe” (receta), escrito enteramente en Ruby, el cual organiza una combinación de configuraciones detalladas por el usuario (vía terminal y en secuencia). Diferente a ello, la arquitectura de ésta herramienta contiene el diseño de un motor “background” (completamente transparente al usuario) que recibe los datos que el usuario le proporciona (como por ejemplo, el nombre que le ha dado a su aplicación y la combinación de configuraciones iniciales escogida) y automatiza el proceso de creación de la aplicación y la edición de código en la misma, adaptándola a los parámetros provistos. (Ver Figura 31)


Figura 31. Esquema Básico de Procesos en RORBOT

La herramienta está creada dentro de un ambiente GNU/Linux y trabaja sobre un servidor Puma, con un motor de bases de datos en PostgreSQL y puesta en marcha gracias al Framework RubyOnRails. Debido a esto, su Back-End está desarrollado enteramente en lenguaje Ruby y su Front-End está basado en HAML (una gema de Rails para escribir código HTML abstracto), SASS (un preprocesador de código CSS) y JavaScript/jQuery, además de cierto porcentaje de código en lenguaje Ruby embebido en las vistas, con datos que suministran los controladores (funcionalidad que permite la gema HAML mencionada anteriormente).

La funcionalidad de la herramienta está basada en 2 grandes controladores, los cuales se encargan del tratamiento de los datos que proporciona el usuario: las aplicaciones que se desarrollan y la personalización de cada una de esas aplicaciones. Cada uno de esos controladores contiene un sistema de seguridad que impide el acceso a usuarios no registrados al sistema y restringe el acceso a los registrados, dando permiso para ingresar únicamente a las aplicaciones creadas por los mismos y a los datos de cada una de esas aplicaciones, sin permitir el acceso a las creadas por terceros.

Contiene además, varios modelos con cada una de las configuraciones posibles y sus opciones, además de los 2 grandes modelos que se comunican con los 2 controladores mencionados anteriormente: aplicaciones y personalización. Por medio de éstos se hacen

los salvamentos de los datos suministrados, y las lecturas cuando el software así lo requiera.

Entre otras funcionalidades se encuentra la de los Mailers, con los que se realizan envíos masivos de correo electrónico, generados desde la misma aplicación y gestionados por una aplicación externa llamada Mandrill App, la cual se encarga de hacer llegar dichos correos a sus destinatarios.

### 5.2.3. Descripción del Proceso de Desarrollo

En términos generales, el desarrollo de ésta contribución se llevó a cabo en diferentes etapas. Principalmente, se establecieron los requerimientos (para éste caso, se hizo el estudio previo mencionado en el apartado anterior), para escoger los tipos de configuración inicial y cada una de sus opciones a presentar para los usuarios/desarrolladores.

Luego, ya en fase de desarrollo, se establecieron los modelos para poder estructurar de manera adecuada la base de datos de la aplicación y las funciones a implementar en los controladores. Por último, se desarrolló la interfaz de usuario. Fueron construidas las vistas, adaptándolas a los datos que suministran las funciones de cada uno de los controladores, y utilizando las clases que Bootstrap establece para el diseño de cada uno de los tipos de etiqueta utilizados. Para finalizar, fueron implementadas una serie de pruebas unitarias, por módulos y de prueba y error, para verificar el correcto funcionamiento de la aplicación.

El desarrollo específico de los controladores y las interfaces de usuario, fue elaborado por módulos. Con los modelos ya generados, y el archivo de gemas (bibliotecas de Ruby) de la aplicación ya establecido, se desarrolló cada módulo en orden de dependencia, siendo el más dependiente, el último en ser elaborado.

- ∅ El primer módulo es el de autenticaciones (Ver Figura 32), en el cual, el usuario entrante tiene acceso a varias funciones, entre las cuales están: El login, el registro en el sistema, la recuperación de la contraseña (en caso de estar registrado y

haberla perdido) y la revalidación de la cuenta (en el caso de que ocurra un error en el envío del correo electrónico de validación de la cuenta al momento de llevarse a cabo el registro)

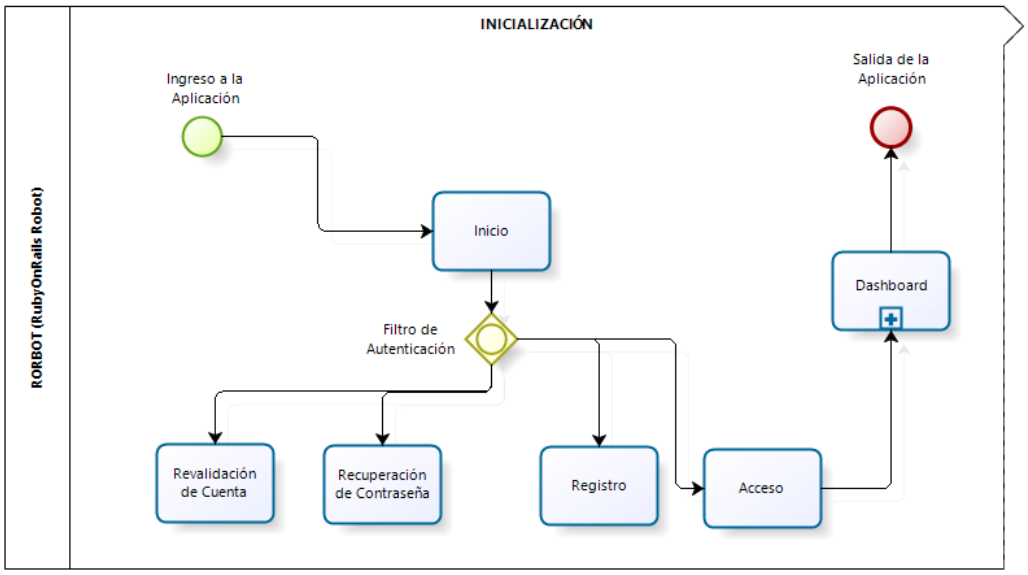


**Figura 32. Arquitectura del Módulo de Autenticación**

∅ El segundo módulo es el del tablero de operaciones (aquí denominado “dashboard” - Ver Figura 33), en el cual, el usuario autenticado tiene acceso a otros módulos de la herramienta: Aplicaciones (para crear nuevos desarrollos y ver los existentes), personalización (configuraciones iniciales predilectas), estadísticas (configuraciones más usadas, entre otras) y configuración de su perfil (datos personales, perfiles en redes sociales como Twitter, GitHub, etc.)

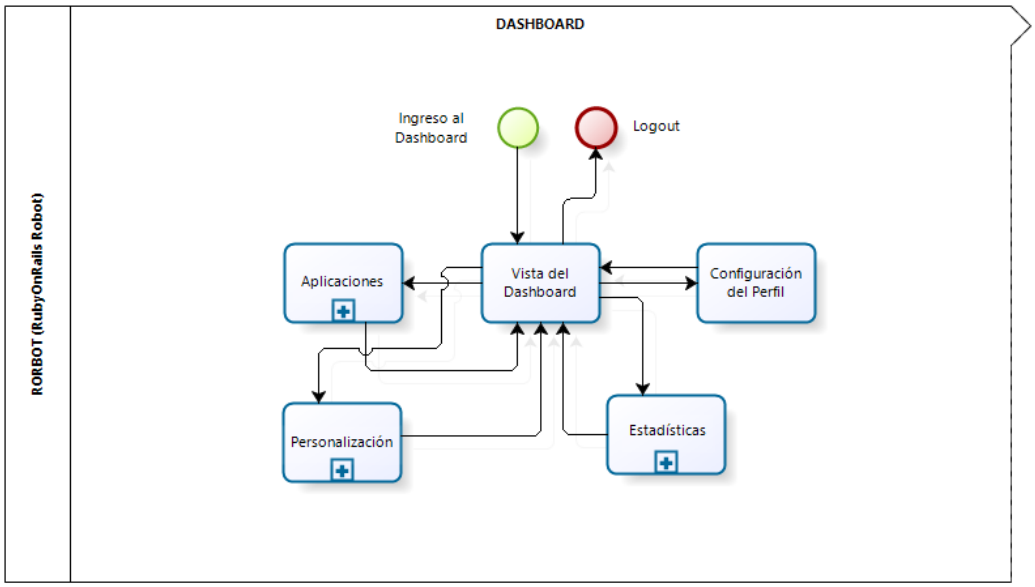


Figura 33. Arquitectura del Módulo del Dashboard

∅ El siguiente módulo es el de Aplicaciones (Ver Figuras 34 y 35). En éste, el usuario/desarrollador autenticado, tiene acceso a una lista de sus aplicaciones iniciales desarrolladas a partir de la herramienta, además de poder acceder a cada una de ellas. Sumado a esto, tiene la opción de crear nuevas aplicaciones iniciales que aparecerán en la lista anteriormente mencionada, una vez sean dadas de alta en la herramienta. Una vez configuradas las aplicaciones, el usuario tendrá acceso a extraerlas, para llevarlas a su ordenador y seguir el desarrollo pertinente de manera local.

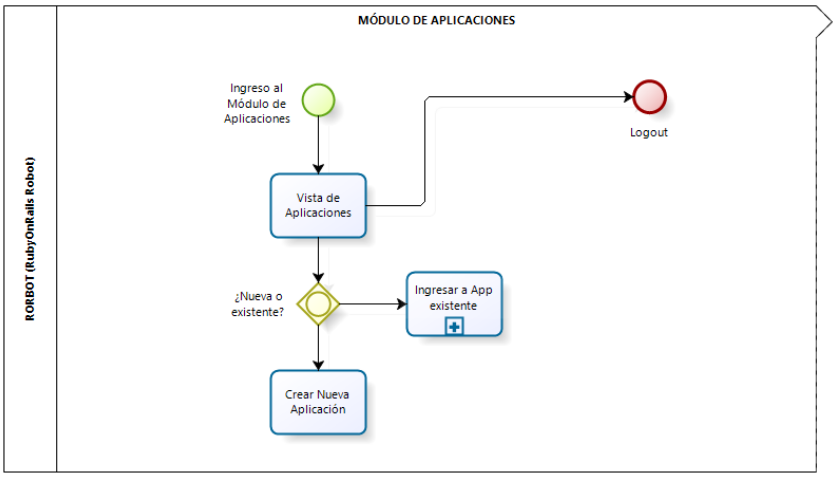


Figura 34. Arquitectura del Módulo de Aplicaciones (1)

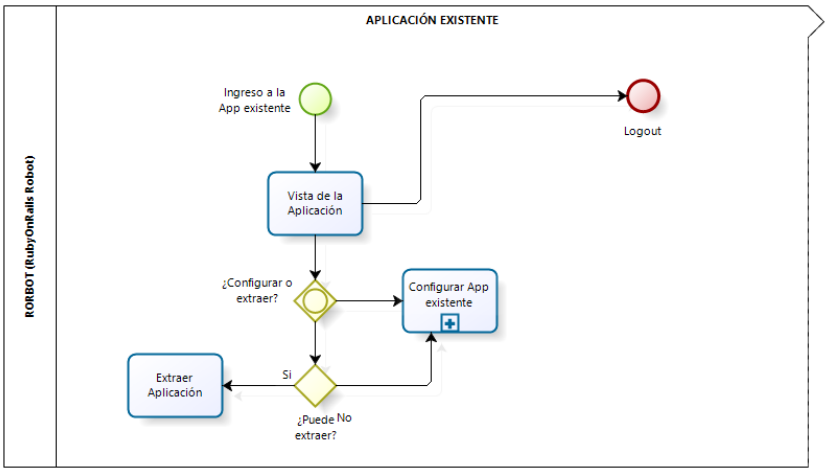


Figura 35. Arquitectura del Módulo de Aplicaciones (2)

∅ Por último, fue desarrollado el módulo de las configuraciones (Ver Figura 36), en el cual fueron provistas para el usuario/desarrollador, las diferentes configuraciones mencionadas en apartados anteriores y cada una de sus respectivas opciones.

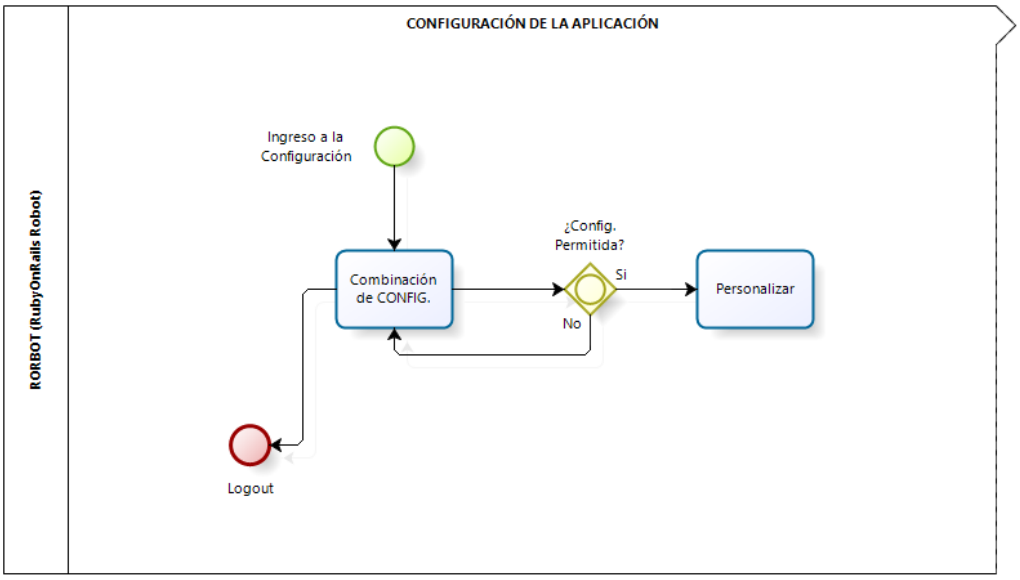


Figura 36. Arquitectura del Módulo de Configuración

### 5.2.4. Descripción de la Interfaz de Usuario

Principalmente, la aplicación cuenta con un módulo de autenticación, el cual es proporcionado por la gema **Devise**, en donde el usuario puede registrarse, validarse y conectarse a la herramienta. Esta le proporciona al momento del registro, un correo electrónico de validación de usuario para posteriormente permitirle el ingreso.

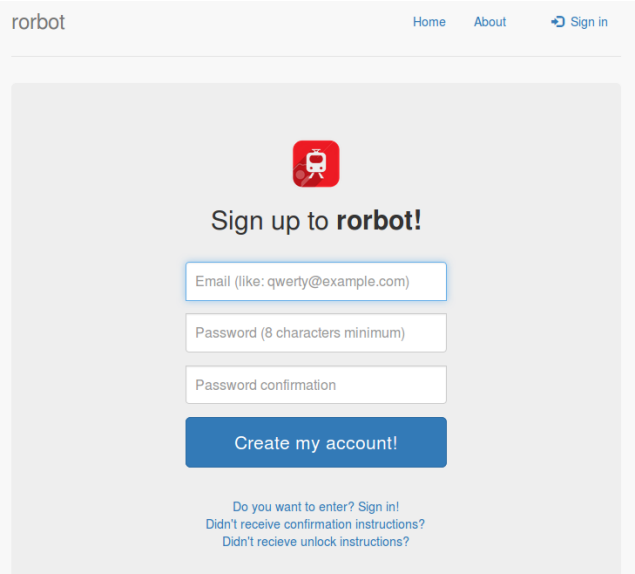


Figura 37. Vista de la Plantilla de Registro

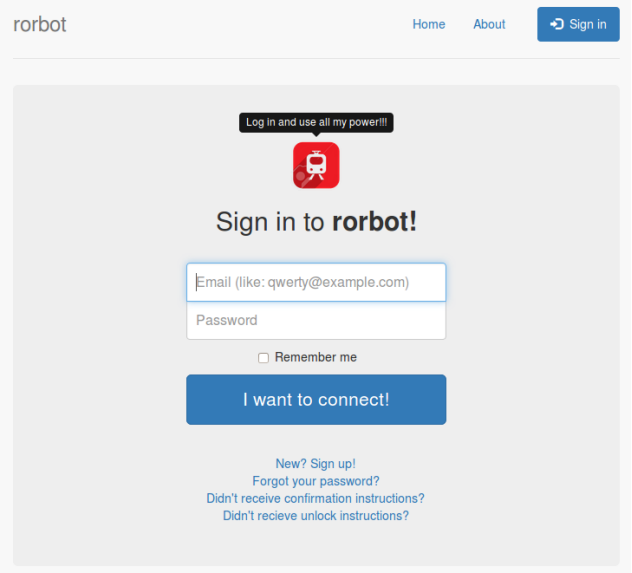


Figura 38. Vista de la Plantilla del Login

Una vez conectado, el usuario tendrá acceso a su “dashboard” (tablero), en donde va a tener varios registros a su disposición. Cada opción es un enlace a cada módulo provisto por el sistema para el uso exclusivo del usuario/desarrollador.

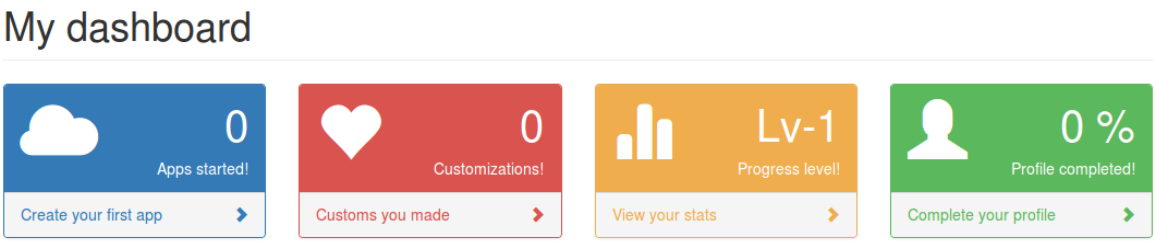


**Figura 39. Dashboard o Tablero de Operaciones**

El primero (ver Figura 39 – de izquierda a derecha), es el conteo de aplicaciones iniciadas en ésta herramienta, módulo en el cual, el usuario/desarrollador va a tener acceso a la lista de aplicaciones que ha iniciado, con todos los datos de cada una de éstas y su estado actual. Con el contador en ceros, la herramienta entiende que el usuario es nuevo y le permite tener acceso a un enlace directo para que pueda crear su primera aplicación.

El segundo, es un contador de personalizaciones, que el usuario ha sistematizado al momento de crear aplicaciones con cierta combinación de configuraciones iniciales y en la cual, a dicho proyecto, lo ha catalogado como “favorito”. En éste módulo aparecen dichas combinaciones de configuraciones iniciales agrupadas, para que el usuario pueda hacer uso de ellas, siempre que lo desee.

El tercero, es un módulo de estadísticas, las cuales va construyendo el usuario/desarrollador a medida que avanza su experiencia dentro de la herramienta. Contiene los datos de las combinaciones de configuraciones iniciales usadas, una lista de las configuraciones más usadas y determina un nivel por parte del usuario, que se mide de acuerdo a la experiencia que lleva dentro de la herramienta.

El cuarto, es un nivel (en porcentajes) de la completitud del perfil. Ya que el usuario puede ingresar casi que de manera instantánea a la aplicación, está previsto un módulo para que éste complete sus datos, que incluyen entre otros, preferencias de gemas y otras funciones predeterminadas en Rails y enlaces a sus perfiles de redes sociales (Twitter, GitHub, entre otras) para permitirle realizar interacciones entre las redes y RORBOT.

Al momento de crear nuevas aplicaciones en la herramienta, el usuario solo debe escoger el nombre provisional que va a colocarle (después de extraída la aplicación puede ser cambiado su nombre sin inconvenientes), una breve descripción de la misma, y el motor de bases de datos de su preferencia (dentro de los RORBOT mantiene como opciones posibles). Luego de ello, la herramienta se encarga de crear la aplicación bajo éstas condiciones, y salvarla (temporalmente) en el servidor. (Ver Figura 40)

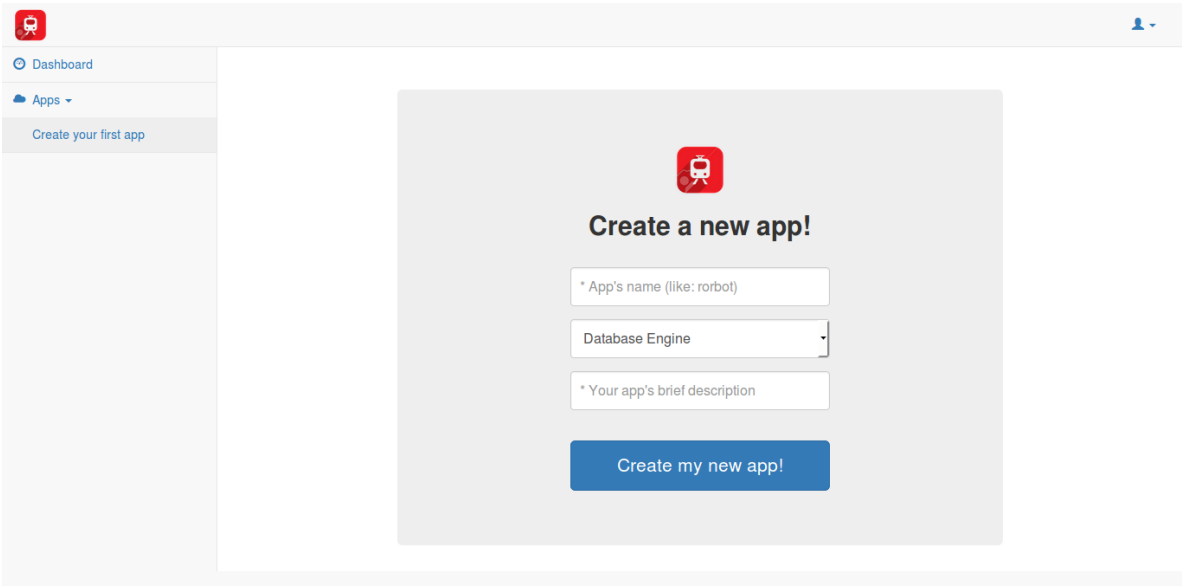


Figura 40. Formulario de Creación de la Nueva App

Una vez creada, pasa a ser parte de la lista de aplicaciones (Ver Figura 41) a la que podrá hacer el seguimiento respectivo el usuario propietario de la misma. Además, tendrá acceso a los datos específicos de la aplicación recién creada (ver Figura 42) y a su módulo de configuración, en la que podrá agregarle la combinación de configuraciones iniciales de su preferencia (ver Figura 43).

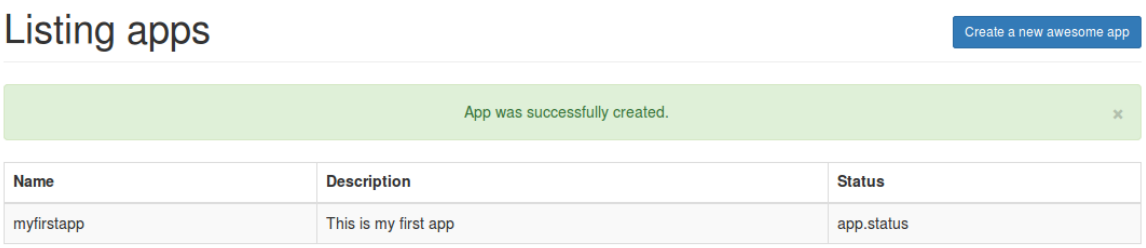


Figura 41. Listado de Aplicaciones Iniciales Creadas

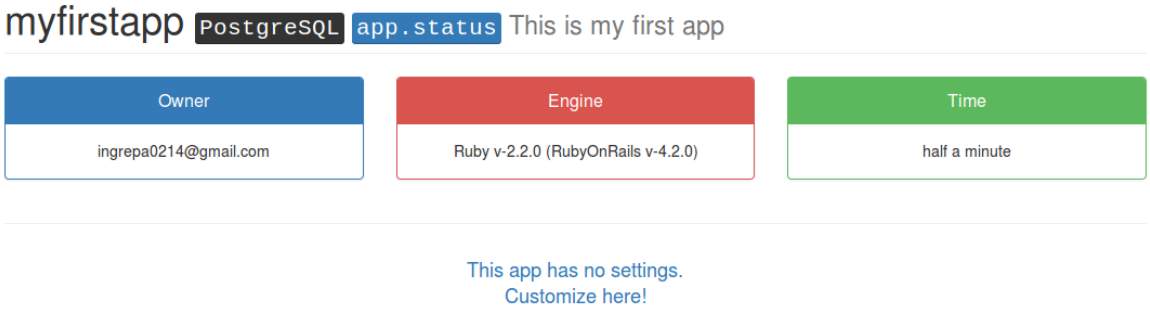


Figura 42. Vista Detallada de la Aplicación Inicial Creada

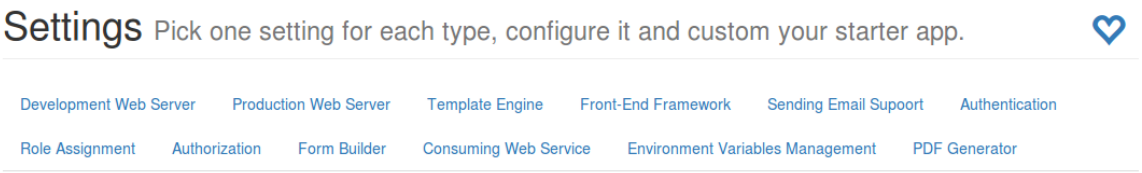


Figura 43. Módulo de Configuraciones Iniciales

En la vista de configuraciones, el usuario puede catalogar su proyecto como “favorito” (corazón) para agregar dicha combinación de configuraciones a escoger, al módulo de personalizaciones, en el cual no solo va a tener un acceso más rápido, sino que va a poder editar dicha combinación de configuraciones cada que lo desee.

Además, en la misma vista, tiene acceso para seleccionar entre las múltiples opciones de configuración que RORBOT ofrece, las cuales son: Servidor Web en Fase de Desarrollo, Servidor Web en Fase de Producción, Gestor de Plantillas (ERB/HAML), Entorno de Código Abierto para Desarrollo Front-End, Soporte para Envío Masivo de Correo Electrónico, Gestor de Autenticaciones, Gestor de Asignación de Roles, Gestor de Autorizaciones, Constructor de Formularios, API de Consumo de Servicios Web, Gestor de Variables de Entorno y Generadores de PDF. (Ver Figura 44)

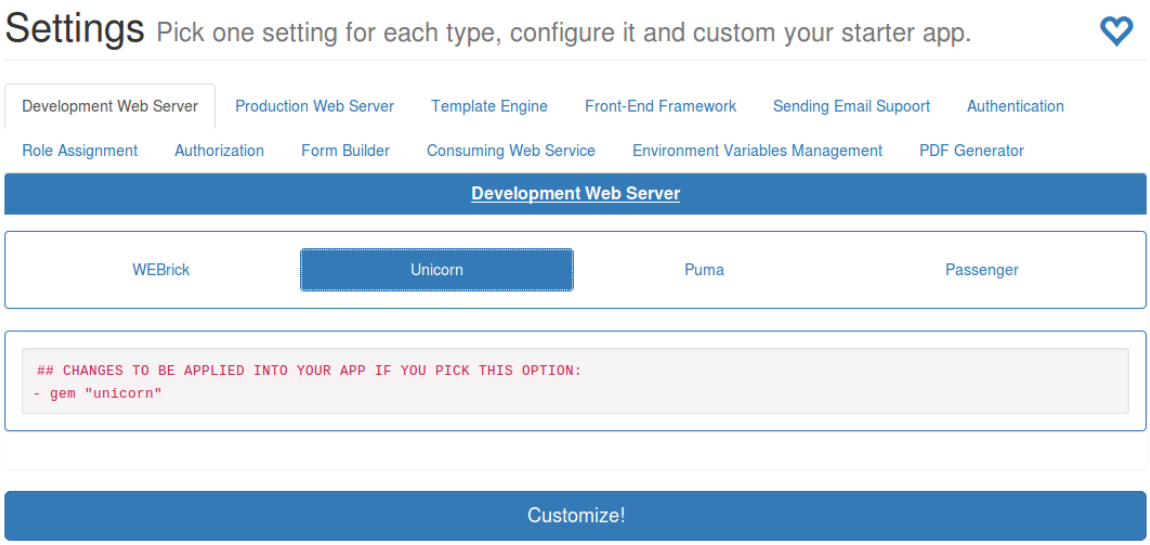


**Figura 44. Método de Selección de Configuraciones**

Cada vez que el usuario seleccione una opción de configuración, la herramienta le mostrará las modificaciones que hará sobre el proyecto, de acuerdo a la opción escogida. Y le habilitará el poder "salvar" la combinación de configuraciones iniciales escogida para la aplicación.

Una vez salvada la combinación de configuraciones iniciales escogida, ésta aparecerá en la vista de la aplicación, con todos los datos pertinentes. (Ver Figura 45). Además, dicha aplicación ya estará habilitada para ser **extraída** al ordenador de su desarrollador y posteriormente eliminada del servidor (siempre y cuando no haya sido catalogada como "favorita" por el usuario, lo que hace que ésta se mantenga en el servidor hasta que el usuario mismo decida eliminarla).

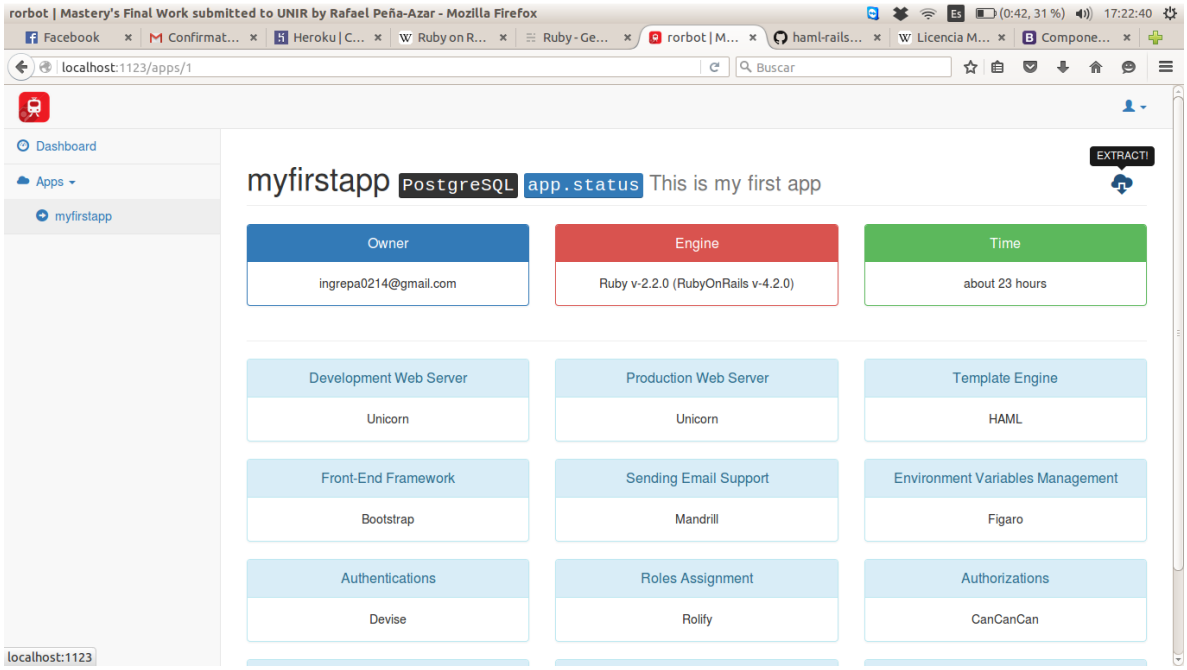


Figura 45. Vista Detallada de la Aplicación Inicial Configurada

## 5.3. EVALUACION

En éste apartado se pretende realizar una evaluación de la herramienta propuesta como contribución, mediante un análisis de métricas entre las cuales se valoran los siguientes aspectos:

- ∅ Usabilidad
- ∅ Funcionalidad

El análisis de la usabilidad, evalúa los aspectos referentes a la facilidad de uso de la aplicación, siendo un tópico sumamente importante, asignando un peso del 50% al concepto. Mientras que el análisis de funcionalidad, hace referencia a las funciones que presta la herramienta, a la cual se le ha asignado, al igual que el ítem anterior, un peso de concepto del 50%. No se ha evaluado accesibilidad, debido a que, por razones de concepto de la herramienta misma, únicamente va dirigida a desarrolladores de aplicaciones en Rails.

Los criterios utilizados para realizar dicha evaluación, son los mismos propuestos y establecidos por Olsina y Camarazana:

- ∅ Cada parámetro se puntúa de 1 a 10.
- ∅ Los parámetros que no aplican, no se puntúan y no cuentan en la evaluación.

El primer ítem evaluado es la **Usabilidad** (Ver Tabla 1) en la cual determinamos por medio de los criterios, el nivel de facilidad de uso de la aplicación.

**Tabla 1. Métrica de Usabilidad de la Aplicación**

| 50% | | USABILIDAD | 5,8 | 2,9 |
|---|---|---|---|---|
| | 20% | **COMPRENSIBILIDAD DEL SITIO** | **7** | **1,4** |
| | | Página Principal | 7 | |
| | | Consistencia en la Navegación | 7 | |
| | 20% | **RETROALIMENTACIÓN** | **1** | **0,2** |
| | | Indicadores de Ultima Actualización | 1 | |
| | | Directorio de Enlaces | 1 | |
| | | Facilidades FAQ | 1 | |
| | 20% | **ASPECTOS DE INTERFACES** | **7** | **1,4** |
| | | Estabilidad de la Presentación | 8 | |
| | | Uniformidad del Estilo del Sitio | 8 | |
| | | Preferencia Estética | 7 | |
| | 20% | **MISCELANEAS** | **7** | **1,4** |
| | | Descargas | 4 | |
| | | Intrusión Publicitaria | 10 | |
| | 20% | **TEXTOS** | **7** | **1,4** |
| | | Textos Adaptados a la Web | 7 | |

La métrica planteada en la valoración del ítem **Usabilidad**, muestra que el nivel general de facilidad de uso del sitio corresponde a un nivel un poco más alto del promedio.

El segundo ítem evaluado es la **Funcionalidad** (Ver Tabla 2) en la cual determinamos el nivel de servicio que prestan las funciones de la aplicación en cuestión.

**Tabla 2. Métrica de Funcionalidad de la Aplicación**

| 50% | FUNCIONALIDAD | | 9,0 | 4,5 |
|---|---|---|---|---|
| | **20%** | **Navegabilidad Local** | **7** | **1,4** |
| | | Nivel de Interconexión | 6 | |
| | | Orientación | 8 | |
| | **20%** | **Navegabilidad Global** | **10** | **2,0** |
| | | Acoplamiento de Subsitios | 10 | |
| | **20%** | **Objetos de Control Navegacional** | **9** | **1,8** |
| | | Permanencia | 9 | |
| | | Estabilidad | 9 | |
| | | Nivel de Desplazamiento | 9 | |
| | **20%** | **Predicción Navegacional** | **9** | **1,8** |
| | | Enlace con Título | 9 | |
| | **20%** | **Funciones Específicas** | **10** | **2,0** |

La métrica planteada en la valoración del ítem **Funcionalidad**, muestra que están bien definidas las funciones que presta la herramienta, superando ampliamente el promedio.

En base a los datos arrojados por las métricas utilizadas para evaluar tanto el nivel de usabilidad como el de funcionalidad, la siguiente tabla (Ver Tabla 3) muestra ya en términos generales, los resultados obtenidos en la evaluación.

**Tabla 3. Resultados Obtenidos por la Métrica Aplicada**

| ASPECTO | PUNTUACION | PONDERADO | PUNT. FINAL |
|---|---|---|---|
| USABILIDAD | 5,8 | **50%** | **2,9** |
| FUNCIONALIDAD | 9,0 | **50%** | **4,5** |
| **TOTAL PUNT.** | **–** | **100%** | **7,4** |

Como se puede observar, el resultado de la valoración **(7,4)** determina el nivel de favorabilidad con el que cuenta la herramienta al momento de su presentación como contribución al proyecto presentado. Posee un nivel promedio de usabilidad, lo cual no implica que la aplicación sea de difícil uso para los usuarios, y cuenta con una funcionalidad bien definida, que parte desde un justo estudio previo del caso hasta un desarrollo que cumple con todos los objetivos planteados en el inicio del proyecto.

# CAPITULO 6
# CONCLUSIONES Y TRABAJOS FUTUROS

## 6.1. CONCLUSIONES

A lo largo de éste proyecto, se ha enfatizado en la importancia de hacer que los desarrolladores se centren en la funcionalidad de sus proyectos y no en temas tan triviales como la configuración, el cual, manualmente, toma más tiempo del debido. Además, se ha intentado centrar el proyecto como tal, a la filosofía que promueve RubyOnRails: **No repetir**, lo cual es perfectamente adaptable teniendo en cuenta que en muchas ocasiones, los desarrolladores de aplicaciones web en Rails, tienden a utilizar en muchos de sus proyectos, para algunas funciones, los mismos tipos de configuración, por lo cual, para cada uno de esos proyectos, tienden a repetir cierta combinación de configuraciones iniciales.

El objetivo principal se cumplió a cabalidad mediante la herramienta propuesta, debido a que le otorga al usuario/desarrollador la rapidez necesaria para llevar a cabo todas las configuraciones iniciales que desee adaptar a una aplicación y por medio de una función muy simple de usar, permite mantener las configuraciones para que las pueda usar para extraer más aplicaciones, adaptando el mensaje de Rails de no repetir y ahorrar tiempo.

El proyecto como tal, inició haciendo un estudio previo de las configuraciones posibles a adaptar a la herramienta, se efectuó un análisis para determinar cuáles debían hacer parte, teniendo en cuenta que algunas de ellas deben ser obligatorias para el correcto funcionamiento de la aplicación, y otras, que, aunque no son de carácter obligatorio, son muy útiles o son ya muy famosas y están rígidamente testeadas en su campo como para no ser tenidas en cuenta.

Posteriormente, se procedió a desarrollar la herramienta en base al análisis realizado, generando los modelos necesarios para estructurar la base de datos, los controladores para generar toda la funcionalidad de la aplicación, y las vistas (interfaces) para hacerle llegar a los usuarios, todos los datos suministrados por los controladores, además de permitirle a los mismos, configurar las aplicaciones que generen a partir de la herramienta.

Gracias al módulo de aplicaciones implementado, el usuario/desarrollador puede crear nuevas aplicaciones directamente desde la herramienta (y no vía terminal de comandos) y a partir de allí, realizar la combinación de configuraciones iniciales que desee adaptar a dicha aplicación. Y es, debido al módulo de configuraciones, que el usuario/desarrollador puede personalizar las combinaciones de configuraciones iniciales seleccionadas para utilizarlas cuantas veces lo desee, lo cual valida a la perfección, el objetivo planteado al principio de éste proyecto.

La aplicación cuenta con un nivel de usabilidad tal, que se hace intuitivo su manejo por parte del usuario/desarrollador, lo cual hace que no tenga ningún inconveniente durante su uso, y que tan siquiera sea necesario leer un manual para abordar la herramienta. Además, su aplicabilidad es de un campo bastante extenso, debido a que no existen límites para los desarrollos web a partir del framework RubyOnRails, por lo que se puede llegar a iniciar cualquier tipo de aplicación a partir de la herramienta presentada como contribución.

## 6.2. TRABAJOS FUTUROS

En el presente apartado se propone una serie de propuestas y líneas futuras con el fin de retroalimentar la presente investigación y ampliar el camino propuesto para el desarrollo de aplicaciones con configuraciones iniciales personalizadas.

- ∅ Se podría proponer un incremento en la cantidad de tipos de configuración que ofrece la aplicación, así como incrementar el número de opciones en cada una de las configuraciones presentadas.

- ∅ Otro proyecto podría ser el de ampliar los horizontes que actualmente ésta herramienta posee y abrir un campo de investigación en el cual se determine la inclusión de otros tipos de entornos de desarrollo a la herramienta actual y bajo diferentes lenguajes de programación. Bien podría ser el caso de Python/Django, framework que está creciendo cada vez más a nivel mundial. También podría atacarse el framework Laravel, con el cual PHP ha logrado evolucionar notablemente.
- ∅ Se podría intentar realizar también una inclusión de frameworks de desarrollo híbrido, no como opción de configuración para los que generalmente son enfocados a front-end, sino como opción de entorno, en el cual también se determinen opciones de configuración para desarrollo de aplicaciones iniciales en los mismos. Un ejemplo de framework enfocado al desarrollo híbrido que pueda incluirse puede ser IONIC.

# REFERENCIAS

B.V., P. H. (2012). *Phusion Passenger*. Recuperado el 4 de Diciembre de 2015, de https://www.phusionpassenger.com/

Bates, R., Rite, B., Shelley, M., & Ermolovich, V. (2011). *CanCanCan's GitHub Repo*. Recuperado el 8 de Diciembre de 2015, de https://github.com/CanCanCommunity/cancancan

Ben-Ari, N., Shatrov, K., & Witek, G. (2010). *Sorcery's GitHub Repository*. Recuperado el 7 de Diciembre de 2015, de https://github.com/NoamB/sorcery

Brown, G. (2014). *PrawnPDF*. Recuperado el 10 de Diciembre de 2015, de http://prawnpdf.org

Catlin, H., & Weizenbaum, N. (2006). *HAML GitHub Repository*. Recuperado el 5 de Diciembre de 2015, de https://github.com/haml/haml

Collins, K. (2012). *HolyGrailHarness GitHub Repo*. Recuperado el 2 de Febrero de 2016, de https://github.com/metaskills/holy_grail_harness

Ellis, S. (2015). *An Introduction to ERB Templating*. Recuperado el 4 de Diciembre de 2015, de http://www.stuartellis.eu/articles/erb/

Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Boston: Addison-Wesley.

Harrington, D. (2010). *SavonRB*. Recuperado el 14 de Diciembre de 2015, de http://savonrb.com/version2/

Intridea. (2011). *Rails Wizard*. Recuperado el 2 de Febrero de 2016, de www.railswizard.org

Johnston, J. (2014). *Rails4 StarterKit GitHub Repo*. Recuperado el 2 de Febrero de 2016, de https://github.com/starterkits/rails4-starterkit

Kehoe, D. (2012). *Send Email with Rails*. Recuperado el 5 de Diciembre de 2015, de http://railsapps.github.io/rails-send-email.html

Nicklas, J. (2012). *Pundit GitHub Repo*. Recuperado el 7 de Diciembre de 2015, de https://github.com/elabs/pundit

Nunemaker, J. (2008). *HTTParty Jnunemaker's Repository (GitHub)*. Recuperado el 12 de Diciembre de 2015, de https://github.com/jnunemaker/httparty

Phoenix, E. (2014). *Puma GitHub Repository*. Recuperado el 4 de Diciembre de 2015, de https://github.com/puma/puma

Plataformatec. (2009). *Devise's GitHub Repository*. Recuperado el 7 de Diciembre de 2015, de https://github.com/plataformatec/devise

Puente, R. (2014). *El Blog de Rodrigo Puente*. Recuperado el 23 de Diciembre de 2015, de http://blog.rodrigopuente.com/ruby-gemas-que-son-y-para-que-sirven/

RailsApps. (2012). *RailsApps Composer GitHub Repo*. Recuperado el 2 de Febrero de 2016, de https://github.com/RailsApps/rails_apps_composer

RailsCoreTeam. (2013). *RubyOnRails*. Recuperado el 21 de Diciembre de 2015, de http://guides.rubyonrails.org/getting_started.html

Richert, S. (2012). *Figaro GitHub Repository*. Recuperado el 6 de Diciembre de 2015, de https://github.com/laserlemon/figaro

Show, Z. A. (2007). *Unicorn Server's Bogomips Site*. Recuperado el 4 de Diciembre de 2015, de http://unicorn.bogomips.org/

Sterrett, M. Z. (2011). *RubyGems*. Recuperado el 18 de Diciembre de 2015, de https://rubygems.org/gems/wicked_pdf

Takahashi, M., & Gotou, Y. (2000). *WEBrick GitHub Repository*. Recuperado el 4 de Diciembre de 2015, de https://github.com/nahi/ruby/tree/webrick_trunk