Universidad Nacional Autónoma de Nicaragua, UNAN – León
Facultad de Ciencias y Tecnología
Ingeniería en Telemática

Elaboración de guías prácticas de laboratorio para el componente curricular Software como un Servicio de la carrera Ingeniería en Telemática del Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua, León.

Monografía para optar al título de Ingeniero en Telemática

Autor(es): Br. Lenin Ramón García Calero
Br. Jefferson Alexander Rojas Saballo

Tutor: M.Sc. Julio César González Moreno

León, Nicaragua
Abril de 2018

Elaboración de guías prácticas de laboratorio para el componente curricular Software como un Servicio de la carrera de Ingeniería en Telemática del Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua, León.

Resumen

El desarrollo del presente trabajo tiene el propósito de aportar un documento de guías prácticas de laboratorio para el componente curricular Software como un Servicio, correspondiente al plan académico 2011 de la carrera Ingeniería en Telemática de la Facultad de Ciencias y Tecnología UNAN-León; teniendo como objetivo que sirva de apoyo tanto a los docentes de Departamento como a los estudiantes que forman parte de este, o a cualquier otra persona interesada en el tema.

Las guías tienen definido un formato específico de documento, que facilite su lectura y desarrollo, éstas siguen una secuencia de acuerdo con el nivel de complejidad de los temas abordados en el componente curricular. También se ha elegido el uso del framework Ruby on Rails como parte fundamental para completar las guías. Con estas guías prácticas, se pretende que los estudiantes que cursan el componente antes mencionado puedan desarrollar las competencias que se quieren lograr en la microprogramación de éste.

Dentro de las guías prácticas propuestas se abordan temas en relación con el framework Ruby on Rails, de los cuales se pueden mencionar: Patrón de diseño MVC, comandos básicos Ruby on Rails, Jerarquía de directorios, Etiquetas del lenguaje ERB, integración de librerías como bootstraap entre otros; también temas relacionados con lo que es el lenguaje Ruby: Ejercicios string y enteros, definición de métodos, uso de estos métodos, definición de hash, clases. Y por último una sección en la que se propone la elaboración de unos pequeños proyectos prácticos donde los estudiantes podrán llevar a cabo y relacionar la mayoría de los temas antes mencionados.

Dedicatoria (s)

El presente trabajo es justo dedicárselo a Dios por haberme dado lo más valioso como lo es la vida y luego a la persona más importante en mi vida, la que me ha dado la motivación para no desfallecer en los momentos difíciles y siempre esforzarme a dar lo mejor de mí en mi preparación profesional ella es mi hija Bélenin Sofía García Morales.

También se lo dedico a mi familia que han sido pilares fundamentales en los logros que he venido cosechando en mi vida, a todas esas personas que me han aconsejado, guiado y apoyado entre ellas mí esposa quien en estos últimos años me ha acompañado e incentivado en mi crecimiento personal y profesional.

Lenin R. García

El presente trabajo se lo tengo que dedicar a Dios por darme la oportunidad y por permitirme haber terminado la carrera.

De igual forma a mi madre, mis hermanos, a mi familia y todas las personas que siempre me han apoyado.

Jefferson A. Rojas

Agradecimiento

Mi gratitud es a Dios por todas las bendiciones puesta en mi vida, a cada uno de los miembros de mi familia en especial a mis padres que me han dado su apoyo y sobre todo el ejemplo de trabajo y honradez. A cada una de las personas que me han apoyado y guiado a lo largo de esta etapa.

A nuestro tutor M.Sc. Julio César González Moreno, por el apoyo y empeño mostrado para hacer posible este trabajo al igual a todos los profesores que nos compartieron sus conocimientos y nos formaron profesionalmente.

A cada uno de mis compañeros que en algún momento de estos cincos años han trabajado conmigo.

Lenin R. García

Agradecimiento

Primeramente tengo que agradecer a Dios por todas las bendiciones que me ha regalado en la vida, por darme sabiduría y las fuerzas para seguir adelante, porque sin su ayuda nada es posible.

A mi madre Johanna Saballos y a mí tía Janitzia Saballos por ayudarme, apoyarme y estar siempre a mí lado; a mí tío Marcelo Hernández por todas sus enseñanzas, apoyo; a Yader Cisneros por ayudarme siempre y a toda mi familia en general.

A tutor M.Sc. Julio César González Moreno, por todo su apoyo, sus orientaciones, su tiempo y enseñanzas que me ha brindado, así también a todos los docentes que me ayudaron a lo largo de carrera.

A mis compañeros y amigos que tuve a lo largo de la carrera.

Jefferson A. Rojas

Índice general

Índice de figuras

Capítulo I:

Aspectos introductorios

1. Introducción

El principal propósito del presente trabajo es la elaboración de guías prácticas de laboratorio para el componente curricular de Software como un Servicio, impartido en el ciclo VIII del plan académico 2011 en la carrera de Ingeniería en Telemática de la Universidad Nacional Autónoma de Nicaragua, León; dicho componente tiene un total de 3 créditos y está compuesto por un total 60 horas divididas entre teoría y práctica a lo largo del semestre, además tiene como pre-requisito el componente Programación Orientada a la Web.

En este trabajo se elaboraron una serie de guías prácticas de laboratorio y fue realizado con el interés que sea de utilidad para que los docentes del Departamento de Computación que imparten el componente tengan un documento guiado que cuente con una secuencia de complejidad de acuerdo con las competencias que se plantean en la microprogramación del componente (Poveda, 2015), dichas guías tienen como base principal la utilización del framework Ruby on Rails, el cual es un framework escrito en lenguaje de programación Ruby, que emplea el patrón de diseño MVC (Modelo Vista Controlador), mayormente utilizado para el desarrollo de aplicaciones web.

La realización de las guías se dividió en cuatro grandes bloques dentro de los cuales se abordaron temas de mucha importancia según lo que se presenta en la microprogramación del componente en cuestión, como son: lenguaje ruby, creación de un proyecto en Rails y su estructura, patrón de diseño MVC, validaciones de datos, rutas en Rails, técnicas de desarrollo ágil entre otras cosas; cada una de las guías poseen ejercicios planteados de manera secuencial en relación a las temáticas que se abordan en el componente, de igual forma tienen una secuencia exponencial con respecto al nivel de dificultad y complejidad de cada una de estas, con la idea de que faciliten una mejor comprensión por parte de los estudiantes. Es importante señalar que la elaboración de las guías, están mayormente enfocadas en lo que es el desarrollo de software utilizando el framework antes mencionado (Ruby on Rails), permitiéndoles a los estudiantes obtener mayores conocimientos que les sirvan de ayuda en el mercado laboral actual.

2. Antecedentes

A través de los años desde que se habilitó el plan académico 2011 en el Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua, León; se han ido realizando trabajos monográficos en relación con la elaboración de prácticas de laboratorios para los distintos componentes de la carrera de Ingeniería en Telemática, dentro los cuales se pueden mencionar:

"Elaboración de prácticas de Switching, Routing y Servicios de Red con IPv6 para el componente Electiva X correspondiente al plan académico 2011 de la carrera de Ingeniería en Telemática del Departamento de Computación de la UNAN-León", trabajo elaborado por la Br. Delia María Jaime Toruño, el Br. Hugo Mariano García Machado y el Br. William Francisco Aguilar Zapata, en Junio 2015. En este documento se crearon propuestas de prácticas de laboratorios de Switching, Routing y Servicios de Red, usando el protocolo IPv6, con la finalidad de usarlo en la asignatura de "Despliegue de IPv6" de la carrera de Ingeniería en Telemática del Departamento de Computación de la UNAN-León dentro del cual se elaboraron 14 prácticas.

"Desarrollo de prácticas de laboratorio utilizando la filosofía hacking ético profesional, para el apoyo de la docencia de las asignaturas relacionadas con la seguridad informática del Departamento de Computación de la UNAN-León". Trabajo elaborado por el docente Ing. Jorge Soes Centeno en julio del año 2015. Se proponen prácticas de laboratorio de hacking ético que puedan ser utilizadas de forma parcial o total en las asignaturas impartidas por el Departamento de Computación, y tiene un total de 8 prácticas propuesta.

"Elaboración de prácticas de laboratorio para el componente Computación en la nube del plan académico 2011 de la carrera de Ingeniería en Telemática de la UNAN-León", trabajo elaborado por el Br. Angel Evelio Maradiaga Leytón y el Br. Ervin Ismael Montes Téllez en agosto del año 2015. Este documento tiene como objetivo principal crear propuestas de prácticas de laboratorio de Computación en la nube con la finalidad que sean de utilidad en la clase de Computación en la nube correspondiente al plan académico 2011 de la carrera Ingeniería en Telemática. En el documento se proponen 8 prácticas de laboratorio con sus respectivas soluciones.

"Prácticas de laboratorio para el componente curricular: "Tecnologías de Redes Celulares", utilizando la herramienta Atoll, para la carrera de Ingeniería en Telemática de la UNAN-León". Elaborado por la Br. Martha Elizabeth Aguinaga Mora y el Br. Edwin Ernesto Fajardo Valenzuela en el período de agosto 2015 a noviembre 2016. En este trabajo el objetivo es la elaboración de una serie de prácticas de laboratorio para el componente electivo de Tecnologías de Redes Celulares, correspondiente al plan académico 2011 de la carrera de Ingeniería en Telemática de la Universidad Nacional Autónoma de Nicaragua, en las cuales se implementan las tecnologías de las redes celulares, utilizando una herramienta de planificación de redes inalámbricas conocida como Atoll. En el documento se proponen 5 prácticas de laboratorio con sus respectivas soluciones.

Es importante mencionar que existen sitios web y documentos en los cuales se plantean ejercicios prácticos utilizando el framework Ruby on Rails dentro de los cuales se puede mencionar el sitio oficial del framework, en el cual se encuentran una serie de guías en la que se utiliza como base principal dicho framework antes mencionado, a pesar de esto, dentro del Departamento de Computación de la UNAN-León no se cuenta con un documento oficial de guías prácticas elaborado dentro de la Universidad que se aplique a la realidad de los estudiantes y que tenga relación con las temáticas y competencias planteadas en la micrograma ción del componente curricular Software como un Servicio.

3. Planteamiento del problema

A la fecha de realización del presente trabajo; dentro del plan académico 2011 de la carrera Ingeniería en Telemática de la UNAN-León en comparación con el plan académico 2007, se encuentran nuevos componentes que ayudan a fortalecer los conocimientos prácticos y teóricos de los estudiantes en el transcurso de la carrera. Sin embargo algunos de los nuevos componentes que se agregaron en el plan académico 2011, no cuentan con un documento de guías prácticas de laboratorio desarrolladas por docentes del Departamento de Computación que contenga una secuencia de contenidos que sirvan al estudiante, para una mejor comprensión de la misma y que sean de apoyo al docente, para poder impartir y evaluar la adquisición de competencias por parte de los estudiantes de manera sistemática, por consiguiente resulta indispensable la elaboración de dichas guías basadas en la competencias que se describen en sus respectivas microprogramaciones.

El componente Software como un Servicio que forma parte ciclo VIII del plan académico antes mencionado (plan 2011), es distribuido en dos partes: teoría y práctica; basándose en la micrograma ción del componente en la parte teórica se imparten temas relacionados con: computación en la nube, tipos de arquitectura en los que se basa el Software como un Servicio y la evolución que estas han experimentado; por otro lado, en la parte práctica se emplea lo que es el desarrollo de software utilizando el framework Ruby on Rails, así también se introducen temas de mucha importancia como son las técnicas de desarrollo ágil de aplicaciones web, tales como: desarrollo guiado por pruebas y desarrollo guiado por comportamiento.

En relación a las problemáticas antes mencionadas surgen las siguientes interrogantes:

¿Qué tipo de formato se debe utilizar en las guías prácticas de laboratorio para el componente curricular Software como un Servicio de la carrera de Ingeniería en Telemática del Dpto. de Computación, para que facilite a los estudiantes una mejor comprensión de las mismas?

¿Qué herramientas y versiones de software son necesarias para que el estudiante pueda realizar las soluciones a las guías prácticas de laboratorio para el componente curricular Software como un Servicio?

¿Qué secuencia se debe seguir en relación a las temáticas que se deben abordar en el desarrollo de las guías prácticas de laboratorio para que al finalizar el componente el estudiante haya obtenido los conocimientos, habilidades y actitudes necesarias para adquirir las competencias planteadas en la microprogramación?

4. Justificación

Tomando en cuenta la microprogramación del componente Software como un Servicio, y debido a que dentro del Departamento de Computación de la UNAN-León no se cuenta con un documento teórico y práctico elaborado por los docentes, que sirva de soporte a dicho componente, se vio la necesidad de crear un documento el cual contenga una serie de guías prácticas de laboratorio para el componente Software como un Servicio de la carrera de Ingeniería en Telemática.

4.1. Originalidad

Dentro del Departamento de Computación de la UNAN-León existen trabajos monográficos los cuales se basan en el desarrollo de prácticas para distintos componentes que se imparten en el transcurso de la carrera, sin embargo como se hace mención anteriormente no se cuenta con un documento de guías prácticas de laboratorio para el componente Software como un Servicio perteneciente al plan académico 2011 de la carrera de Ingeniería en Telemática de la UNAN-León, por consiguiente en el presente trabajo se propone la elaboración de guías prácticas de laboratorio para el componente Software como un Servicio en las cuales se utiliza el framework Ruby on Rails, con lo que se pretende colaborar con el Departamento de Computación para que sirva de apoyo al docente, para poder llevar a cabo la parte del laboratorio y a los estudiantes, para obtener mayores conocimiento sobre la temáticas que se abordan en el componente.

4.2. Alcance

Los principales beneficiados con la realización del presente trabajo incluyen al propio Departamento de Computación el cual posee la carrera en donde se imparte dicho componente, junto con los docentes responsables; de igual forma los estudiantes que cursan la carrera y a cualquier otra persona interesada en este tema.

4.3. Producto

Fruto de este trabajo se obtuvo una serie de guías prácticas, las cuales contemplan las siguientes características:

- Guiado: en cada guía se describe paso a paso la manera en la que se debe elaborar cada ejercicio.
- Sencillo: con relación al desarrollo de las guías, se pretende que sean elaboradas de forma simple para que faciliten una mejor comprensión de los estudiantes.
- Secuencial: esto hace referencia a los temas, el nivel de dificultad y complejidad para que cada una de las guías tenga una forma lógica y secuencial definida.

4.4. Impacto

Con la elaboración del presente trabajo, el Departamento de Computación de la UNAN-León, tendrá a su disposición un documento base para el componente Software como un Servicio, el cual servirá de mucha ayuda tanto para docentes del Departamento como para estudiantes de la carrera de Ingeniería en Telemática.

5. Objetivos

5.1. Objetivo general

Elaborar guías prácticas de laboratorio para el componente curricular Software como un Servicio de la carrera de Ingeniería en Telemática del Departamento de Computación de la Universidad Nacional Autónoma de Nicaragua, León.

5.2. Objetivos específicos

1. Definir un formato específico que se debe utilizar en las guías prácticas de laboratorio para el componente curricular Software como un Servicio de la carrera de Ing. en Telemática del Dpto. de Computación con el fin de facilitar a los estudiantes una mejor compresión de las mismas.
2. Determinar las herramientas y versiones de Software necesarias para que el estudiante pueda realizar las guías prácticas de laboratorio para el componente curricular Software como un Servicio y presente una solución satisfactoria de la misma.
3. Identificar la secuencia que se debe seguir en relación a las temáticas que se deben abordar en el desarrollo de las guías prácticas de laboratorio para que al finalizar el componente el estudiante haya alcanzado los conocimientos, habilidades y actitudes necesarias para adquirir las competencias planteadas en la microprogramación.

Capítulo II:

Aspectos teóricos

6. Marco teórico

6.1. La Nube

6.1.1. Definición.

Es el término que se le da a los servidores que almacenan datos y los gestionan, es un concepto de negocio, que se da ofreciendo servicios a través de Internet; la nube en sí, es una red de telecomunicaciones pública. (Avila Mejía, 2011)

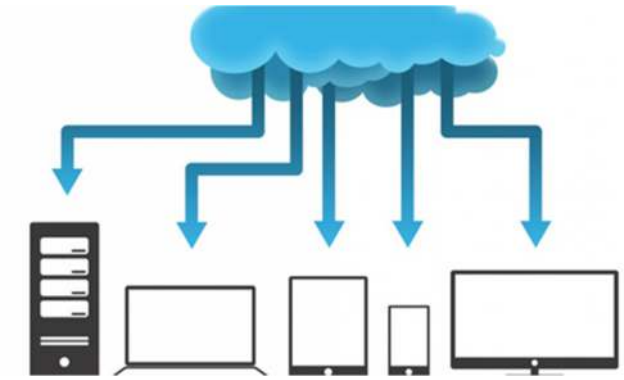

Figura 1 Interpretación anidada de la nube

6.2. Modelo de implementación en la nube

Figura 2 Tipos de nube

6.2.1. Nube pública.

Se refiere a un modelo que da servicios en la nube, en el que el proveedor del servicio hace los recursos masivamente escalables como la CPU, capacidades de almacenamiento, o aplicaciones disponibles para el público en general a través de Internet. Actualmente existen muchos proveedores de servicios de nube publica que van desde la infraestructura, plataforma de desarrollo como servicio, hasta aplicaciones especiales como servicio.

6.2.2. Nube privada.

Se refiere a un modelo de despliegue, en el que empresas generalmente grandes corporaciones que se encuentran en diferentes ubicaciones, ofrecen servicios a la red corporativa para sus propios usuarios internos. Recientes avances en virtualización, y la consolidación de centro de datos, han permitido a los administradores de redes y centros de datos corporativos se conviertan en proveedores de servicios, que satisfagan las necesidades de sus clientes dentro de la corporación.

6.2.3. Nube comunitaria.

En este tipo de nube toda la infraestructura es compartida por varias organizaciones, y brinda soporte a una comunidad en específica, que comparten las mismas preocupaciones, por ejemplo: misión, políticas de seguridad, requisitos, salud pública, etc. Esta puede ser gestionada por las organizaciones, o ya sea por una tercera parte y puede establecerse en las propias instalaciones de la comunidad, grupo o fuera de ellas.

6.2.4. Nube híbrida.

Es un modelo de implementación en el que se combinan características de las nubes públicas y privadas, donde una organización proporciona servicios en la nube, y administra algunos recursos de apoyo en la empresa y tiene otros proporcionados externamente. Arquitectónicamente se considera una nube híbrida, como una nube privada que extiende su límite a un ambiente de nube de terceros, para obtener recursos adicionales de forma segura. (Armando Escalante y Borko Furht, 2010)

6.3. Computación en la nube

6.3.1. Definición.

Según el National Instute of Standards Technology (NIST), se define la computación en la nube o cloud computing, como un modelo que permite el acceso bajo demanda a través de un conjunto compartido de recursos configurables, (redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar rápidamente, con el mínimo esfuerzo de gestión o interacción del proveedor de servicio. (Aguilar, 2012).

Figura 3 Representación de Cloud Computing.

6.3.2. Importancia de la escalabilidad.

Cada vez más, los usuarios demandan de una mayor cantidad de recursos. La característica de escalabilidad, permite expandir fácilmente los recursos para acoplarse a las necesidades de los usuarios.

6.3.3. Características de la computación en la nube.

Según el NIST se compone de 5 características esenciales:

- Autoservicio bajo demanda: un consumidor puede proveerse unilateralmente de tiempo de servidor y almacenamiento en red, a medida que lo necesite, sin requerir interacción humana con el proveedor del servicio.
- Acceso ubicuo a la red: se realiza mediante mecanismos estándares, que promueven el uso por plataforma de clientes ligeros (teléfonos móviles, computadoras portátiles, PDAs, tabletas).
- Distribución de recursos independientes de la posición: los recursos de computación del proveedor son agrupados (“pooled”) para servir a múltiples consumidores, utilizando un modelo multi-distribuido con diferentes recursos físicos y virtuales, asignados y reasignados dinámicamente, conforme a la demanda del consumidor.

- Elasticidad rápida: las funcionalidades se pueden proporcionar, de modo rápido y elástico, en algunos casos automáticamente. Sus características de aprovisionamiento dan la sensación de ser ilimitadas y pueden adquirirse en cualquier cantidad o momento.
- Servicio medido: los sistemas de computación en la nube, controlan y optimizan automáticamente el uso de recursos, potenciando la capacidad de medición, en un nivel de abstracción apropiado al tipo de servicio (almacenamiento, procesamiento, ancho de banda y cuentas activas de usuario). El uso de recursos puede ser monitorizado, controlado e informado, proporcionando transparencia para el proveedor y para el consumidor. (Aguilar, 2012).

6.3.4. Ventajas.

- Se puede acceder a archivos, desde cualquier parte del mundo a cualquier hora que deseemos por Internet.
- La mayoría de los servicios son “Gratuitos”.
- Ofrece gran capacidad de almacenamiento.
- Gran capacidad de computación de datos.
- Escalabilidad.
- Anula los gatos de adquisición de hardware.
- Ahorra los gastos en uso de personal.

6.3.5. Desventajas.

- Los archivos pueden ser hurtados y usados para hacer algún tipo de delito.
- La información puede ser revelada, o pasar datos de la empresa dependiendo de los términos del servicio.
- Para poder tener acceso a los datos se requiere de acceso a Internet.

6.4. Modelos de servicios de computación en la nube

Se refiere a los servicios específicos, a los que se puede acceder en una plataforma de computación en la nube. (Aguilar, 2012).

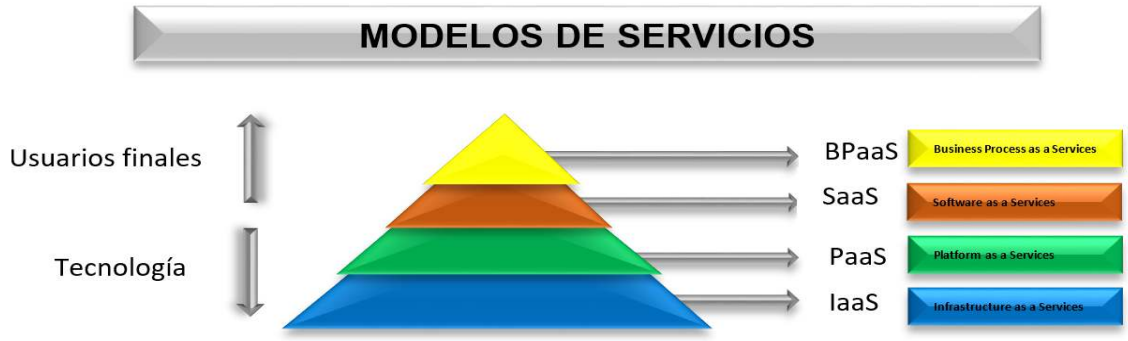

Figura 4 Modelos de servicios

6.4.1. Infraestructura como servicio (IAAS)

En este modelo, el proveedor ofrece utilizar sus recursos informáticos hardware; en otras palabras, el usuario puede comprar los recursos del hardware, procesamiento, almacenamiento, etc. En IAAS es posible aumentar los recursos informáticos para realizar operaciones de mayor rendimiento.

Entre los sistemas más conocidos se encuentra: Amazon Web Service, este ofrece almacenamiento, sistemas de bases de datos, conmutación distribuida etc.

6.4.2. Plataforma como servicio (PAAS)

Los servicios PAAS, proveen desde la nube todos los componentes necesarios, para crear una nueva aplicación informática, ofreciendo un servicio que normalmente integra un entorno de desarrollo y una interfaz de programación de aplicaciones.

Existen diferentes niveles o implementaciones, por ejemplo:

- Amazon (Web Services): ofrece alojamiento y máquinas virtuales.
- Force.com: permite usar cualquier aplicación web empresarial en su plataforma.
- APP Engine: permite alojar aplicaciones web de terceros.

6.4.2.1. ***Ventajas.***

- Escalabilidad garantizada.
- Pago por consumo.
- Desarrollo más sencillo.
- Integración con el resto de la plataforma.
- Administración remota.
- Despliegue transparente.
- Altísima Disponibilidad.

6.4.2.2. ***Desventajas.***

- Herramientas muy limitadas.
- Dependencia con el proveedor.

6.4.3. Software como un servicio (SAAS)

El software como servicio, es una de las modalidades más importantes de la computación en la nube, este consiste en ofrecer una gran variedad de aplicaciones completamente por Internet, y están disponibles normalmente a toda hora por cualquier cliente que desee utilizar la aplicación.

Las aplicaciones son accesibles por una gran variedad de dispositivos, con diferentes sistemas operativos y diferentes hardware, y pueden ser accesibles desde una sencilla interfaz, como lo puede ser un navegador web.

Algunos ejemplos de SAAS:

- Outlook Integration.
- Chatter.
- Mobile Support.
- Gmail.
- Google maps.

6.4.3.1. *Puntos a evaluar para un servicio SAAS.*

Visto desde el punto de vista de un usuario se pueden enumerar los siguientes:

- Fiabilidad: debe disponer de un servicio 24/7, o siquiera aproximado, además de eso debe tener seguridad y confidencialidad con los datos.
- Personalizable: posibilidad de personalizar el servicio, a las necesidades y gustos que poseen los clientes, con el fin de brindar total satisfacción con el cliente.
- Conectividad: poder conectar las aplicaciones de los usuarios, con las aplicaciones que se encuentran en Internet por medio de APIs.
- Precio: el precio tiene que ser menor a lo que el usuario puede invertir, para conseguir el mismo servicio.

6.4.3.2. *Ventajas.*

- Posibilidad de hacer uso de la aplicación, independientemente del hardware del computador solo requiere de un navegador web, con acceso a Internet.
- Los costos son considerablemente bajos.
- Se requiere de un navegador web, con conexión a Internet para disfrutar de las aplicaciones.
- No hay necesidad de piratear (plagiar) la aplicación.
- Hay variedad de servicios.

6.4.3.3. *Desventajas.*

- Perdida de archivos total o temporalmente si no se paga por el servicio.
- Depende de una conexión a Internet y de la capacidad de la red.
- Libertad a la empresa que ofrece los servicios de usar los datos de los usuarios.
- Desconocimiento del paradero fisco de los datos de los usuarios. (Junta de Castilla y León. , 2010).

6.4.4. Bussines Process As A Service (BPaaS)

Es uno de los últimos acrónimos surgidos en el ámbito de computación en la nube, es un modelo de servicio, donde las aplicaciones que se ofrecen en cloud son del tipo procesos de negocio, en este sentido BPaas podría considerarse como el nivel superior de las distintas

capas que se distinguen en la nube, o bien, como un subconjunto o particularización de la capa SaaS.

El modelo de servicio BPaaS necesita, por tanto, de componentes específicos de plataforma para el desarrollo y ejecución de su tipología de aplicaciones. Entre los cuales se pueden mencionar:

- BPM (Business Process Management) / BAM (Business Activity Monitoring), con herramientas de modelado y desarrollo de flujos, motores de ejecución de workflows y monitorización de negocio en tiempo real.
- Componentes de integración de aplicaciones y servicios: EAI (Enterprise Application Integration), ESB (Enterprise Service Bus), adaptadores.
- Componentes de integración de datos: ETL (Extracting, Transformation and Loading), EII (Enterprise Information Integration).
- Componentes de integración de elementos de presentación (portlets, contenidos).

La plataforma con estos componentes específicos, se podría denominar plataforma de soporte al modelo de servicio BPaaS (Guijardo, 2010).

6.5. Patrón de Diseño

6.5.1. Definición

Todas las cosas tienen un patrón, la vida está hecha de patrones, inclusive se observa patrones que se dan en la naturaleza, o las formas de comportarse y reaccionar ante un problema.

Teniendo en cuenta lo anterior, se pensó en usar este mismo concepto en el diseño de aplicaciones, usar patrones que faciliten la solución a problemas informáticos. Los patrones de diseño son técnicas, usada para resolver los problemas de desarrollo de software y están pensados para que tengan cambios, también se puede decir que es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular.

6.5.2. Elementos de un patrón:

- Nombre: describe el problema de diseño.
- El problema: describe cuándo aplicar el patrón.
- La solución: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.
- La consecuencia: es el resultado del aplicar el patrón

6.5.3. Requisitos para ser considerado patrón de diseño.

- Deber ser efectivo, debe haber sido probado anteriormente, resolviendo diferentes problemas similares.
- Debe ser reutilizable, ser aplicado sin ningún problema a circunstancias similares que se presenten.

6.5.4. Historia

El primer artículo que hablaba de patrones en leguajes de programas orientado a objetos, fue publicado en 1987, luego en 1990 se publica el libro “Design Patterns” escrito por el grupo Gang of Four (GOF) compuesto por Erich Gama, Richard Helm, Ralph Johnson y John Vlisides, en este artículo se recogían 23 patrones comunes de software en el que se explicaban detalladamente cómo funcionaba.

El GOF presenta los 23 patrones clasificados según tres propósitos:

1. Creacional

Para poder agregar objetos y no se tenga que cambiar código, tendrá que dejar de depender de los objetos que existan, algunos patrones son Factories, Singleton y Builder.

2. Estructural

Este abarca la manera en la que los objetos, se conectan con los otros objetos, para que al modificar el sistema no se tengan que modificar también las conexiones entre los objetos. El proxy y su adaptador.

3. De Comportamiento

Objetos que contienen tipos particulares de acciones dentro de un programa. Este contiene ejemplos de comandos, estrategias, etc.

Tabla 1 Tipos de patrones

Tipo	Según la clase	Según el objeto
Creación	• Factory Method	• Abstract Factory. • Singleton. • Prototype. • Builder.
Estructura	• Adapter	• Adapter • Bridge. • Composite. • Decorator. • Facade. • Flyweight. • Proxy.
De comportamiento	• Interpreter • Template Method	• Chain of Resposibility. • Command. • Iterator. • Mediator. • Memento. • Obsever. • State. • Strategy. • Template Method. • Visitor.

6.5.5. Definición de Tipos de patrones

1. Abstract Factory

Sirve para crear familias de objetos o dependientes entre sí.

2. Factory Method

Utilidad:

Separar la clase que crea los objetos de la jerarquía de objetos a instanciar.

Ventajas

- Centralización de la creación de objetos.
- Facilita la escalabilidad del sistema.
- El usuario se abstrae de la instancia a crear.

3. Singleton

Utilidad:

Asegurar que una clase tiene una sola instancia y proporcionar un punto de acceso global a ella.

Ventajas:

- Es necesario cuando hay clases que tienen que gestionar de manera centralizada un recurso.
- Una variable global no garantiza que sólo se instancie una vez.

4. Prototype

Permite crear un objeto nuevo, a partir de un prototipo ya que en recurso y tiempo no es fiable empezar desde cero.

5. Builder

Usado para la creación de objetos complejos de un único objeto fuente.

6. Adapter

Permite convertir la interfaz de una clase en otra interfaz esperada por los clientes. Así también permite que clases con interfaces incompatibles se comuniquen.

7. Bridge

Desacopla un método abstracto de su implementación, para que los dos puedan ser modificados independientemente.

8. Composite

Componer objetos en estructuras de árbol para representar jerarquías y permitir a los clientes tratar objetos simples y compuestos de manera uniforme.

Ventajas

- Permite tratamiento uniforme de objetos simples y complejos, así como composiciones recursivas.
- Simplifica el código de los clientes, que sólo usan una interfaz.
- Facilita añadir nuevos componentes, sin afectar a los clientes.

Inconvenientes

- Es difícil restringir los tipos de los hijos.
- Las operaciones de gestión de hijos en los objetos simples pueden presentar problemas: seguridad frente a flexibilidad.

9. Decorator

Permite agregar tareas a un objeto dinámicamente. Proporciona una alternativa flexible a la sub clasificación para extender una funcionalidad.

10. Facade

El patrón FACADE simplifica el acceso a un conjunto de clases, proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases.

Ventajas

- Los clientes no necesitan conocer las clases que hay tras la clase FACADE.
- Se pueden cambiar las clases “ocultadas” sin necesidad de cambiar los clientes. Sólo hay que realizar los cambios necesarios en FACADE.

11. Flyweight

Evita la creación de objetos similares, provocando una mejora en el rendimiento de la aplicación.

12. Proxy

Encargado de proporcionar un objeto que haga de intermediario al momento de servir páginas web, de modo que pueda controlarse el acceso que se tiene a él.

13. Chain of Resposibility

Evita acoplar el emisor de una petición a su receptor, dando la posibilidad de dar a más de un objeto la capacidad de responder.

14. Command

Este encapsula una petición dentro de un objeto.

15. Interpreter

Se encarga de asignar un intérprete para un lenguaje en concreto. El sistema tiene que ser capaz de reconocer sentencias de un lenguaje previamente conocido por su gramática.

16. Iterator

Proporciona una forma de acceder a los elementos de una colección de objetos de manera secuencial, sin revelar su representación interna. Define una interfaz que declara métodos para acceder secuencialmente a la colección.

Ventaja:

La clase que accede a la colección solamente a través de dicho interfaz, permanece independiente de la clase que implementa la interfaz.

17. Mediator

Sirve de mediador para las comunicaciones entre los objetos; este encapsula la comunicación entre los objetos, evitando con ello la necesidad de que lo hagan directamente entre sí.

18. Memento

Se encarga de mantener los diferentes estados que puede tener un objeto, permite la recuperación de algún estado en un tiempo determinado.

19. Obsever

Define una dependencia 1:n de forma que cuando el objeto 1 cambie su estado, los n objetos sean notificados y se actualicen automáticamente. Se utiliza con el objetivo de que en un toolkit de GUI sirva para, separar los objetos de presentación (vistas), de los objetos de datos, de forma que se puedan tener varias vistas sincronizadas de los mismos datos (editor-subscriptor).

20. State

Este es útil cuando el comportamiento del objeto depende del estado en el que se encuentre.

21. Strategy

Encapsula algoritmos relacionados en clases y los hace intercambiables. Se permite que la selección del algoritmo se haga según el objeto que se trate.

Ventajas:

- Se permite cambiar el algoritmo dinámicamente.
- Se eliminan sentencias condicionales para seleccionar el algoritmo deseado.

22. Template Method

Este define la estructura de un algoritmo de operación.

23. Visitor

Este permite añadir funcionalidades a una clase sin tener que modificarla separando así el algoritmo de una estructura de dato. (GoF)

6.5.6. ¿Por qué usar los patrones de diseño?

Los patrones de diseño son soluciones bien pensadas a problemas comunes de programación, no inventar lo que ya está inventado.

Una de las primeras cosas por la cual usar un patrón de diseño es el ahorro de tiempo, debido a que en la programación la mayoría de los casos se ocupa la misma lógica para solucionar un problema, utilizando patrones que se han utilizados en soluciones anteriores, ahorrando el tiempo de pensar y modelar otra solución a ese problema.

Existen dos razones fundamentales:

- Ayudan a validar con seguridad el código:

Estructuras probadas por muchos desarrolladores que han aprobado y mejorado para mejor seguridad y desempeño.

- Entender un lenguaje común:

Ayuda a comprender y mejorar la lógica, de igual forma a comprender como otros desarrolladores han solucionado y pulido soluciones al problema lógico de la programación.

6.6. MVC (Modelo Vista Controlador)

6.6.1. Definición.

Es un patrón de arquitectura de software utilizado en sistemas que requieren separar la lógica de negocio o empresarial de la interfaz de usuario, el cual se divide en 3 partes fundamentales: modelo, vista y controlador. (Simple Example of MVC (Model View Controller) Design Pattern for Abstraction, 2008) La idea principal del patrón MVC es la reutilización de código y la separación de conceptos. (Best MVC Pratices, s.f.)

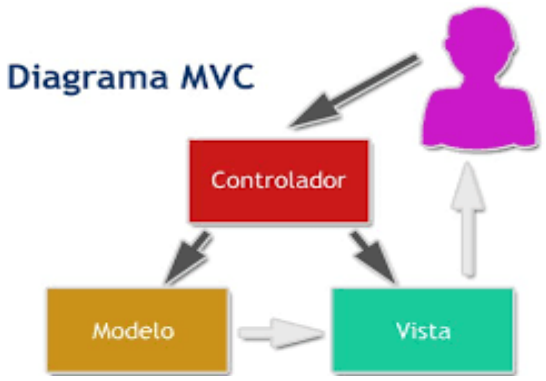

Figura 5 Diagrama MVC

6.6.2. Historia.

El patrón de diseño MVC fue introducido por Trygve Reenskaug en su visita a Xerox Parc en los años 70, su primera aparición fue en Smalltalk-80. El primer documento importante publicado fue "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", realizado por Glenn Krasner y Stephen Pope, publicado entre agosto y septiembre de 1988.

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones. (Model View Controller History, 2014).

6.6.3. Aspectos del patrón MVC.

- HMVC (MVC Jerárquico).
- MVA (Modelo-Vista-Adaptador).
- MVP (Modelo-Vista-Presentador).
- MVVM (Modelo-Vista Vista-Modelo).

6.6.4. Descripción del patrón MVC.

- Modelo

Los modelos son los encargados de manipular los datos en la aplicación: como almacenarlos, cambiarlos, como trabajar con ellos. Una aplicación MVC tiene típicamente un modelo por cada tipo de entidad manipulada por la aplicación, los modelos contienen el código que comunica con el nivel de almacenamiento.

- Vista

Son las presentadas al usuario y contienen información acerca de los modelos con los cuales el usuario puede interactuar, sirven como la interfaz entre los usuarios del sistema y sus datos.

- Controlador

Encargado en mediar la interacción en ambas direcciones, por ejemplo, cuando un usuario interactúa con una vista, una acción específica del controlador correspondiente a la actividad del usuario es invocada. (Armando Fox y David Patterson, 2012).

6.6.5. ¿Cómo funciona?

Al mandar el usuario una petición al navegador el controlador responde a la solicitud del cliente, una vez que el controlador nota lo que el usuario ha solicitado, pide al modelo la información, luego el modelo responde al usuario con los datos que le ha solicitado, y por último se los manda a la vista.

6.6.6. Uso en aplicaciones Web

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitudes de framework, comerciales y no comerciales.

Los primeros framework MVC para desarrollo web planteaban un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. (Wikipedia, s.f.)

6.6.7. Frameworks MVC

Tabla 2 Frameworks MVC

Lenguaje	Licencia	Nombre
.NET	Castle Project	MonoRail
.NET	Apache	Maverick.NET
C++	BSD license	Treefrog
Objective C	Apple	Cocoa
Ruby	MIT	Ruby on Rails
Ruby	MIT	Merb
Ruby	MIT	Rhodes
Java	Apache	Grails
Java	GPL	Interface Java Objects
JavaScript	MIT	Angular JS
Perl	GPL	CGI::Builder
PHP	MIT	Laravel
PHP	GNU/GPL	KumbiaPHP
PHP	MIT	Symfony
PHP	Apache 2.0	Yupp PHP framework
Python	BSD	Django

6.7. APIs Restful

6.7.1. ¿Qué es una API?

El concepto viene del significado de sus ciclas en español “interfaz de programación de aplicaciones”, según explica el científico en computación Benjy Weinbergues, un API es una especificación formal sobre como un módulo de software se comunica o interactúa con otros. (ABC tecnología, 2015)

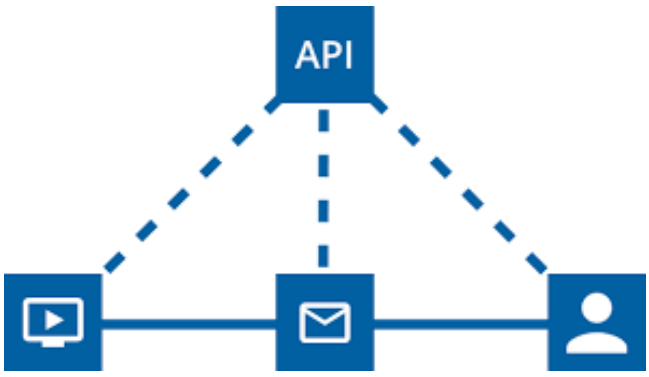

Figura 6 Qué es una API

6.7.2. ¿Qué es REST?

REST es el acrónimo de REpresentational State Transfer. Es un estilo arquitectónico de los sistemas de hipermedia distribuidos y fue presentado por primera vez por Roy Fielding en 2000. REST tiene sus propias restricciones que deben cumplirse si una interfaz necesita ser referida como Restful, las cuales son:

1. **Cliente-Servidor:** al separar las preocupaciones de la interfaz de usuario de las preocupaciones de almacenamiento de datos, mejora la portabilidad de la interfaz de usuario en múltiples plataformas y mejora la escalabilidad al simplificar los componentes del servidor.
2. **Sin estado:** cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender la solicitud y no puede aprovecharse de ningún contexto almacenado en el servidor. Por lo tanto, el estado de la sesión se mantiene completamente en el cliente.
3. **Cacheable:** las restricciones de caché requieren que los datos de una respuesta a una solicitud se etiqueten implícita o explícitamente como cacheables o no. Si una respuesta es almacenable en caché, se le da a la caché del cliente el derecho de reutilizar los datos de las respuestas para solicitudes equivalentes posteriores.
4. **Interfaz uniforme:** se necesitan múltiples restricciones arquitectónicas para guiar el comportamiento de los componentes las cuales son: identificación de recursos, manipulación de recursos a través de representaciones, mensajes auto descriptivos, hipermedia como el motor del estado de la aplicación.
5. **Sistema en capas:** esto permite que una arquitectura se componga de capas jerárquicas al restringir el comportamiento de los componentes, de modo que cada componente no pueda ver más allá de la capa inmediata con la que interactúa. (RESTAPI, s.f.)

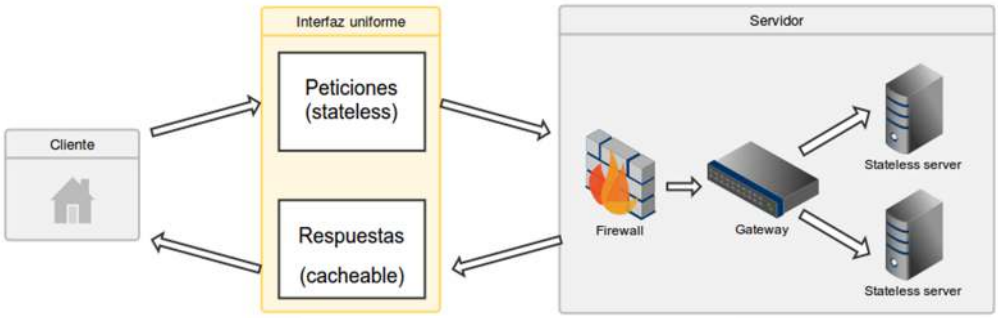

Figura 7 Reglas REST

6.7.3. URI

Los objetos en REST siempre se manipulan a partir de la URI o identificador uniforme de recursos, es una secuencia compacta de caracteres que identifican un recurso abstracto o físico.

Se caracterizan de la siguiente manera:

- **Uniforme:** la uniformidad proporciona varios benéficos, permite diferentes tipos de identificadores de recursos que se utilizan en el mismo contexto, incluso cuando los mecanismos utilizados para acceder a un recurso pueden diferir.
- **Recurso:** el término recurso se usa en un sentido general para lo que puede ser identificado como un URI ejemplo: incluir un documento electrónico, una imagen, etc.
- **Identificador:** incorpora la información requerida para distinguir lo que se identifica de todas las otras cosas dentro de su alcance de identificación. (Network working group, 2005)

Una solicitud http está compuesta por un método http y una URI, una URI completa inicia con el nombre de un esquema de comunicación, seguida del nombre del host, puerto y el recurso solicitado. Un recurso generalmente significa algo que puede ser entregado al navegador, por ejemplo: una imagen, una lista de películas en formato html, etc. (Armando Fox y David Patterson, 2012)

Figura 8 Componentes de URI

6.8. Lenguaje de Programación

Es el lenguaje que puede controlar el comportamiento de una máquina, está compuesto por un conjunto de reglas semánticas y sintácticas, una forma práctica en el que una persona puede dar instrucciones a una máquina.

Grupo de instrucciones de un lenguaje de programación:

- Instrucciones de entrada-salida.
- Instrucciones aritméticas-lógicas.
- Instrucciones selectivas.
- Instrucciones repetitivas.
- Niveles de lenguaje, según la abstracción en la que se encuentra las sentencias:
- Lenguaje máquina.
- Lenguaje de bajo nivel.
- Lenguaje de alto nivel.

El lenguaje máquina habla el mismo idioma de los computadores, pero los demás lenguajes no, para ello se deben traducir las instrucciones a un lenguaje que pueda entender la máquina.

6.8.1. Traductores a lenguaje máquina.

- Compiladores: este traduce a un lenguaje inferior.
- Intérpretes: este hace lo mismo que los compiladores con la diferencia que este la realiza línea a línea.

Una de las ventajas del intérprete es la portabilidad, ya que permite que el programa pueda ser traslado a otra plataforma, mientras que el compilador solo podrá funcionar en la plataforma que se compiló.

6.9. Lenguaje Ruby

6.9.1. Definición.

Figura 9 Logo de Ruby

Es un lenguaje de programación interpretado con capacidades de llamadas directas al sistema operativo y facilidades de desarrollo; se considera como un lenguaje orientado a objetos.

Yukihiro Matsumoto a menudo manifiesta que está "tratando de hacer que Ruby sea natural, no simple". También agrega "Ruby es simple en apariencia, pero completo por dentro, como el cuerpo humano".

Según (TIOBE, 2018) que es el encargado de medir los lenguajes de programación a nivel mundial ubica al lenguaje Ruby en la posición número 10 del ranking, gran parte de su crecimiento se atribuye a la popularidad alcanzada por aplicaciones desarrolladas con Ruby, en particular el framework de desarrollo web Ruby on Rails. (Ruby, s.f)

Tabla 3 Lenguaje más utilizados

Posición	Lenguaje
1	Java
2	C
3	C++
4	Python
5	C#
6	Visual Basic .NET
7	PHP
8	JavaScript
9	Delphi/Object Pascal
10	Ruby

6.9.2. Historia

Fue creado por el japonés Yukihiro Matsumoto entre el año 1993 y 1995, a partir de sus lenguajes preferidos: Perl, Smalltak, Eiffel y Lisp. Luego se presenta públicamente en su versión 0.95, al ser la primera versión era escaso en funcionalidad, pero gracias al esfuerzo se ofrece después la versión de Ruby 1.0.

En 1997 empiezan las empresas a interesarse en Ruby, en este mismo año se escribe un primer artículo sobre el lenguaje, años después IBM se interesa en el lenguaje y publica el artículo Latest Open Source gem from Japan.

El crecimiento en un principio fue lento, pero siempre ha crecido a paso firme, hasta que en el 2004 el framework de Rails fue liberado en la versión de Rails 1.0.

6.9.3. Ruby Lenguaje Multiplataforma

Es un lenguaje multiplataforma, es decir, puede correr en una gran cantidad de dispositivos de diferentes arquitecturas, entre las que se tienen:

- Microsoft Windows.
- Mac OS.
- GNU/Linux.
- MS-DOS.
- Unix.

Es un lenguaje extensible gracias a las librerías que se le pueden agregar; no solamente escritas en lenguaje Ruby, sino también en C y otras más, además Ruby permite que se puedan incorporar funcionalidades en las clases bases. En resumen, en Ruby se pueden hacer modificaciones en todo lo que se quiera en el lenguaje.

De igual forma en Ruby se puede programar diferentes cosas, pero en las que más tiene auge es en el desarrollo de aplicaciones web.

¿Por qué alguien que ya está familiarizado con Perl o Python podría cambiarse a Ruby?

Si está satisfecho con Perl o Python no es necesario. Pero si siente que debe haber un lenguaje mejor; Ruby puede ser el idioma de elección.

Aprender un nuevo idioma es inofensivo; da nuevas ideas y puntos de vista, no tiene que cambiar solo se aprende y se prueba. (Stewart, 2001)

6.9.4. Programar en Ruby

Viendo todo como un objeto. Yukihiro Matsumoto dijo: “quería un lenguaje que fuera más poderoso que Perl, y más orientado a objetos que Python”.

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. La orientación a objetos pura de Ruby se suele demostrar con un simple ejemplo que aplica una acción a un número.

```
5.times {print "Ruby todo es un objeto!"}
```

En otros 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. (Ruby, s.f)

6.9.5. Ruby un lenguaje Flexible

Ruby se considera un lenguaje flexible, porque permite a sus usuarios alterarlo libremente. Se puede agregar funcionalidades nuevas o redefinir las que ya están predefinidas. Lo que intenta Ruby es no limitar al desarrollador.

Por ejemplo, se puede agregar un método llamado restar a la clase número.

```
Class numero
  def restar(x)
    Self.-(x)
  end
end
y = 5.resta 6
```

Ruby no necesita que se declare la variable. Se utilizan convenciones simples para nombrar y determinar el alcance de la misma.

- x puede ser una variable local.
- @x variable de instancias.
- $x variable de tipo global.

Esto ayuda a identificar fácilmente los roles de las variables. (Ruby, s.f)

6.10. Embedded Ruby (ERB, eRuby)

Es un lenguaje de plantilla basado en Ruby, escrito por Seky Masatoshi; una plantilla ERB se parece a un documento de texto plano intercalado con etiquetas que contienen el código Ruby. Cuando se evalúa, este código etiquetado puede modificar texto en la plantilla.

6.10.1. Etiquetas

ERB tiene 2 etiquetas para el código Ruby, una etiqueta para comentarios y una forma de los delimitadores de etiquetas.

- <%= Expression %> Inserta el valor de una expresión.
- <% %> Ejecuta el código, pero no inserta valores.
- <%# Comentario %> Utilizada para hacer comentario.
- <%% or %%> Delimitador de etiquetas literales.

El texto fuera de una etiqueta se convierte en texto literal, pero está sujeto a cualquier código Ruby etiquetado que lo rodee. Ejemplo: si el texto está rodeado por una instrucción if etiquetada solo aparece en el resultado si la condición es verdadera. (puppet, s.f.)

```
<% titulo_pagina = " Demostración lenguaje ERB" %>
<% saludo = "hola estudiante" %>

<html>

    <head>
    <title> <%= titulo_pagina %> </title>
     </head>

     <body>
            <p> <%= saludo %> </p>
            <p> Este es un ejemplo de cómo utilizar ERB</p>

     </body>
 </html>
```

6.11. Ruby on Rails

6.11.1. Definición.

Es un Framework de desarrollo de aplicaciones web escrito en lenguaje Ruby, de código abierto, permite escribir menos código mientras se logra más que muchos otros lenguajes, es un software obstinado. Supone que existe una mejor manera de hacer las cosas y está diseñado para alentar de esa manera. (Rails, s,f)

Figura 10 Logo de Rails

6.11.2. Historia

El Framework de Rails fue creado por David Heinemeier Hansson conocido como DHH, programador Danés. El framework fue desarrollado como herramienta de David para facilitarle el trabajo de la creación de la aplicación Web Basecamp, proyecto de la empresa 37 Signals. En el año 2004 David liberó el código declarándolo como software libre y luego en el año 2005 empieza aceptar colaboraciones para la mejora del Framework Rails. (Hansson, 2006)

Sigue el paradigma MVC (Modelo Vista Controlador), este permite configurar ciertas cosas del controlador de manera libre para adaptarlo y permitir manejar información de bases de datos por medio de consultas ActiveRecord en base a atributos o campos de modelos ya creados.

6.11.3. Principios básicos de Rails

- DRY (Don´t Repeat Yourself):

Es un principio de desarrollo de software que establece que “cada pieza de conocimiento debe tener una representación única, inequívoca y autorizada dentro de un sistema”. Al no escribir la misma información una y otra vez, el código es más fácil de mantener, más entendible y menos problemático.

- Convención sobre configuración:

Rails tiene opiniones sobre la mejor manera de hacer muchas cosas en una aplicación web, y se predetermina a este conjunto de convenciones, en lugar de exigirle que especifique las minucias a través de archivos de configuración infinitos. (Rails, s,f)

6.11.4. Gemas en RoR.

Las gemas son librerías que se incorporan a las aplicaciones desarrolladas en Rails, dichas gemas o también conocidas como librerías son contribuciones de usuarios externos, cada gema tiene un nombre y versión, y se pueden usar para ampliar o modificar la funcionalidad en las aplicaciones de Ruby.

Figura 11 Logo de RubyGem

6.11.4.1. *RubyGem.*

Es el sistema que utiliza el framework para manejar librerías o gemas contribuidas por usuarios externos. El RubyGem lo instala el framework de manera predeterminada. (Armando Fox y David Patterson, 2012).

6.11.4.2. *Gemfile.*

Es uno de los principales archivos que utiliza el framework, los archivos de Gem requieren al menos una fuente de gemas en forma de URL para un servidor de RubyGem, en este archivo es donde se establecen las gemas que necesita la aplicación para su funcionamiento, el framework al momento de crear un proyecto instala distintas gemas por defecto, entre las cuales se puede mencionar sqlite3 que es el gestor de base de datos que utiliza a modo de prueba.

Para agregar alguna gema a un proyecto en Rails sólo se debe agregar el nombre y la versión de la gema al archivo Gemfile del proyecto.

Luego con el siguiente comando se instalarán las gemas.

```
$ bundle install
```

Si en un determinado caso bundle no se encuentra instalado ejecutar el comando.

```
$ gem install bundler
```

6.11.4.3. *Gemas de uso común para el desarrollo de aplicaciones*

Cualquier cosa es buena para facilitar el trabajo a un desarrollador. Sin embargo, es imprescindible contar con las mejores herramientas que lo simplifique y que lo hagan más fácil. En la programación con Ruby on Rails estás son algunas de las gemas más comunes que se pueden emplear:

- Devise

Es la gema por excelencia para gestionar las sesiones de usuarios en Ruby on Rails. Los desarrolladores la definen como muy fácil de utilizar gracias a sus detalladas instrucciones.

- Rspec-Rails

Es un Framework de testing que promueve el desarrollo guiado por comportamiento (BDD) y ofrece sintaxis muy orientada a ese paradigma.

- Paperclip

Sirve principalmente para la gestión de archivos en una página web o entorno. Posibilita una gestión muy eficiente sobre los ficheros y sobre los propios modelos. Posee una serie de funcionalidades para la gestión de video, imagen y sonido.

- Resque

Sirve para controlar procesos en background. Da la posibilidad de lanzar procesos asíncronamente al proceso real de la aplicación; la integración que ofrece esta gema con Rails es muy práctica puesto que por un lado permite añadir procesos a la cola de procesos, por ejemplo: enviar un email, procesar una imagen en segundo plano o hacer consultas a un servicio externo que tarda mucho.

- Capistrano

Permite desplegar aplicaciones en entornos de servidor, automatiza los procesos de actualización de código o arranque de servidor. Está escrita en Ruby, pero puede utilizase en otro tipo de aplicaciones.

- Doorkeeper

Se utiliza en entornos de APIs y de la facultad de añadir a una aplicación Rails un proveedor Oauth. Al incluir este protocolo permite mantener una sesión desde una API y que se le otorgue una sesión.

- Grape

Es un Framework especial para la creación de APIs. Grape permite definir de manera muy clara todos los requisitos de un servicio específico de API.

- Cancancan

Es una biblioteca de Ruby que sirve para restringir a que recursos tiene acceso un usuario determinado. Todos los permisos se pueden definir en uno o varios archivos, y no pueden ser duplicados a través de controladores, vistas, consultas de base de datos.

- Rubycritic

Es una herramienta para medir la calidad del código que se está escribiendo y generando, es una macrogema que integra varias herramientas de revisión de código y de análisis estático del mismo para ver su calidad, complejidad, la redundancia, cuántas veces se ha utilizado una función, si hay funciones que queden en desuso o que defectos pueden tener.

- Dotenv-rails

Ayuda con el manejo de las variables globales de entorno a nivel de la aplicación, especialmente aquellas que el programador no quiere tener dentro de su código. (Lucia El Asri, 2016)

- Mysql2

Está diseñada para servir en caso de una conexión, consulta e iteración de los resultados, esta gema se vincula con la librería libmysqlclient de MySQL y con alternativas compatibles como MariaDB.

6.11.5. Uso de Rails

El uso de Rails es considerablemente sencillo. Para poder empezar a usar Rails basta con seguir estos sencillos pasos y así crear una primera aplicación web:

- Abrir el terminal y escribir lo siguiente:

```
$ rails new NombreProyecto
```

Se crea una carpeta con todos los archivos necesarios para que pueda correr dicha aplicación. En donde se tendrá que escribir “NombreProyecto”, que es el nombre que se le dará al proyecto se podrá cambiar por el nombre que se desee dar a su proyecto.

- Acceder a esta carpeta. Dentro del terminal se escribirá:

```
$ cd NombreProyecto
```

- Iniciar el servidor de Rails.

```
$ rails server
```

Para asegurarse que la aplicación está funcionando habrá que ubicarse en el navegador web y escribir en la barra de direcciones: 127.0.0.1:3000 o bien localhost:3000 que hace referencia a la misma dirección local del servidor que se utiliza, se observará acá la página principal del proyecto web creado recientemente.

6.11.5.1. *Jerarquía de directorios de un proyecto en rails*

Como se ha mencionado anteriormente, al crear un nuevo proyecto en Rails se crean un conjunto de directorios y archivos necesarios para su correcta ejecución, en la siguiente tabla se muestra lo que se conoce como la jerarquía de directorios de un proyecto en Rails.

Tabla 4 Jerarquía de un proyecto en RoR

Directorio	Descripción
app/	Se encuentra todo lo relacionado a los modelos, vistas, controladores y helpers que son partes de la aplicación.
bin/	Contiene un conjunto de scripts que porta el framework.
config/	Se encuentran los archivos de configuración de las rutas, base de datos, etc.
db/	Almacena el esquema actual de la base de datos.
Gemfile	Uno de los archivos más importantes del proyecto, este es el archivo de gemas encargado de especificar las dependencias de la aplicación rails.
lib/	Carpeta donde se incluyen módulos que extienden la aplicación.
log/	Contiene los ficheros de registros del servidor.
public/	Directorio de donde se sirven los archivos estáticos y los recursos compilados.
Rakefile	Sirve para localizar las tareas que se pueden ejecutar desde la línea de comandos.
README.md	Contiene una pequeña descripción de la aplicación.
test/	Test unitarios de la aplicación.
tmp/	Archivos temporales como caché, archivos pid del sistema, etc.
vendor/	Carpeta donde se introducir el código de terceros a la aplicación.

6.12. Datos en Rails

6.12.1. ¿Qué es Active Record?

Active Record es la M en MVC, el modelo, el cual es la capa del sistema responsable de representar los datos y la lógica de negocio para manipularlos. Active Record facilita la creación y manipulación de objetos de negocio quienes requieren ser almacenados persistentemente en una base de datos. Active Record fue descrito por Martin fowler en su libro Patterns of Enterprise Application Architecture, los objetos soportan tanto la persistencia y el comportamiento que opera con los datos.

- Mapeo de objetos Relacionales

Comúnmente nombrado por sus siglas ORM, es una técnica que conecta la riqueza de los objetos de una aplicación con las tablas de un sistema de base de datos relacional. Las propiedades y las relaciones de un objeto en una aplicación pueden ser fácilmente guardadas y recuperadas desde la base de datos sin escribir sentencias SQL directamente con el mínimo código en general de acceso a la base de datos.

Active Record brinda varios mecanismos en relación a los modelos; los más importantes dan la capacidad para:

1. Representar modelos y sus datos.
2. Representar asociaciones entre esos modelos.
3. Representar jerarquías de herencia a través de modelos relacionados.
4. Validar modelos antes de que sean guardados o cambiados en la base de datos.
5. Mantener las operaciones de la base de datos orientadas a objetos.

6.12.2. Convención sobre configuración

Active Record sigue las convenciones adoptadas por Rails; por tanto, no se necesitan hacer muchas configuraciones cuando se crea un modelo.

- Convenciones sobre los nombres

Rails convertirá al plural los nombres de las clases para encontrar la respectiva tabla en la base de datos, por ejemplo: para una clase book, debe haber una tabla de base de datos llamada books. Los mecanismos de pluralizar de Rails tienen la capacidad de pluralizar y singularizar ambos en palabras regulares e irregulares. Cuando se utiliza nombres de clases compuestos de dos o más palabras, el nombre de la clase del modelo debería seguir las convenciones de Ruby usando la forma CamelCase, mientras que el nombre de la tabla debe contener las palabras separadas por guion bajo.

6.12.3. CRUD

Es el acrónimo de los cuatro verbos que se utilizan para trabajar con los datos: Create, Read, Update, y Delete. Active Record automáticamente crea métodos que permiten a una aplicación leer y manipular los datos guardados dentro de las tablas.

- Create

Los objetos Active Record pueden crearse desde un hash, un bloque o configurar manualmente antes de la creación. El método new retornará un nuevo objeto mientras create retornará el objeto y lo guardará en la base de datos.

- Read

Active Record provee varios métodos utilizados para acceder a los datos dentro de la base de datos entre los cuales están los métodos: all, first, last, find_by, etc.

- Update

Una vez que un objeto Active Record ha sido recuperado, sus atributos pueden ser modificados y volver a ser guardados en la base de datos.

- Delete

De igual forma una vez recuperado el objeto Active Record también puede ser destruido, lo cual lo borrará de la base de datos. (RailsGuide_Active_basic, s.f.)

6.12.4. Migraciones.

Rails provee un lenguaje de dominio específico para manejar un esquema de base de datos llamado migraciones, son ficheros guardados que se ejecutan contra cualquier base de datos que Active Record soporte utilizando rake, Rails mantiene el historial sobre que fichero fue actualizado en la base de datos y provee características para deshacer los cambios. Para crear una tabla sólo se debe ejecutar el comando: rake db:migrate y para deshacerlo: rake db:rollback.

Las migraciones son una forma conveniente de modificar el esquema de la base de datos a lo largo del tiempo de manera consistente y sencilla. Usa DSL de Ruby para que no tenga que escribir SQL a mano, lo que permite que su esquema y sus cambios sean independientes de la base de datos.

Se puede pensar que cada migración es como una nueva versión de la base de datos. Un esquema comienza con nada en él, y cada migración lo modifica para agregar o eliminar tablas, columnas o entradas. Active Record puede actualizar su archivo db/schema.rb para que coincida con la estructura actualizada de la base de datos. (Rails, s.f.)

6.12.5. Validaciones de Active Record.

Las validaciones a nivel de modelo son la mejor manera de garantizar que solo se guarden datos válidos en su base de datos. Son independientes de la base de datos, no pueden pasar por alto los usuarios finales y son convenientes para probar y mantener. Rails los hace fáciles de usar, proporciona ayuda incorporada para las necesidades comunes y también le permite crear sus propios métodos de validación, por ejemplo: puede ser importante para una aplicación asegurarse de que cada usuario proporcione una dirección de correo electrónico y una dirección postal válidas.

```
class Person < ApplicationRecord
   validates :name, presence: true
end
```

Existen otras formas de validar datos antes de que se guarden en su base de datos. Pros y Contras de estas formas:

- Las restricciones de la base de datos y / o los procedimientos almacenados hacen que los mecanismos de validación dependan de la base de datos y pueden dificultar las pruebas y el mantenimiento. Sin embargo, si su base de datos es utilizada por otras aplicaciones, puede ser una buena idea usar algunas restricciones en el nivel de la base de datos. Además, las validaciones a nivel de base de datos pueden manejar de manera segura algunas cosas (como la singularidad en tablas muy utilizadas) que de otro modo pueden ser difíciles de implementar.
- Las validaciones del lado del cliente pueden ser útiles, pero generalmente no son confiables si se usan solas. Si se implementan usando JavaScript, pueden omitirse si JavaScript se desactiva en el navegador del usuario. Sin embargo, si se combina con otras técnicas, la validación del lado del cliente puede ser una forma conveniente de proporcionar a los usuarios comentarios inmediatos a medida que utilizan el sitio.
- Las validaciones a nivel de controlador pueden ser tentadores de usar, pero a menudo resultan poco manejables y difíciles de probar y mantener. Siempre que sea posible, es una buena idea mantener los controladores delgados, ya que hará que la aplicación sea un placer trabajar a largo plazo.

Se pueden elegir estos en ciertos casos específicos. Según la opinión del equipo de Rails las validaciones a nivel de modelo son las más apropiadas en la mayoría de las circunstancias. (Rails , s.f.)

6.12.6. Rutas en Rails.

El enrutador de Rails reconoce las URL y las envía a la acción de un controlador. También puede generar rutas y URL, evitando la necesidad de codificar cadenas de caracteres en sus vistas.

1. Conexiones

- Cuando una aplicación Rails recibe una solicitud entrante.

```
GET /patients/17
```

- Le pide al enrutador que coincida con una acción del controlador. Si coincide es:

```
get '/patients/:id', to: 'patients#show'
```

Con esto, la solicitud se envía a la acción show del controlador patients con { id: '17' } como parámetro.

2. Generar rutas y URL desde el código

- El archivo routes.rb encargado de las rutas en una aplicación está ubicado en el directorio config. Dentro de este se pueden crear rutas y URL para distintas acciones de la aplicación.

```
get '/patients/:id', to: 'patients#show', as: 'patient'
```

- El controlador de la aplicación debe contener lo siguiente:

```
@patient = Patient.find(17)
```

- Y la vista correspondiente debe contener un enlace.

```
<%= link_to 'Patient Record', patient_path(@patient) %>
```

3. Recursos

Los navegadores solicitan páginas de Rails haciendo solicitudes, usando un método específico de http como: GET, POST, PATCH, PUT, o DELETE. Una ruta de recursos mapea una cantidad de solicitudes relacionadas a acciones en un solo controlador.

- Cuando la aplicación de Rails recibe una solicitud.

```
DELETE /photos/17
```

- Le pide al enrutador que lo asigne a una acción de controlador.

```
resources :photos
```

Rails enviará esa solicitud a la acción destroy del controlador photos con {id: 17} como parámetro.

4. CRUD, verbos y acciones

En Rails, una ruta realiza una asignación entre los verbos HTTP y las URL a las acciones del controlador. Cada acción también asigna una operación CRUD específica en una base de datos. Una sola línea de código en el archivo routes.rb, crea siete diferentes rutas en una aplicación ejemplo:

```
resources :photos
```

Tabla 5 Rutas y verbos http

Verbo http	Ruta	Controlador#Accion	Uso
GET	/photos	photos#index	Muestra una lista de todas las fotos
GET	/photos/new	photos#new	Retorna un formulario html para crear una nueva foto
POST	/photos	photos#create	Crea una nueva foto
GET	/photos/:id	photos#show	Muestra una foto en específico.
GET	/photos/:id/edit	photos#edit	Retorna un formulario html para editar la foto
PATCH/ PUT	/photos/:id	photos#update	Actualiza una foto en específico
DELETE	/photos/:id	photos#destroy	Elimina una foto

5. Ruta root

En Rails se puede especificar cuál debe ser la ruta raíz de la aplicación con el método root. Ejemplo:

```
root to: 'pages#main'

root 'pages#main' # otra manera
```

Esta ruta debe estar siempre al inicio del archivo routes.rb. (Rails)

6.13. Técnicas de desarrollo ágil

6.13.1. Test Driver Development (TDD).

Desarrollo en base a pruebas es una técnica de desarrollo en la cual se escriben test antes de escribir código, se centra básicamente en escribir un pequeño test para un código que no existe todavía, al correr el test va a fallar naturalmente, luego escribir el código necesario para que el test pase, una vez que el test pasa es importante examinar el resultado y reescribir código que este duplicado.

En muchos sistemas de test, cuando el test falla se observa el resultado en rojo, luego cuando el test pasa el resultado se imprime en verde, a esto es que se refiere el ciclo red/green/refactor. (Chelinsky, 2010)

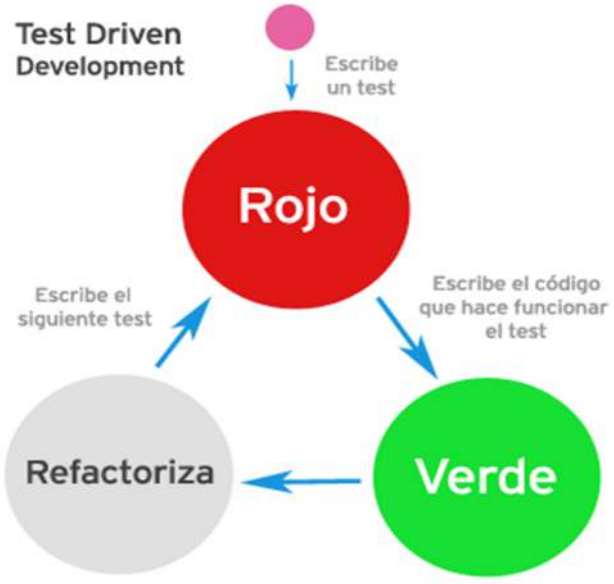

Figura 12 Proceso TDD

6.13.1.1. *Ventajas.*

- Ayuda a pensar en cómo se quiere desarrollar la funcionalidad.
- Creación software modular y flexible.
- Minimizar la necesidad de un “debugger”.
- Aumentar la confianza del desarrollador a la hora de introducir cambios en la aplicación.

6.13.1.2. *Desventajas.*

- Dificultades al momento de aprobar situaciones en las que son necesarios test funcionales o de integración, como pueden ser bases de datos o interfaces de usuario.
- A veces se crean test innecesarios que provocan una falsa sensación de seguridad, cuando en realidad no están probando más que al hecho de que un método haga lo que dice que hace.
- Los test requieren de mantenimiento, a la vez que se mantiene el código, lo cual genera un trabajo extra.
- Es difícil introducir TDD en proyectos que no han sido desarrollados desde el principio con TDD.
- A veces el desarrollo se centra más en cómo construir una funcionalidad que en preguntarse si la funcionalidad es verdaderamente necesaria para el usuario o es como la quería el usuario.

6.13.2. Behavior Driver Development (BDD).

Conocido también como desarrollo guiado por comportamiento, surge a través de TDD, la idea principal es proporcionar a los equipos de desarrollo y administración un proceso de herramientas compartidas para que puedan colaborarse eficazmente mientras se desarrolla un software. En 2009, Dan North dio la definición más aceptada para BDD:

"BDD es una metodología ágil de segunda generación, de afuera hacia adentro, basada en tirar, de múltiples partes interesadas, de escala múltiple y alta automatización. Describe un ciclo de interacciones con resultados bien definidos, lo que resulta en la entrega de software probado que importa".

Para trabajar con BDD se utiliza DSL que es el lenguaje específico de dominio; este lenguaje utiliza palabras normales en inglés y construcciones lógicas para expresar como debe comportarse el software.

```
describe MovieList do
  context " when first creates "
    it "is empty" do
      movie_list = MovieList.new
      movie_list.should be_empty
    end
  end
end
```

BDD sigue el principio básico de que cada unidad de software debe probarse individualmente. El proceso es:

- Una prueba está diseñada para una unidad específica de software.
- La prueba es realizada para fallar.
- La unidad es luego implementada dentro del test.

El test es hecho de nuevo, verificando que la implementación de la unidad lo hace exitosamente. (Steve Fox, 2016)

Capítulo III:

Diseño metodológico.

7. Diseño Metodológico

El presente trabajo monográfico tiene un enfoque práctico, la teoría proporcionada está basada a la comprensión de conceptos fundamentales para el desarrollo satisfactorio de las prácticas de laboratorio, por ende, la investigación realizada es de carácter aplicada.

7.1. Etapas de la investigación

7.2. Recolección de información

En la primera etapa de la investigación, se ha realizado una lectura no específica de documentaciones sobre Software como un servicio y el framework Ruby on Rails en búsquedas de antecedentes.

Se seleccionó las fuentes que se tomarían como primarias y los temas a desarrollar en cada una de las guías de laboratorio, organizándolas según su nivel de complejidad y de las competencias que se pretende sean adquiridas por los estudiantes a lo largo del curso de manera sistemática.

7.3. Selección de herramientas

En esta etapa, se seleccionaron las distintas herramientas (software) a utilizar en el desarrollo de las guías; basándose en su capacidad, flexibilidad y funcionamiento, buscando estabilidad se ha elegido una versión específica para cada una. Cabe destacar que, para el desarrollo de las guías de laboratorio, a los estudiantes se les facilitará una máquina virtual exportada con todas estas herramientas software instalados, para que se le facilite aún más la rápida adquisición de las competencias que busca la microprogramación , la cual se puede descargar del siguiente enlace https://goo.gl/7m7sBm

Se ha optado por los siguientes softwares y versiones:

- **Virtualizador**
    - Oracle MV VirtualBox v5.1.18
- **Sistema operativo huésped**
    - Debian 9
- **Ruby on Rails**
    - Ruby v 2.4.1
    - Rails v 5.1.4
- **Editor de texto**
    - **S**ublimetext3

➢ **Las credenciales de la máquina virtual son las siguientes:**

Usuario: debian
Contraseña: ruby1

7.4. Elaboración y redacción de las guías

Una vez con las fuentes de información definidas y los temas organizados según su relevancia y dificultad, se prosiguió al desarrollo de las guías con sus respectivas soluciones; este segmento está divido en 4 bloques dentro los cuales se abordan los siguientes temas:

Bloque 1: lenguaje Ruby

- Ejercicios String y enteros
- Definición de métodos, uso de métodos.
- Definición de hash, clases.

Bloque 2: introducción al framework

- Comando básicos Ruby on Rails.
- Jerarquía de directorios.
- Configuración de archivos.
- Patrón de diseño MVC.

Bloque 3: lenguaje e-Ruby

- Etiquetas <%= %>, <% %>
- Creación proyectos, utilizando técnicas de desarrollo ágil.
- Creación de estilos.
- Integración de bootstrap.

Bloque 4: propuestas de proyectos prácticos.

- Proyecto desarrollo de una aplicación al estilo instagram.
- Proyecto blog.

7.5. Formato de guías

Para la elaboración de las guías se ha definido un formato, en la cual están basadas cada una de las guías presentadas en este trabajo.

Título

- Nombre de la guía.

Introducción

- Una pequeña descripción de lo que se pretende realizar en la guía.

Objetivos

- Valoración general de los que se espera lograr en el desarrollo de cada una de las guías.

Tiempo

- Tiempo estimado para la elaboración de la práctica.

Requerimientos

- Características de hardware y software necesarios para la elaboración.

Referencia

- Información de apoyo en la elaboración de las guías.

Desarrollo

- Se explicará de manera detallada los pasos a seguir para la realización de las guías.

Ejercicios Propuestos entregados al docente

- Propuestas relacionadas con las actividades que el estudiante ha realizado anteriormente en la parte del desarrollo de la guía.

7.6. Detalle de horas para la elaboración de las guías

En la siguiente tabla se muestra un detalle de las horas y sesiones de clases en que se divide la elaboración de cada una de las guías.

Tabla 6 Detalles de las guías

Trabajo	Título	Horas	Sesiones
Guía 1	Introducción lenguaje Ruby (parte I)	**2**	**1**
Guía 2	Introducción al lenguaje Ruby (parte II)	**2**	**1**
Guía 3	Trabajando con Rspec y Ruby	**2**	**1**
Guía 4	Introducción al framework RoR	**2**	**1**
Guía 5	Incorporación de código Ruby dentro del framework Rails	**2**	**1**
Guía 6	Patrón de diseño MVC usando scaffold	**2**	**1**
Guía 7	Ruby on Rails con MYSQL	**2**	**1**
Guía 8	Ruby on Rails con TDD	**2**	**1**
Guía 9	Integración bootstrap en RoR	**2**	**1**
Proyecto 1	Desarrollo de una aplicación al estilo instagram	**6**	**3**
Proyecto 2	Blog	**6**	**3**
Total horas		**30**	**15**

7.7. Línea de trabajo

Una vez redactadas todas las guías y proyectos, se decidió realizar una clasificación de estas, en dos grupos: guías complementarias y guías base, esto con la intención que en situaciones donde sea afectado los encuentros de clases ya sea por feriados u otros escenarios, el docente pueda modificar la secuencia de estas.

Guías complementarias: Son todas las guías en que los temas abordados no son nuevos para los estudiantes, son temas o principios básicos que los estudiantes han venido aprendiendo en otros componentes y lenguajes de programación, por lo que se podrían asignar como auto estudio o para ser resueltas en horas no presenciales.

Por lo que se considera que el docente podría enfocarse en aquellos temas que los estudiantes desconocen por ser tan particulares de Ruby o del framework Ruby on Rails los cuales por su relevancia están dentro del grupo de **guías bases.**

Según lo antes mencionado, la clasificación resultante queda como se muestra a continuación.

Tabla 7 Línea de trabajo

Trabajo	Título	Clasificación
Guía 1	Introducción lenguaje Ruby (parte I)	Complementaria
Guía 2	Introducción al lenguaje Ruby (parte II)	Complementaria
Guía 3	Trabajando con Rspec y Ruby	**Base**
Guía 4	Introducción al framework RoR	**Base**
Guía 5	Incorporación de código Ruby dentro del framework Rails	**Base**
Guía 6	Patrón de diseño MVC usando scaffold	Complementaria
Guía 7	Ruby on Rails con MYSQL	**Base**
Guía 8	Ruby on Rails con TDD	**Base**
Guía 9	Integración bootstrap en RoR	Complementaria
Proyecto 1	Desarrollo de una aplicación al estilo instagram	**Base**
Proyecto 2	Blog	Complementaria

Cabe destacar que, aunque una guía sea considerada complementaria no es porque no aporte suficientes conocimientos; sino, es para adaptar el presente trabajo a la realidad de la Universidad en cuanto a pérdida de fechas de clases y por la naturaleza de las guías de ser guiadas, perfectamente pueden ser resueltas en horas no presenciales apoyándose en las referencias y ejemplos plasmados en cada una de estas.

Capítulo IV:

Desarrollo guías

Guía 1: Introducción al lenguaje Ruby (parte I)

Introducción

Ruby es un lenguaje de programación orientado a objetos de código abierto, reconocido por ser un lenguaje rápido y sencillo. Su creador Yukihiro "Matz" Matsumo mezcló sus lenguajes favoritos (Perl, Smalltalk, Eiffel, Ada y Lisp) para formar un nuevo lenguaje que incorporara lo mejor de cada uno.

En la presente guía el estudiante aprenderá un poco sobre el lenguaje Ruby, podrá trabajar con variables String, números; métodos relacionados con String y cómo se comportan estos objetos dentro del lenguaje.

Objetivos

- Aprender a declarar variables en Ruby.
- Conocer el funcionamiento de los métodos del lenguaje Ruby.
- Distinguir los tipos de objetos que contiene el lenguaje Ruby.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria RAM mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- RubySur. Aprende a programar. rubysur.org/aprende.a.programar/
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/quickstart/

Desarrollo

1.String.

1.1. Crear un directorio llamado ruby, donde se almacenarán los ejercicios que se llevarán a cabo a lo largo de esta guía.

1.2. Crear un programa primer_programa.rb, se puede hacer desde el terminal o desde el editor de texto, asignar 2 variables de tipo String para luego imprimir por pantalla las 2 variables concatenadas.

```
var_1 = "hola"
var_2 = "mundo"
puts var_1  + var_2
```

1.3. Ejecutar el programa en el terminal.

```
$ ruby primer_programa.rb
```

Obtendrá una salida de las 2 variables concatenadas

1.4. Editar el archivo creado anteriormente, agregar el siguiente código y ver lo que se muestra por pantalla.

```
var_1 = "hola"
var_2 = "mundo"
puts var_1  * 2
```

Como se observa, se multiplica el contenido de las variables que son de tipo String, por la cantidad de veces que se le diga, en este caso la palabra hola se muestra 2 veces y la palabra mundo 3 veces. (**Figura 13**)

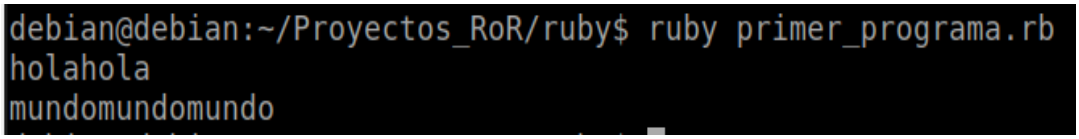

Figura 13 Ejecución del programa variables String

2.Números

2.1. Crear un programa nuevo llamado programa_numero.rb, en el que se asignarán 2 variables enteras para realizar operaciones de aritmética básica.

```
var_1 = 20
var_2 = 5

#suma
puts var_1 + var_2
puts ""

#resta
puts var_1 - var_2
puts ""

#multiplicar
puts var_1 * var_2
puts ""

#dividir
puts var_1 / var_2
puts ""

#modulo
puts var_1 % var_2
puts ""

#números aleatorios
puts rand(100)
```

2.2. Ejecutar el programa en el terminal y observar la salida.

```
$ ruby programa_numero.rb
```

```
debian@debian:~/Proyectos_RoR/ruby$ ruby programa_numero.rb
25

15

100

4

0

54
```

Figura 14 Ejecución del programa de números

3.Conversiones

3.1 En Ruby existen distintos métodos que se aplican a objetos como los String, números enteros, etc. Existen métodos especiales de conversiones que se utilizan en diferentes formas o casos, para observar el funcionamiento de estos, crear un archivo programa_conversiones.rb, declarar una variable entera y concatenar con un texto.

```
var_1 = 22
puts var_1 + " Esto es un entero"
```

3.2 Ejecutar el programa en el terminal.

```
$ ruby programa_conversiones.rb
```

Imprime un error a como se observa en la siguiente figura, esto es debido a que no se puede concatenar un objeto de tipo entero con una cadena de caracteres.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby programa_conversiones.rb
programa_conversiones.rb:3:in `+': String can't be coerced into Integer
 (TypeError)
        from programa_conversiones.rb:3:in `<main>'
```

Figura 15 Error de ejecución del programa de conversiones

3.3 Para solucionar ese error, hacer uso del método **to_s**, editar el programa y agregar:

```
var_1 = 22
puts var_1.to_s + " Esto es un entero"
puts ""
puts var_1
```

Se obtendrá una salida como en la mostrada en la siguiente figura, como se observa, aunque se ha utilizado el método to_s, la variable **var_1** sigue teniendo el mismo valor entero, pero su representación es como cadena de caracteres.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby programa_conversiones.rb
22 Esto es un entero

22
```

Figura 16 Ejecución programa de conversiones

3.4 Editar nuevamente el programa para hacer uso de los métodos **to_i,** el cual convierte una variable a entero y **to_f,** el cual convierte una variable a flotante.

```
var_1 = 22
var_2 = "22"

puts var_1.to_s + " Esto es un entero"
puts ""

puts var_2 + " Esto es una cadena"
puts "La suma de las variables es:"

puts var_2.to_i + var_1
puts var_2.to_f
```

3.5 Guarda los cambios y ejecutar el programa en el terminal.

```
$ ruby programa_conversiones.rb
```

Se obtendrá la siguiente salida.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby programa_conversiones.rb
22 Esto es un entero

22 Esto es una cadena
La suma de las variables es:
44
22.0
```

Figura 17 Ejecución segundo programa de conversiones

4.Métodos gets y chomp.

Se ha visto que el método **puts** se utiliza para imprimir en la pantalla; por el contrario, para leer existe el método **gets** que trabaja junto con el método **chomp**, lo que hace este último es eliminar el carácter “enter” al momento de que el método gets lee un dato del teclado.

4.1 Crear un programa leer.rb y agregar el siguiente código.

```
puts "Ingrese su primer nombre"
nombre = gets
puts "Bienvenido "+ nombre + "disfrute! "
```

Al ejecutar el programa se obtiene la siguiente salida.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby leer.rb
Ingrese su primer nombre
Pedro
Bienvenido Pedro
 disfrute!
```

Figura 18 Ejecución del programa del método gets

Como se puede observar, el método gets recibe el carácter "enter" como un carácter más de lectura, para solucionar eso es que se utiliza el método chomp.

4.2 Editar el programa anterior y utilizar el método chomp al momento de leer el nombre.

```
puts "Ingrese su primer nombre"
nombre = gets.chomp

puts "Bienvenido #{nombre} disfrute! "
```

Al ejecutar el programa se observa la diferencia, cuando se usa el método **chomp** que ya no captura el enter como un carácter más, de igual forma se observa como la variable **nombre** es impresa de una forma distinta a las anteriores.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby leer.rb
Ingrese su primer nombre
Pedro
Bienvenido Pedro disfrute!
```

Figura 19 Ejecución del programa del método get.chomp

5.Métodos de String

Como se menciona anteriormente, en Ruby existen distintos métodos que se pueden aplicar a cada uno de los objetos del lenguaje, en esta sección se conocerá sobre los métodos relacionados a los String.

5.1 Crear un nuevo programa string.rb y agregar el siguiente código:

```
puts "Ingrese su nombre"
nombre = gets.chomp

    #Imprime el nombre ingresado
    puts "Nombre => "+ nombre

    # convierte al revés el nombre
    puts "Método reverse => " + nombre.reverse

    #Mayúscula
    puts "Método upcase => " + nombre.upcase

    #Minúscula
    puts "Método downcase => " + nombre.downcase

    #Intercambia las minúsculas por mayúscula (viceversa)
    puts "Método swapcase => " + nombre.swapcase

    #cambia el primer carácter a mayúscula
    puts "Método capitalize => " + nombre.capitalize

    #devuelve el tamaño del string ingresado
    puts "Método length => " + nombre.length.to_s
```

5.2 Ejecute el programa en el terminal y observar el comportamiento de los métodos.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby string.rb
Ingrese su nombre
María
Nombre => María
Método reverse => aíraM
Método upcase => MARÍA
Método downcase => maría
Método swapcase => mARÍA
Método capitalize => María
Método length => 5
```

Figura 20 Ejecución del programa de métodos para String

6.Condicionales y bucles

6.1 Los condicionales y los bucles en Ruby funcionan de la misma manera que en otros lenguajes de programación, para ver el funcionamiento, crear un programa nuevo y agregar el siguiente código.

```
iterador = " "

while iterador.downcase != "s"

puts "Ingrese un nombre"
nombre = gets.chomp
tamaño = nombre.length

    if (tamaño >= 5 )

     puts "Su nombre tiene más de 5 caracteres"

    else

     puts "Su nombre tiene menos de 5 caracteres"

    end
puts "\nPara salir presione la letra S"


iterador = gets.chomp
end
puts "Ha salido del programa"
```

6.2 Ejecutar e interactuar con el programa para ver su funcionamiento.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby condicionales.rb
Ingrese un nombre
Francisco
Su nombre tiene mas de 5 caracteres

Para salir presione la letra S
n
Ingrese un nombre
juan
Su nombre tiene menos de 5 caracteres

Para salir presione la letra S
n
Ingrese un nombre
Rafael
Su nombre tiene mas de 5 caracteres

Para salir presione la letra S
s
Ha salido del programa
```

Figura 21 Bucles y condicionales

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía, probar su funcionamiento y analizar cada uno de los programas planteados.

2. Crear un programa en Ruby que imprima por pantalla una pirámide de caracteres, el programa solicita al usuario un carácter cualquiera, y un número entero que será el tamaño de la pirámide como se muestra en la figura 22.

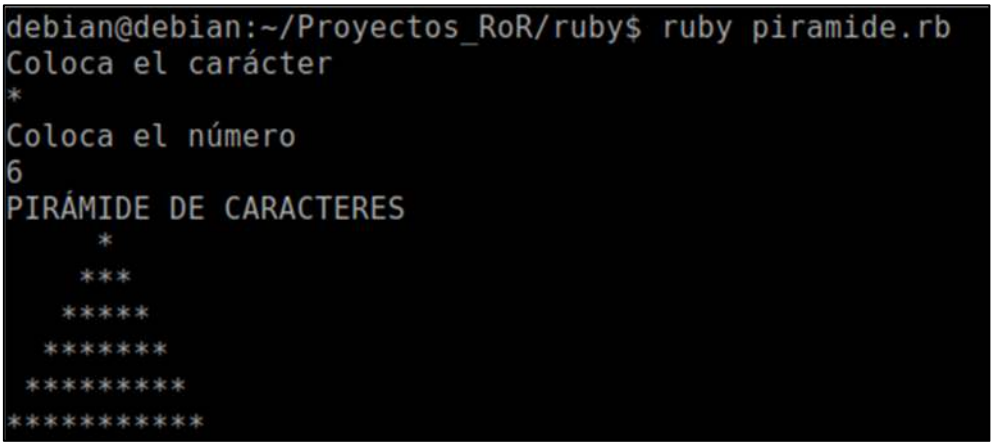

Figura 22 Ejecución del ejercicio propuesto

Guía 2: Introducción al lenguaje Ruby (parte II)

Introducción

Ruby es considerado un lenguaje flexible, no limita al programador, soporta ser alterado libremente, agregar funcionalidades, quitarlas o redefinirlas, "todo es un objeto". Es un lenguaje de guiones (scripts) que permite hacer llamadas al sistema directamente e incluye una gestión de memoria de manera automática.

En esta guía el estudiante aprenderá diferentes métodos que se pueden aplicar en relación a los array y el comportamiento de estos, podrá definir métodos propios y clases en Ruby y también conocer más sobre los tipos de variables del lenguaje.

Objetivos

- Conocer cómo se declara un array en Ruby.
- Analizar el funcionamiento del método each.
- Aprender a definir métodos propios utilizando el lenguaje Ruby.
- Crear hashes, clases, métodos y los tipos de variables en Ruby.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- RubySur. Aprende a programar. rubysur.org/aprende.a.programar/
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/quickstart/
- James Britt and Neurogami. Hash. ruby-doc.org/core-2.5.0/Hash.html

Desarrollo

1. Array.

1.1. En el directorio ruby, crear un programa en Ruby y asignar a un array los días de la semana, para luego imprimirlos por pantalla.

```
#array
semana=["lunes","martes","miércoles","jueves","viernes","sábado","domingo"]

puts semana

puts "\nImprimir por posición"
puts semana [1]
puts semana [3]
puts semana [5]
```

1.2. Ejecutar el programa en el terminal y analizar lo que imprime.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby array.rb
lunes
martes
miércoles
jueves
viernes
sábado
domingo

Imprimir por posición
martes
jueves
sábado
```

Figura 23 Ejecución del programa de array

Como se observa en la figura 23, cada uno de los datos del array tiene una posición dentro de él, pudiéndose llamar a cada uno de ellos, con el simple hecho de escribir en qué posición se encuentra el dato que se quiere mostrar por pantalla.

2. Método each

2.1. El método each en Ruby se utiliza como iterador para recorrer un array, tomando como ejemplo el programa anterior, crear uno nuevo y utilizar el método each para recorrer el array e imprimirlo por pantalla.

```
semana = ["lunes","martes","miércoles","jueves","viernes","sábado","domingo"]

      i= 0

      semana.each do |dia|

          puts "dia " + i.to_s + "=" + dia

          i +=1

      end
```

La variable i sólo se utiliza como un contador para mostrar que dato es el almacenado en cada posición del array.

2.2. Ejecutar el programa y verificar su funcionamiento

```
debian@debian:~/Proyectos_RoR/ruby$ ruby metodo_each.rb
dia 0=lunes
dia 1=martes
dia 2=miércoles
dia 3=jueves
dia 4=viernes
dia 5=sábado
dia 6=domingo
```

Figura 24 Ejecución del programa usando el método each

El método each es el método más utilizado al momento de recorrer un array en Ruby, itera cada posición que existe en el array de manera ordenada.

3. Métodos para trabajar con array

En Ruby existen muchos métodos específicamente para trabajar con array, entre los cuales se pueden encontrar: **pop, push, join, last, split**. En este enunciado se mostrará el funcionamiento de algunos de ellos, los cuales son muy útiles en el desarrollo de aplicaciones en donde se trabaja con el lenguaje Ruby.

3.1. A continuación, se deberá realizar un programa en el que se utilicen algunos de los métodos antes mencionados.

```
semana = ["lunes","martes","miércoles","jueves","viernes","sábado","domingo"]

puts "Array en Ruby"
puts semana

puts "\nMétodo to_s"
puts semana.to_s

puts "\nMétodo join"
puts semana.join (",")

puts "\nMétodo first"
puts semana.first

puts "\nMétodo last"
puts semana.last

puts "\nMétodo length"
puts semana.length
```

3.2. Ejecutar el programa y verificar el funcionamiento, es importante ver cómo se comporta cada uno de los métodos con respecto al array.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby metodo_array.rb
Array en Ruby
lunes
martes
miércoles
jueves
viernes
sábado
domingo

Método to_s
["lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"]

Método join
lunes,martes,miércoles,jueves,viernes,sábado,domingo

Método first
lunes

Método last
domingo

Método length
7
```

Figura 25 Ejecución del programa de métodos de array

3.3. Modificar el programa anterior, y hacer uso de los métodos push y pop para ver la diferencia del comportamiento entre ambos, en relación a su uso sobre los arrays.

```
semana = ["lunes","martes","miércoles","jueves","viernes","sábado","domingo"]

puts "Array completo"
puts semana.to_s

 puts "\nMétodo POP"
 puts semana.pop

 puts "\nMétodo length"
 puts semana.length

 puts "\nÚltimo dato"
 puts semana.last

 puts "\nMétodo PUSH"
 puts semana.push "final"

 puts "\nTamaño nuevo"
 puts semana.length
```

Al ejecutar el programa se observa que el método **pop** extrae el último dato dentro del array y lo elimina del conjunto, a diferencia del método **push**, que lo que hace es insertar el valor del dato indicado al final del conjunto.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby push_pop.rb
Array completo
["lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"]

Método POP
domingo

Método length
6

Último dato
sábado

Método PUSH
lunes
martes
miércoles
jueves
viernes
sábado
final

Tamaño nuevo
7
```

Figura 26 Ejecución del programa de métodos push y pop

4. Métodos propios

4.1. En Ruby como en cualquier otro lenguaje de programación se pueden definir métodos para que realicen cierto trabajo, para entender un poco mejor de esto, crear un nuevo programa llamado metodos_propios.rb y agregar el siguiente código.

```
#definición de métodos
#método con parámetros
def edad (año_naciento, año_actual)

    edad = año_actual.to_i - año_naciento.to_i
   puts "\nTu edad actual es #{edad} años"
end
#método sin parámetros
def nombre
   puts "Ingrese su nombre"
   nombre = gets.chomp

   if nombre.downcase
      nombre = nombre.upcase
   else
       nombre = nombre
   end
   puts "Bienvenido #{nombre}"
end

#Invocación de los métodos
nombre
puts "Ingrese su año de nacimiento"
año_naciento = gets.chomp

puts "Ingrese el año actual"
año_actual = gets.chomp

edad(año_naciento, año_actual)
```

Como se puede observar se han definido dos métodos, uno llamado **nombre** que no recibe parámetros y el otro llamado **edad**, que recibe dos parámetros y que será el encargado de calcular la edad de una persona.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby metodos_propios.rb
Ingrese su nombre
María
Bienvenido MARÍA
Ingrese su año de nacimiento
1995
Ingrese el año actual
2018

Tu edad actual es 23 años
```

Figura 27 Ejecución del programa de métodos propios

5. Hash

5.1. Algo muy utilizado en el lenguaje Ruby son los hashes al momento de trabajar con datos. Crear un programa llamado hash.rb y agregar el código a continuación.

```
colorHash  = {}

colorHash['rojo ']  = '#FF0000'
colorHash['verde']  = '#008000'
colorHash['azul ']  = '#0000FF'

colorHash.each do |tipoCodigo, color|
  puts tipoCodigo + ' :  ' + color
end
```

5.2. Ejecutar el programa en el terminal para obtener la salida.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby hash.rb
rojo  :  #FF0000
verde :  #008000
azul  :  #0000FF
```

Figura 28 Ejecución del programa hash

En los hashes, cada dato almacenado se guarda con un nombre que se conoce como una clave, esta clave puede ser textos o número y es por medio del cual se puede identificar cada uno de los datos que pertenecen al hash; como se puede observar en la **figura 28**, se imprime cada dato del hash por medio de su clave.

5.3. Para ver de otra manera el funcionamiento, crear un nuevo programa hash_2.rb y agregar el código.

```
user = {}
user = { :name => "Juan Pérez", :email => "JuanP@example.com" }

puts "Nombre de usuario: #{user [:name]} "
puts "\nCorreo: #{user [:email]}"
```

Al ejecutar el programa se observa cómo se hace referencia a los datos del usuario, haciendo uso de la clave para poder mostrarlos por pantalla.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby hash_2.rb
Nombre de usuario: Juan Pérez

Correo: JuanP@example.com
```

Figura 29 Ejecución del segundo programa de hash

6. Clases

6.1. Crear un programa clases.rb, en el cual se creará una clase Palíndromo que contendrá un método para verificar una frase ingresada, como aparece a continuación.

```
class Palindromo

  def verificar_frase (frase)

    if frase == frase.reverse
      puts "La frase #{frase} ES palíndromo"

    else
    puts "La frase #{frase} NO es palíndromo"

    end

  end

end

puts "Ingrese una frase"
frase = gets.chomp

verificar = Palindromo.new
```

6.2. Ejecutar el programa e ingresar la palabra "level" para verificar su correcto funcionamiento.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby clases.rb
Ingrese una frase
level
La frase level ES palíndromo
```

Figura 30 Ejecución del programa de clases

Una frase Palíndromo es una palabra o expresión, que es igual si se lee de izquierda a derecha que de derecha a izquierda. Puede ejecutar el programa de nuevo e interactuar con el funcionamiento, para realizar distintas pruebas ingresando otras palabras o frases.
Ejemplo: "anitalavalatina".

7. Variable de instancia.

7.1. Las variables de instancia son variables de un objeto, una de las diferencias de las variables locales es que estas existen hasta que el método ha terminado e inician con arroba "@". Crear un programa en Ruby llamado variables.rb y escribir lo siguiente:

```
class Dado

 def initialize
    rodar
 end

 def rodar
    @numero_mostrar = 1 + rand(6)
 end

 def mostrar
   @numero_mostrar
 end
end

puts "cuantas veces desea lanzar el dado"
lanzar = gets.chomp

while lanzar.to_i > 0

    lanza_dado = Dado.new.mostrar
    puts "\nLanzamiento "
    puts lanza_dado
    lanzar = lanzar.to_i - 1

end
```

Como se observa en el código la variable **numero_mostrar**, se utiliza en los métodos rodar y mostrar, y siempre mantiene el mismo el valor.

7.2. Ejecutar el programa y verificar el funcionamiento de la variable numero_mostrar, la cual mantiene su valor en todos los métodos hasta ser mostrada por pantalla.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby variables.rb
cuantas veces desea lanzar el dado
4

Lanzamiento
3

Lanzamiento
3

Lanzamiento
2

Lanzamiento
1
```

Figura 31 Ejecución del programa de variables de instancia

Ejercicios propuestos para ser entregados al docente

1. Realice cada uno de los enunciados de la guía, probar el funcionamiento y analizar cada uno de los programas planteados.

2. Crear un programa en Ruby que contenga un hash, el cual este compuesto de nombre = clave y celular = valor, el programa deberá mostrar el hash completo, solicitar el nombre que sería la clave y retornar el celular que sería el valor, correspondiente a ese nombre. Deberá validar si el dato existe en el hash y que cuando se ingrese un nombre en minúscula a como se muestra en la **figura 32,** el nombre **Juan** se ingresó en minúscula y el programa devuelve el celular correspondiente al nombre.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby asignacion2.rb
Nombre          Celular

María           2248-6559
Pedro           9845-6532
Juan            8265-4536
Alberto         7896-4514
-------------------------
Ingrese un nombre
juan
-------------------------
El número de celular de Juan : 8265-4536
```

Figura 32 Ejercicio propuesto 2

3. Realice un programa en Ruby que solicite por pantalla un número cualquiera y que imprima la suma de los números pares e impares que componen el número ingresado, para la solución crear una clase de nombre **Calcular** la cual contendrá 2 métodos, el primer método para los cálculos de los números pares y el segundo método para los cálculos de los numero impares, se deberá mostrar a como se muestra en la **figura 33**.

```
debian@debian:~/Proyectos_RoR/ruby$ ruby suma_pares_impares.rb
Ingrese el número maximo a sumar:
15
La suma de los números PARES de 15 es:
56
La suma de los números IMPARES de 15 es:
64
```

Figura 33 Ejecución del ejercicio propuesto

Extra: hacer que el programa número 1 que se muestra en la **figura 32** se ejecute varias veces hasta presionar una tecla.

Guía 3: Trabajando con Rspec y Ruby

Introducción

Rspec es una herramienta de pruebas escrita en Ruby, mayormente utilizada para el desarrollo en base a pruebas o TDD y desarrollo en base comportamiento o DBB. En esta guía el estudiante pondrá en práctica los conocimientos obtenidos en el desarrollo de las guías anteriores en base a los principios básicos del lenguaje Ruby, de igual forma podrá introducirse en lo que es el desarrollo en base a pruebas utilizando la gema Rspec.

Objetivos

- Aprender a usar la herramienta Rspec.
- Crear pruebas en Ruby utilizando Rspec.
- Escribir código Ruby y hacer funcionar los test.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Sublimetext • Rspec	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- David Chelimsky. The Rspec book. https://goo.gl/GKNzwK
- Ed Wassemann. Pruebas RSpec para principiantes, Parte 1. https://code.tutsplus.com/es/articles/rspec-testing-for-beginners-part-1--cms-26716
- saasbook. Ruby Intro. https://github.com/saasbook/hw-ruby-intro
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/quickstart/

Desarrollo

1. Crear un directorio donde almacenar el proyecto.

   1.1. Crear un archivo llamado Gemfile dentro del directorio creado anteriormente, este archivo es donde se agregará la gema que necesita el proyecto, en este caso solo se utilizará la gema rspec para realizar los tests al código.

   1.2. Abrir el archivo Gemfile y agregar las siguientes líneas, para poder integrar rspec al proyecto.

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

2. En el terminal, ubicarse dentro del directorio de trabajo y escribir los siguientes comandos.

   2.1. Instalar la gema rspec, que es la que se utiliza para trabajar con TDD y que servirá para elaborar los tests.

```
$ bundle install
```

   2.2. Ejecutar rspec.

Se crearán automáticamente dos archivos: .rspec y spec/spec_helper.rb que son los archivos que utiliza rspec para correr los tests.

```
$ bundle exec rspec --init
```

3. En el directorio raíz del proyecto, crear una carpeta nueva llamada lib

   3.1. Dentro de la carpeta lib, crear un archivo ruby.rb y agregar lo siguiente.

```
def intro array
#escribir el código
end

def nombre nom
#escribir el código
end

def datos? array
#escribir el código
end
```

Uno de los principales motivos de crear tests, es que dan garantía de un código de mejor calidad que ha sido sometido a diferentes eventualidades y todos los posibles bugs fueron corregidos. Como se observa, en el código anterior existen 3 métodos que serán donde se va escribir el código Ruby en base a los tests que se aplicarán en el proyecto, mostrado a continuación.

4. Dentro de la carpeta **spec,** crear un archivo **rspec_1.rb** y copiar el siguiente código, este código es cada uno de los tests aplicados, para escribir el código Ruby dentro de los métodos que están en el archivo **ruby.rb**.

```
require "ruby.rb"

describe "#intro" do

    it "Debe estar definada" do
            expect{intro([1,2,3,4])}.not_to raise_error
    end

    it "retorna la suma del array" do
            expect(intro ([2,6,10,5,2])).to eq(25)
            expect(intro ([15,6,10,19])).to eq(50)
            expect(intro ([15,6,4,5])).to eq(30)
     end

    it "retorna 0 si el array está vacío" do
            expect(intro([])).to eq(0)
     end

     it "retorna el valor del único elemento del array" do
            expect(intro([5])).to eq(5)
     end
end

describe "#nombre" do

    it "Debe iniciar con Mayúscula y concatena la cadena con la palabra hello" do
            expect(nombre("Pedro")).to eq("hello Pedro")
            expect(nombre("Juan")).to eq("hello Juan")
            expect(nombre("María")).to eq("hello María")
    end

    it "No trabaja con un string vacío" do
            expect(nombre("")).to eq("vacío")
    end
end

describe "#datos?" do

    it "retorna true si el tamaño del array es > 0" do
            expect(datos?([2,1,5,6])).to be true
            end
    it "retorna false si el tamaño del array es <= 0" do
            expect(datos?([])).to be false
    end
end
```

5. En el terminal, ejecutar el siguiente comando para verificar el funcionamiento de los test.

```
$ rspec spec/rspec_1.rb
```

Al ejecutar el comando anterior mostrará un conjunto de mensajes en color rojo, los cuales corresponden a cada uno de los tests que se están aplicando en el ejercicio, cuando los tests se muestran en color rojo es debido a que los tests ejecutados están fallando, en relación a la funcionalidad del código al cual se le están aplicando cada uno de ellos.

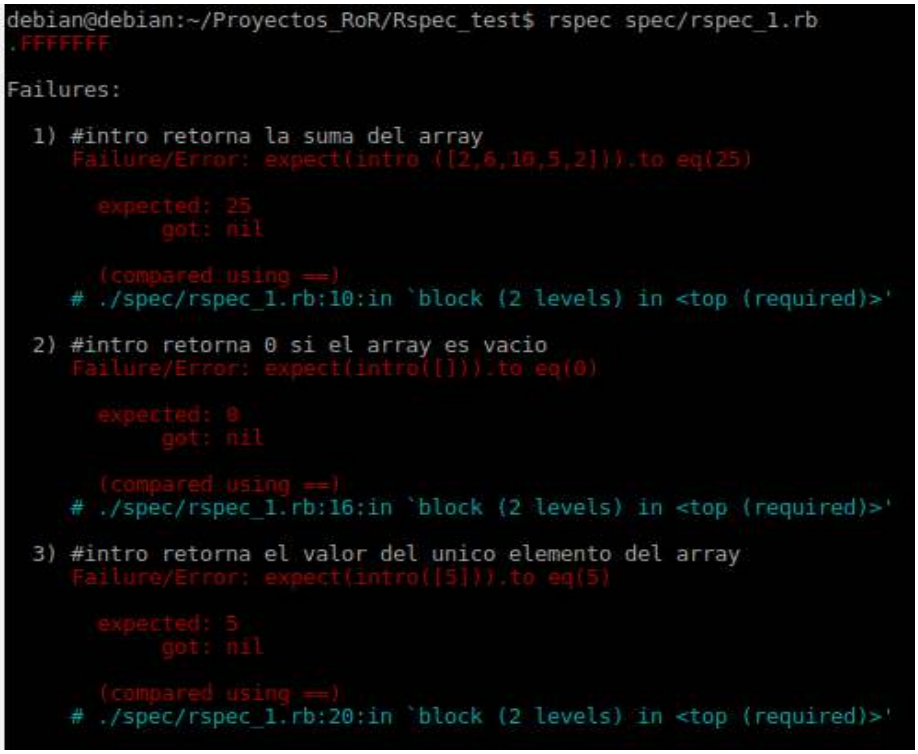

Figura 34 Mensajes de los test en rojo

Al final de todos los mensajes, se muestra exactamente los tests que están siendo ejecutados.

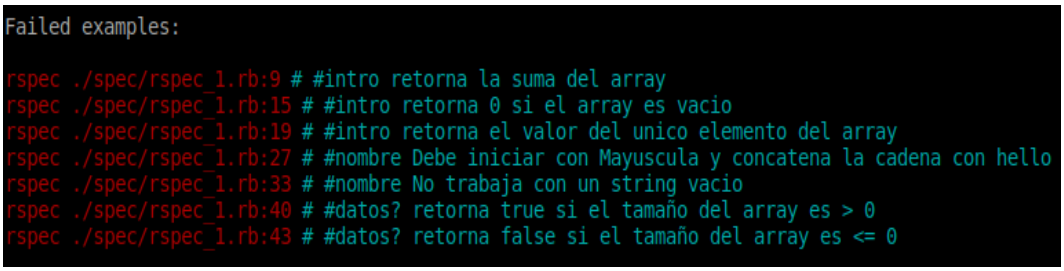

Figura 35 Resumen de los test aplicados

El desarrollo en base a pruebas o TDD no es más que, primero escribir los tests y correrlos esperando que fallen; que es cuando se muestran los mensajes en rojo, escribir código y volver a correr los tests, esperando que estos pasen y que se muestren los mensajes en verde que significa que el código no contiene errores o que los tests funcionan de manera correcta.

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía.

2. Escribir el código Ruby necesario dentro cada una de los métodos creados en el archivo **lib/ruby.rb** para aprobar todos los tests, y que muestre los mensajes en color verde como se observa en la figura 36.

```
debian@debian:~/Proyectos_RoR/Rspec_test$ rspec spec/rspec_1.rb
........

Finished in 0.00657 seconds (files took 0.14529 seconds to load)
8 examples, 0 failures
```

Figura 36 Test resueltos en color verde

Extra: escribir un conjunto de tests en relación a lo aprendido en la guía dentro del archivo **rspec_1.rb,** para luego crear un método y escribir código ruby de tal manera que pueda hacer funcionar los test a como se trabajó en la guía.

Guía 4: Introducción al framework Ruby on Rails

Introducción

Ruby on Rails es un framework de aplicaciones web de código abierto que sigue el paradigma del patrón de diseño Modelo Vista Controlador (MVC), dando la posibilidad de crear aplicaciones web con poco código y configuraciones mínimas.

En esta guía se tendrá un acercamiento al framework para conocer la base de su funcionamiento: versión, configuración, directorios, comandos básicos que se utilizan al momento de crear aplicaciones, etc.

Objetivos

- Verificar las versiones de software instalados
- Conocer comandos básicos del framework RoR.
- Crear una primera aplicación básica en RoR.
- Conocer acerca de la jerarquía de directorios

Tiempo

- Una sesión de clase

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.1.4 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Michael Hartl. Ruby on Rails tutorial (Rail5), Learn Web Development with Rails. https://www.railstutorial.org/book/beginning
- Ruby on Rails org, RailsGuides, The Rails command line. http://guides.rubyonrails.org/command_line.html

Desarrollo

1. Verificar las versiones de software instaladas.

Es importante verificar las versiones de software instaladas, para así poder continuar con la realización de la guía sin ningún problema.

1.1. Verificar la versión de Rails.

```
$ rails -v
```

Con este comando verifica la versión de Rails instalada.

```
debian@debian:~$ rails -v
Rails 5.1.6
```

Figura 37 Versión de Rails

1.2. Verificar la versión de Ruby.

```
$ ruby -v
```

Obtendrá una salida como se muestra en la siguiente figura.

```
debian@debian:~$ ruby -v
ruby 2.4.1p111 (2017-03-22 revision 58053) [i686-linux]
```

Figura 38 Versión de Ruby

1.3. Verificar la versión de sqlite.

```
$ sqlite3 --version
```

```
debian@debian:~$ sqlite3 --version
3.16.2 2017-01-06 16:32:41 a65a62893ca8319e89e48b8a38cf8a59c69a8209
```

Figura 39 Versión de sqlite

2. Crear un nuevo proyecto en Rails.

2.1. Haciendo uso del terminal, ubicarse en el directorio donde se va a almacenar el proyecto.

```
$ cd /home/debian/Proyectos_RoR
```

2.2. Generar un proyecto nuevo utilizando el siguiente comando:

```
$ rails new first_app
```

Al ejecutar el comando anterior se está creando un proyecto nuevo, mostrará unos mensajes de la creación de archivo y directorios a como se observa en la figura 40.

```
debian@debian:~/Proyectos_RoR$ rails new first_app
      create
      create  README.md
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
         run  git init from "."
      create  app
      create  app/assets/config/manifest.js
      create  app/assets/javascripts/application.js
      create  app/assets/javascripts/cable.js
      create  app/assets/stylesheets/application.css
      create  app/channels/application_cable/channel.rb
      create  app/channels/application_cable/connection.rb
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/jobs/application_job.rb
      create  app/mailers/application_mailer.rb
      create  app/models/application_record.rb
      create  app/views/layouts/application.html.erb
      create  app/views/layouts/mailer.html.erb
      create  app/views/layouts/mailer.text.erb
      create  app/assets/images/.keep
      create  app/assets/javascripts/channels
      create  app/assets/javascripts/channels/.keep
      create  app/controllers/concerns/.keep
      create  app/models/concerns/.keep
      create  bin
      create  bin/bundle
      create  bin/rails
      create  bin/rake
      create  bin/setup
      create  bin/update
      create  bin/yarn
      create  config
```

Figura 40 Creación de un proyecto en Rails

2.3. Entrar a la carpeta del proyecto creado.

```
$ cd first_app/
```

Cada vez que se crea un proyecto en Rails, se debe ingresar al directorio que se ha generado con el comando rails new. Este directorio, es el que contiene la nueva aplicación en la que se va a trabajar y en todos los casos se debe estar ubicado dentro de él, para así hacer uso de los distintos comandos que se ejecutan en el desarrollo de aplicaciones con Ruby on Rails.

Jerarquía de directorios

Al momento de crear un nuevo proyecto; Rails genera un conjunto de archivos y directorios necesarios para su funcionamiento, dentro de los más importantes y más utilizados están: el directorio **app** en este se encuentra información sobre las vistas, controladores, archivos css; el directorio **config** en este se encuentran archivos de configuración como **routes.rb** en el cual se configuran las rutas de la aplicación; el directorio **db** es el relacionado a la base de datos del proyecto, en este directorio se almacena el esquema actual de la base de datos y las migraciones que se realizan; el directorio **test** en este es donde se crean los archivos de los tests que se le aplican en la aplicación; y el archivo **Gemfile** este es el encargado de las gemas que se utilizan en el proyecto, a continuación se muestra una breve descripción de cada uno de ellos.

Tabla 8 Jerarquía de directorios de un proyecto en rails

Directorio	Descripción
app/	Se encuentra todos los archivos y directorios relacionados con los **modelos**, **vistas**, **controladores** y **helpers** que son partes de la aplicación. También se encuentra en directorio assets el cual contiene el directorio **images** que es donde se ubican las **images** que contiene la aplicación, el directorios **stylesheets** que contienes los archivos de estilos y el directorio **javascripts** donde guardan los archivos javascripts (js) de la aplicación.
bin/	Contiene un conjunto de scripts que son incluidos dentro del propio framework.
config/	Se encuentran los archivos de configuración de las rutas, de la base de datos etc.
db/	Almacena el esquema actual de la base de datos y las migraciones.
Gemfile	Uno de los archivos más importantes del proyecto, este es el archivo de gemas encargado de especificar las dependencias de la aplicación Rails.
lib/	Carpeta donde se incluyen módulos que extienden la aplicación.
log/	Contiene los ficheros de registros de actividades del servidor.
public/	Directorio de donde se sirven los archivos estáticos y los recursos compilados del framework.
Rakefile	Sirve para localizar las tareas que se pueden ejecutar desde la línea de comandos.
README.md	Contiene una pequeña descripción de la aplicación.
test/	Test unitarios de la aplicación.
tmp/	Archivos temporales como caché, archivos pid del sistema, etc.
vendor/	Carpeta donde introducir el código de terceros a la aplicación.

2.4. Una vez ubicado dentro del directorio del nuevo proyecto creado, iniciar el servidor para comprobar que arranca sin ningún problema.

```
$ rails server
```

Con la versión de Rails que se trabaja, viene instalado el servidor PUMA por defecto, que es una pequeña librería que proporciona un servidor HTTP muy rápido y concurrente para aplicaciones web de ruby, creado por Evan Phoenix a finales del 2011. A diferencia de otros servidores utilizados en un ambiente de desarrollo como WEBrick que es un servidor sencillo y que permite un solo proceso lo cual hace que las conexiones y solicitudes tarden más tiempo; PUMA utiliza hilos, además de procesos de trabajo para aprovechar de una mejor forma la CPU. En el proyecto creado en Rails se puede observar la configuración del servidor en el archivo **config/puma.rb**, este servidor se utiliza comúnmente en un ambiente de desarrollo, pero debido a su gran capacidad de procesamiento también puede ser utilizado en ambiente de producción.

Si el servidor inicia de manera correcta, mostrará unos mensajes en el terminal donde se puede observar el nombre del servidor que se ejecuta, versión de Rails, versión de Ruby, cantidad de hilos o procesos configurados, ambiente en el que se está trabajando, la dirección y puerto en el que escucha el servidor, a como se muestra en la siguiente figura.

```
debian@debian:~/Proyectos_RoR$ cd first_app/
debian@debian:~/Proyectos_RoR/first_app$ rails server
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.11.0 (ruby 2.4.1-p111), codename: Love Song
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

Figura 41 Inicio correcto del servidor

2.5. Abrir el navegador y escribir la siguiente dirección.

```
http://127.0.0.1:3000/
```

Navegando en la dirección anterior, tendrá acceso a la página por defecto del proyecto nuevo.

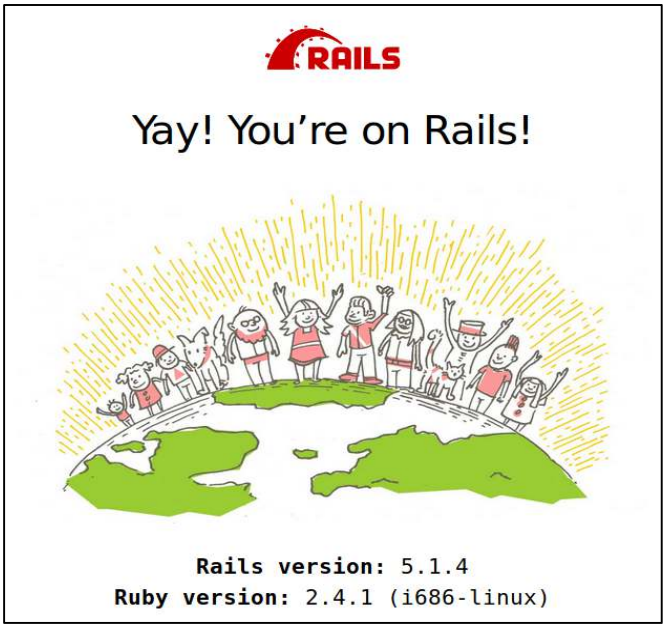

Figura 42 Página de inicio de un nuevo proyecto en Rails

Detener el servidor presionando la combinación de teclas Ctrl+C en el terminal.

3. Agregar una acción al controlador.

3.1. Cuando se crea un proyecto nuevo en Rails, se crea por defecto un controlador llamado Application_controller. Para Verificar que existe el controlador, puede ejecutar el siguiente comando en el terminal.

```
$ ls app/controllers/*_controller.rb
```

Obtendrá una salida como la de la siguiente figura.

```
debian@debian:~/Proyectos_RoR/first_app$ ls app/controllers/*_controller.rb
app/controllers/application_controller.rb
```

Figura 43 Controlador de la aplicación

3.2. Abrir el archivo del controlador que se encuentra en la dirección app/controllers/**application_controller.rb** en el editor de texto, y definir la acción del controlador.

```
def hola
 render html: "hola, mundo!"
end
```

Por defecto, cuando se ejecuta la acción de un controlador, Rails renderiza una vista que se llame igual a la acción (al método) y que se encuentre en una carpeta que se llame igual al controlador dentro del directorio **app/views.** Este comportamiento se puede cambiar haciendo uso del método **render,** con este método se puede renderizar código HTML o una vista diferente**,** como se puede observar en el código anterior en la acción hola se hace un render html del texto "hola mundo"

El archivo controller/**application_controller.rb** deberá quedar como se muestra en la **figura 44.**

Figura 44 Definición de la acción del controlador

4. Configurar el archivo **routes.rb**.

El archivo routes.rb es donde se definen las rutas de la aplicación en Rails, cada ruta está compuesta por: verbos HTTP (GET, POST, PATCH, PUT, DELETE), path, controlador y método (acción). Otra parte importante es el método root, este método es por medio del cual se establece la ruta raíz de la aplicación, esta ruta solicitudes get a una acción, es recomendable que esta ruta siempre este el parte superior del archivo routes.

4.1. Abrir el archivo routes (**config/routes.rb**) y agregar la siguiente línea.

```
root 'application#hola'
```

En este caso al método root se hace relación al controlado y la acción **hola** que pertenece a ese controlador.

El archivo config/**routes.rb** quedará como se muestra a continuación.

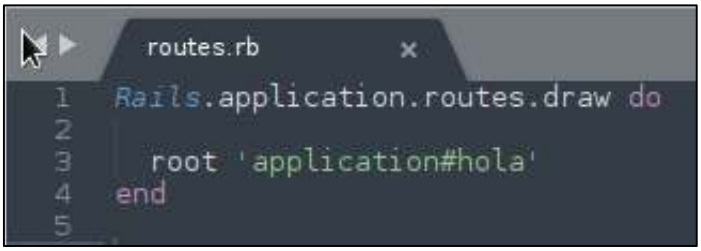

Figura 45 Archivo routes.rb

4.2. Guardar los cambios realizados e iniciar el servidor en el terminal.

```
$ rails s
```

4.3. Abrir el navegador en la dirección 127.0.0.1:3000 y verificar que ahora, la página principal presenta la acción **hola** creada anteriormente.

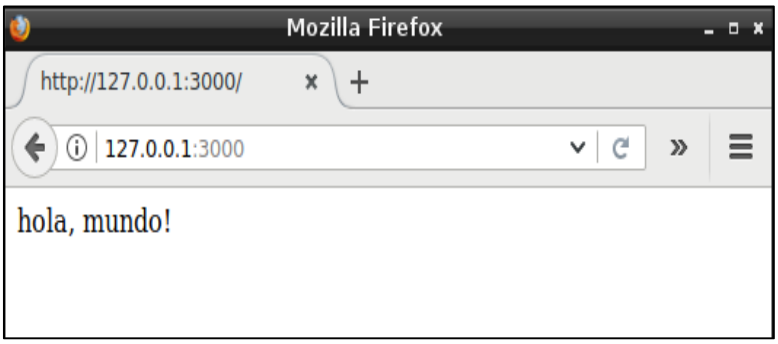

Figura 46 Página principal del proyecto

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía y verificar el correcto funcionamiento.
2. Crear una acción nueva que muestre un texto cualquiera.
3. Editar el archivo routes.rb, para que la aplicación siempre inicie, por defecto, con la nueva acción creada en el inciso anterior.

Guía 5: Incorporación de código Ruby dentro del framework Rails.

Introducción

Ruby on Rails también conocido como RoR, es un framework escrito en lenguaje de programación Ruby desarrollado por David Heinemeier.

En esta guía se creará un pequeño proyecto haciendo uso del framework y se trabajará con las vistas de los usuarios finales, introduciendo código Ruby dentro estas, se aprenderá un poco sobre las hojas de estilos, como agregar un controlador nuevo a un proyecto, así como también se hará uso de archivos de configuración ya anteriormente utilizados.

Objetivos:

- Integrar código Ruby dentro de las vistas.
- Agregar estilos (CSS) a un proyecto en Rails.
- Crear controladores propios.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Michael Hartl. Ruby on Rails tutorial (Rail5), Learn Web Development with Rails. https://www.railstutorial.org/book/
- LibrosWeb.es. capítulo 4. Hola, Rails! http://librosweb.es/libro/introduccion_rails/capitulo_4.html
- Ruby on Rails org, RailsGuide, The Rails command line. http://guides.rubyonrails.org/command_line.html

Desarrollo

1. Creación de un proyecto nuevo.
   1.1. Ubicarse en el directorio donde se va almacenar el proyecto.

```
$ cd /home/debian/Proyectos_RoR
```

   1.2. Generar un nuevo proyecto.

```
$ rails new my_app
```

   1.3. Ubicarse dentro del proyecto e iniciar el servidor para comprobar que funciona sin ningún problema.

```
debian@debian:~/Proyectos_RoR$ cd my_app/
debian@debian:~/Proyectos_RoR/my_app$ rails s
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.11.2 (ruby 2.4.1-p111), codename: Love Song
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

Figura 47 Mensajes al iniciar el servidor de Rails

   1.4. Abrir el navegador y escribir la siguiente dirección.

```
http://localhost:3000
```

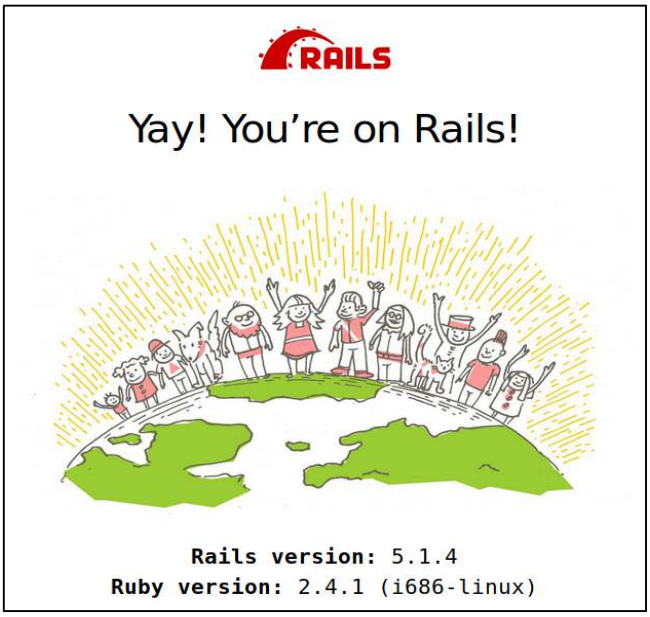

Figura 48 Index por defecto del proyecto

2. Generar un nuevo controlador al proyecto.

2.1. Verificar cuál es el controlador existente dentro del proyecto. De la siguiente forma se obtendrá el nombre de los controladores que existen hasta el momento.

```
$ ls app/controllers/*_controller.rb
```

2.2. Generar el nuevo controlador y una acción correspondiente a ese controlador.

```
$ rails generate controller welcome index
```

```
Running via Spring preloader in process 20793
      create  app/controllers/welcome_controller.rb
       route  get 'welcome/index'
      invoke  erb
      create    app/views/welcome
      create    app/views/welcome/index.html.erb
      invoke  test_unit
      create    test/controllers/welcome_controller_test.rb
      invoke  helper
      create    app/helpers/welcome_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/welcome.coffee
      invoke    scss
      create      app/assets/stylesheets/welcome.scss
```

Figura 49 Detalles de archivos y directorios creados al generar el controlador

Con el comando anterior se genera y agrega un nuevo controlador (llamado welcome) al proyecto, de igual forma se crean un conjunto de archivos y directorios propios del controlador, que trabajan en conjunto para su funcionamiento, dentro de los más importantes se puede mencionar el archivo **welcome_controller.rb** y el archivo de las vistas llamado **index.html.erb,** que se pasó como parámetro al momento de generar el nuevo controlador. Dentro del controlador se crear una acción de nombre index que hace referencia a la vista index.html.erb del controlador, en la mayoría de los casos el controlador necesita de una vista por medio de la cual se muestra al usuario cada una de las acciones que puede realizar un controlador en específico.

2.3. Verificar nuevamente los controladores del proyecto. Podrá observar que ahora ya existe uno nuevo, y que se corresponde con el creado anteriormente.

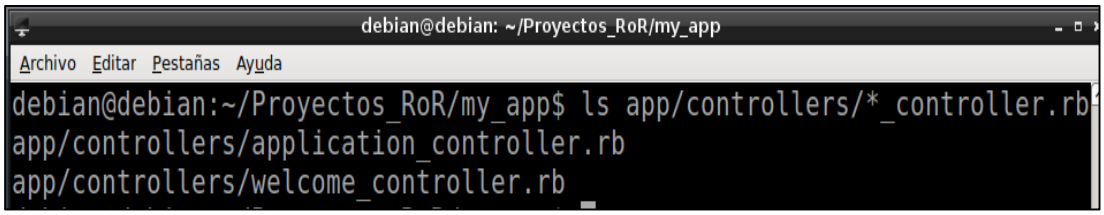

Figura 50 Controladores existentes en el proyecto

2.4. **Abrir** el archivo **welcome_controller.rb** que se encuentra en el directorio (**app/controllers**/) y verificar que existe el método index, que fue creado al momento de generar el controlador, este es el método (acción) que tiene relación con la vista index.html.erb.

Figura 51 Archivo del controlador

3. Trabajar con la vista

Para comenzar a trabajar con las vistas se debe ubicar el directorio (**app/view**/), que es donde se almacenan las vistas del proyecto, ahí se observa un archivo llamado **index.html.erb**, este es por medio del cual los usuarios finales interactúan con la aplicación

3.1. Editar el archivo **index.html.erb** del nuevo controlador y escribir el siguiente código html.

```
<h1>Bienvenido</h1>
<p>Página de inicio</p>
<p>UNAN-León</p>
```

3.2. Editar el archivo **welcome.scss,** que se encuentra en el directorio **(app/assets/stylesheets/),** en este directorio es donde se almacenan todas las hojas de estilos que contiene el proyecto. Todos los archivos de estilos en rails terminan en ".scss" debido a que el framework permite utilizar el lenguaje de estilos llamado Sass (Syntactically Awesome StyleSheets), es una extensión de css que agrega potencia y elegancia al lenguaje básico. Para darle un poco de formato a las vistas, agregar el siguiente código.

```
h1
  {
    color: black;
    height: 30px;
    text-align: center;
  }

p
  {
  height:15px;
  text-align: center;
}
```

3.3. Ir a la dirección (http://localhost:3000/welcome/index), podrá obtener una vista como la de la **figura 52**.

Figura 52 Vista del index

Como se observa en el navegador, se le ha aplicado formato al texto en la vista a través de las hojas estilos.

4. Si navega directamente en **http://localhost:3000/** no se tendrá acceso a la vista antes modificada, para esto, se debe modificar el archivo **routes.rb,** de manera que inicie (root) con la vista de nombre index.html.erb.

5. Actualizar el navegador en la dirección http://localhost:3000/

Figura 53 Nueva página principal del proyecto

Al momento de actualizar el navegador ya se ha obtenido el acceso a la vista en la cual se ha estado trabajando a como se puede observar en la figura 53, esto es debido a que se ha especificado la ruta del controlador, como la ruta raíz o principal del proyecto.

6. Uso del lenguaje Ruby en el framework Rails.

Dentro del framework se trabaja con lenguaje Ruby; de igual forma en las vistas se puede agregar código Ruby a través del **lenguaje Embedded Ruby**, conocido como eRuby o ERB, haciendo uso de etiquetas propias del lenguaje ERB.

6.1. Para ver el funcionamiento ir al archivo **welcome_controller.rb** y crear un array global con los días de la semana dentro del **método index**.

```
@dias_semana =
["Lunes","Martes","Miércoles","Jueves","Viernes","Sábado","Domingo"]
```

6.2. Para poder hacer uso del array dentro de la vista, se debe utilizar lenguaje **ERB** en el archivo de la vista index.html.erb; editar el archivo y escribir el siguiente.

```
<h3>Imprimiendo un array Ruby dentro de una vista</h3>
 <% @dias_semana.each do |dia| %>
   <%= dia %> <br>
<% end %>
```

El código agregado a la vista hace uso de etiquetas propias del lenguaje ERB, como **<%%> y <%=%>,** que se utilizan para poder ingresar código Ruby dentro de las vistas, una de las diferencias entre estas 2 etiquetas es que esta **(<%= %>)** se utiliza cuando se imprime en la vista, como en el caso anterior que imprime cada uno de los datos almacenados en el array, y la etiqueta (**<% %>**) se utiliza para ejecutar código Ruby dentro las vistas pero sin mostrar ningún contenido, el uso de estas se puede observar en la siguiente figura.

Figura 54 Array impreso en la vista utilizando lenguaje ERB

6.3. De igual forma se puede hacer uso de los distintos métodos propios de los array, en el caso anterior se utilizó el método each para recorrer el array. Escribir el siguiente código en el archivo index.html.erb.

```
<h3> Método last</h3>
 <%= @dias_semana.last %>
```

Utilizando el método **last**, se muestra en la vista el último dato almacenado en el array, a como se observa en la **figura 55**.

Figura 55 Uso del método last en la vista

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía.
2. Muestre en la vista **index.html.erb** el primer dato almacenado y el tamaño completo del array, haga uso de métodos similares a los vistos anteriormente.

Ejemplo:

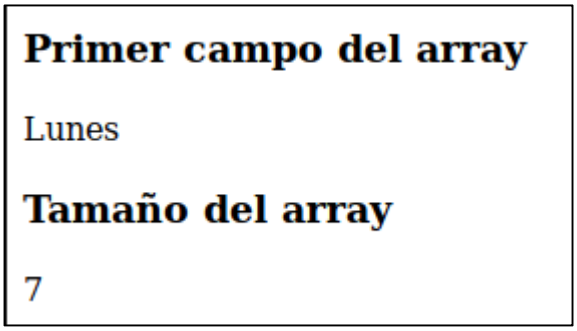

Figura 56 Ejercicio 1, tamaño y primer campo del array

3. Generar un nuevo controlador de nombre “hash” y una acción que es la que tendrá relación con la vista llamada “mostrar”, similar al realizado al inicio de esta guía.
4. Dentro de la acción mostrar que se creó junto al nuevo controlador del proyecto, crear un hash con los datos de una persona: nombre, apellido, teléfono, correo.
5. Modificar la vista del controlador creado anteriormente y mostrar los datos del hash en una tabla html.
6. Agregar código css a la tabla, se mostrará en la vista como en la **figura 57** que se puede ver que la tabla contiene formato, colores, bordes.
7. Configurar el archivo **routes.rb** para cambiar la página de inicio del proyecto y ahora hacerlo apuntar al nuevo controlador junto con su respectiva vista, la cual deberá quedar como se observa en la **figura 57**.

Figura 57 Ejercicio 2, visualizar el contenidode un hash en la vista

Guía 6: Patrón de diseño MVC usando scaffold

Introducción

Ruby on Rails sigue el patrón de arquitectura de software Modelo-Vista-Controlador, que separa los datos de la lógica de negocio, de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

En la siguiente guía se aprenderá a crear aplicaciones web, utilizando el framework Ruby on Rails mediante el generador de código **scaffold**, que permite tener las funcionalidades básicas de administración de un modelo, CRUD (Create, Read, Update, Delete) es común para cualquier sistema transaccional. La idea es también conocer lo que es el patrón de diseño MVC, ya que al hacer uso de scaffold este genera todo el código del modelo, la vista y el controlador de la aplicación.

Objetivos

- Comprender el uso de scaffold y de los métodos CRUD generados.
- Creación de una aplicación usando scaffold.
- Entender el patrón de diseño MVC.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Elicia. Scaffolding en RoR. https://gist.github.com/Elicia/8530046
- Ruby on Rails org, RailsGuide, The Rails command line. http://guides.rubyonrails.org/command_line.html
- Ruby on Rails org, RailsGuide, Active Record Migrations. http://guides.rubyonrails.org/active_record_migrations.html

Desarrollo

1.Crear un nuevo proyecto en Rails.

1.1 Crear un proyecto y ubicarse en su interior haciendo uso del terminal.

```
$ rails new App_Scaffold
```

1.2 Generar un scaffold, con una tabla llamada **Buy** con los campos:

```
$ rails generate scaffold Buy category:string description:text amount:decimal
```

Hay que mencionar algo importante para la creación de modelos, siempre se escriben con la primera letra en mayúscula y en singular; en cualquier otro caso el framework por defecto corrige la escritura, como el caso anterior el nombre del modelo será **Buy**.

Tipos de datos

Cuando se genera un scaffold se debe de pasar como parámetro los tipos que datos que contendrá la tabla de la base de datos donde se guardaran los datos de la aplicación, en la siguiente tabla se muestra los distintos tipos de datos y uso.

Tabla 9 Tipos de datos en Rails

Nombre	Tipo de datos
:binary	Datos binarios
:boolean	Datos tipo booleano puede ser verdadero o falso
:date	Datos de tipo fecha
:datetime	Datos de tipo fecha y hora
:decimal	Datos de tipo decimal.
:float	Datos de tipo flotante
:integer	Datos de tipo entero
:string	Datos de tipo cadena de caracteres.
:text	Dato de tipo texto, parecido al tipo string pero este admite una mayor cantidad de caracteres.
:time	Dato de tipo tiempo.
:timestamp	Tipo de dato similar al datetime, se dice que son sinónimos pero con diferentes características

Cuando se utiliza scaffold para crear operaciones CRUD, este crea todos los archivos y métodos necesarios para el correcto funcionamiento del proyecto. Scaffold crea modelos, vistas, controladores, hojas de estilo, entre otros archivos y directorios como se muestra en la siguiente figura.

```
    invoke  active_record
    create    db/migrate/20180216145848_create_buys.rb
    create    app/models/buy.rb
    invoke    test_unit
    create      test/models/buy_test.rb
    create      test/fixtures/buys.yml
    invoke  resource_route
     route    resources :buys
    invoke  scaffold_controller
    create    app/controllers/buys_controller.rb
    invoke    erb
    create      app/views/buys
    create      app/views/buys/index.html.erb
    create      app/views/buys/edit.html.erb
    create      app/views/buys/show.html.erb
    create      app/views/buys/new.html.erb
    create      app/views/buys/_form.html.erb
    invoke    test_unit
    create      test/controllers/buys_controller_test.rb
    invoke    helper
    create      app/helpers/buys_helper.rb
    invoke      test_unit
    invoke    jbuilder
    create      app/views/buys/index.json.jbuilder
    create      app/views/buys/show.json.jbuilder
    create      app/views/buys/_buy.json.jbuilder
    invoke  test_unit
    create    test/system/buys_test.rb
    invoke  assets
    invoke    coffee
    create      app/assets/javascripts/buys.coffee
    invoke    scss
    create      app/assets/stylesheets/buys.scss
    invoke  scss
    create    app/assets/stylesheets/scaffolds.scss
```

Figura 58 Archivos creados por scaffold

2.Migrar los datos.

Las migraciones son el modo más conveniente de cambiar el esquema de la base de datos a través del tiempo de una manera consistente y fácil, utilizan un lenguaje de definición de esquemas (DSL) en Ruby, por lo que no tiene que escribir SQL.

```
$ rake db:migrate
```

Al escribir el código anterior se obtiene la siguiente salida

```
debian@debian:~/Proyectos_RoR/App_Scaffold$ rake db:migrate
== 20180216145848 CreateBuys: migrating ======================================
-- create_table(:buys)
   -> 0.0023s
== 20180216145848 CreateBuys: migrated (0.0025s) =============================
```

Figura 59 Migración de los datos utilizando migrate

3.Rutas

Las rutas son una parte muy importante de la aplicación. Como se mencionó anteriormente scaffold genera las rutas, para que pueda existir una conexión entre las peticiones del usuario y la aplicación.

Una de las diferencias es que scaffold utiliza enrutamiento de recursos, esto permite declarar todas las rutas comunes para un controlador, en lugar de declarar rutas por separadas por cada una de las acciones que realiza el controlador como: index, show, new etc. Esta ruta se declara en una sola línea de código en el archivo **routes.rb,** como se observa en la figura 60. En este caso scaffold la genera automáticamente.

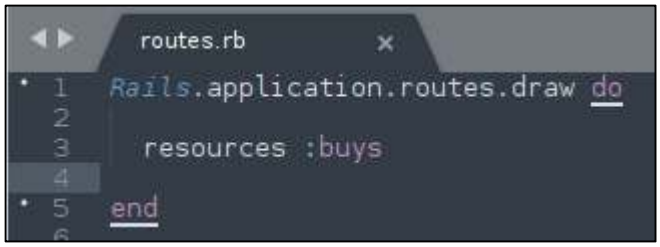

Figura 60 Ruta de recursos

3.1 Escribir el siguiente comando en el terminal y observar las rutas generadas.

```
$ rake routes
```

Observar que se muestran todas las rutas configuradas en el proyecto.

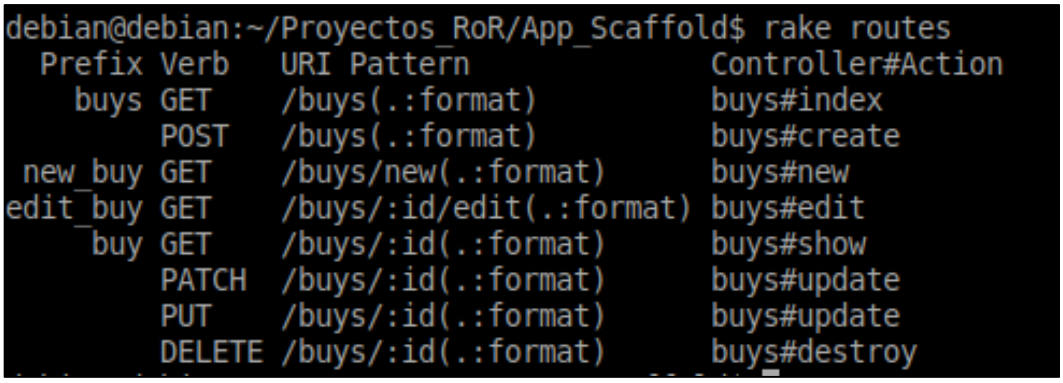

Figura 61 Rutas creadas por scaffold

Cada ruta está compuesta por los siguientes elementos: verbos HTTP (GET, POST, PATCH, PUT, DELETE), path (camino), controlador y método (acción).

En Rails, una ruta proporciona una asignación entre los verbos HTTP y las URL a las acciones del controlador. Por defecto, cada acción también se asigna a operaciones CRUD particulares en la base de datos.

4.Verificar el controlador.

Observar que se han creado los distintos métodos del controlador en el archivo **buys_controller.rb** dentro del directorio (app/controller/).

buys_controller.rb

```ruby
class BuysController < ApplicationController
  before_action :set_buy, only: [:show, :edit, :update, :destroy]

  # GET /buys
  # GET /buys.json
  def index
    @buys = Buy.all
  end

  # GET /buys/1
  # GET /buys/1.json
  def show
  end

  # GET /buys/new
  def new
    @buy = Buy.new
  end

  # GET /buys/1/edit
  def edit
  end

  # POST /buys
  # POST /buys.json
  def create
    @buy = Buy.new(buy_params)

    respond_to do |format|
      if @buy.save
        format.html { redirect_to @buy, notice: 'Buy was successfully created.' }
        format.json { render :show, status: :created, location: @buy }
      else
        format.html { render :new }
        format.json { render json: @buy.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /buys/1
  # PATCH/PUT /buys/1.json
  def update
    respond_to do |format|
      if @buy.update(buy_params)
        format.html { redirect_to @buy, notice: 'Buy was successfully updated.' }
        format.json { render :show, status: :ok, location: @buy }
      else
        format.html { render :edit }
        format.json { render json: @buy.errors, status: :unprocessable_entity }
      end
    end
  end
  def destroy
    @buy.destroy
    respond_to do |format|
      format.html { redirect_to buys_url, notice: 'Buy was successfully destroyed.' }
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_buy
      @buy = Buy.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def buy_params
      params.require(:buy).permit(:category, :description, :amount)
    end
end
```

Figura 62 Métodos del controlador completo

5.Configurar el index generado por el scaffold como la página de inicio de la aplicación.

5.1 Escribir la siguiente línea de código en el archivo config/**routes.rb**.

```
root "buys#index"
```

5.2 Iniciar el servidor.

```
$ rails server
```

5.3 Abrir el navegador y acceder a la dirección.

```
http://localhost:3000/
```

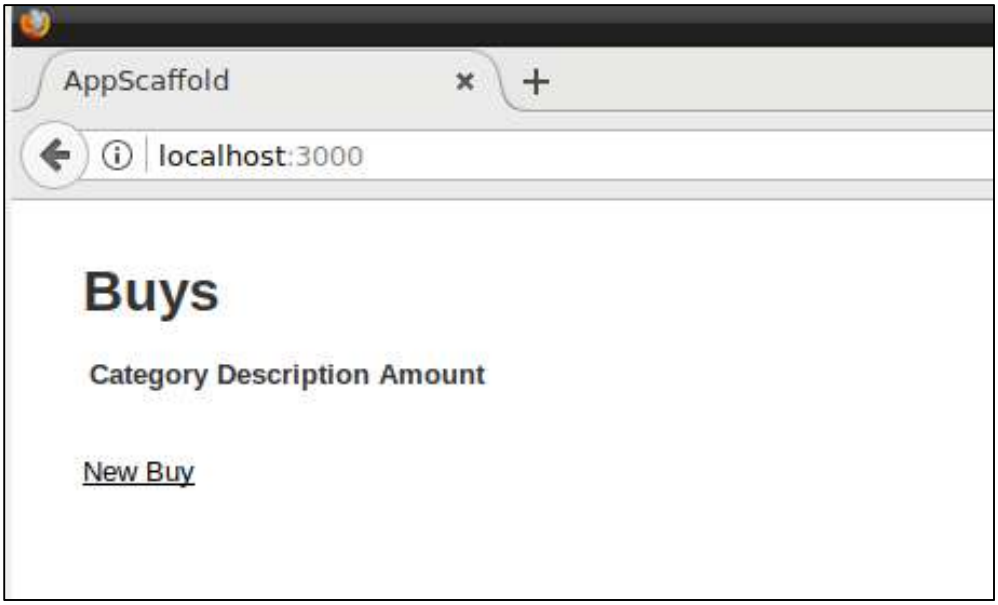

Figura 63 Página principal del proyecto

Se muestra la interfaz vacía, debido a que no se ha creado ningún dato y como se observa en la figura 63, este es el index generado por el scaffold, una vez configurado el archivo routes.rb se estableció como la página principal de la aplicación.

6.Verificar el funcionamiento de las vistas.

6.1 En la vista mostrada anteriormente, presiona new buy, se mostrará el formulario para poder agregar datos a la aplicación.

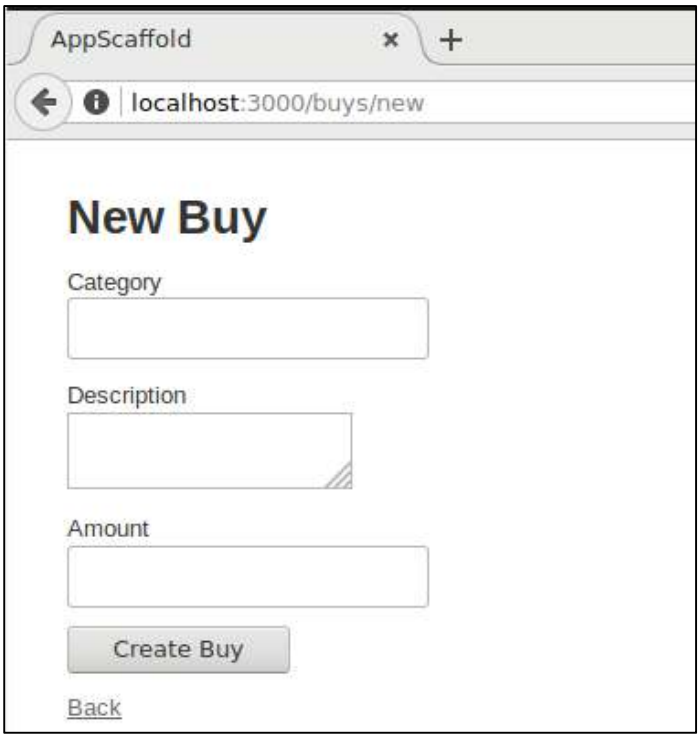

Figura 64 Formulario para crear una nueva compra

6.2 Ingresar una nueva compra llenando el formulario, al presionar el botón **create buy,** el dato se ha creado y almacenado correctamente.

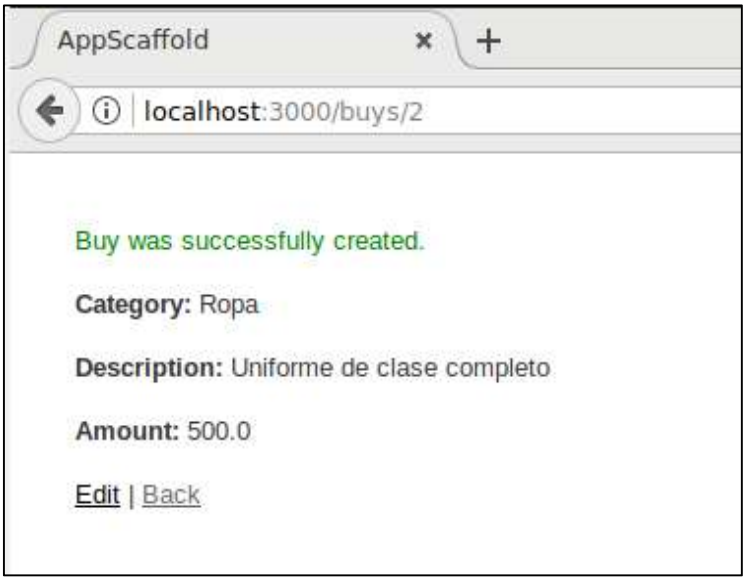

Figura 65 Datos creados correctamente

6.3 Click en **Edit,** podrá observar que permite editar el dato anteriormente almacenado.

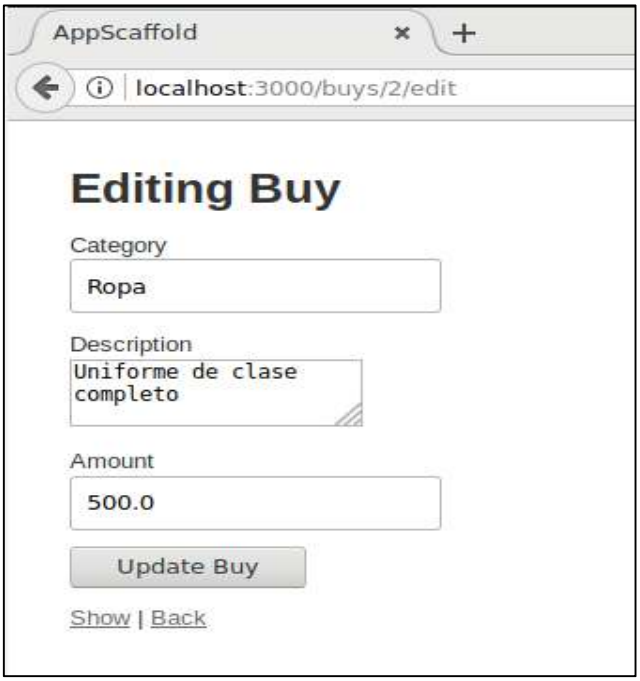

Figura 66 Editando el dato almacenado

6.4 Verificar que se muestran los datos almacenados dentro del index de la aplicación.

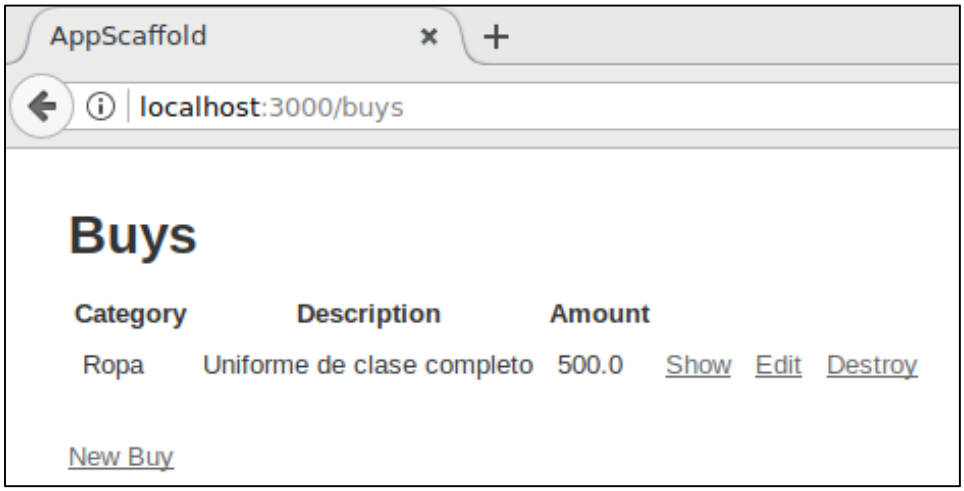

Figura 67 Datos almacenados

Puede interactuar con la aplicación agregando nuevos datos, de manera que pruebe las distintas funcionalidades creadas haciendo uso de scaffold.

7.Validaciones al modelo buy

Si el usuario agrega un nuevo dato en la aplicación, podrá agregar datos vacíos debido a que no se le ha agredo ningún tipo de validaciones al modelo. Una de las ventajas de Rails es permite validar los campos del formulario de una manera más sencilla, en este caso solo se validará que los campos del modelo no estén vacíos para poder agregar un nuevo dato a la aplicación.

7.1 Abrir el archivo /**model/buy.rb** y agregar el siguiente código.

```
validates :category, :description, :amount, presence: true
```

Si ahora intenta agregar un nuevo dato vacío le mostrará un conjunto de mensajes a como se muestra en la siguiente figura.

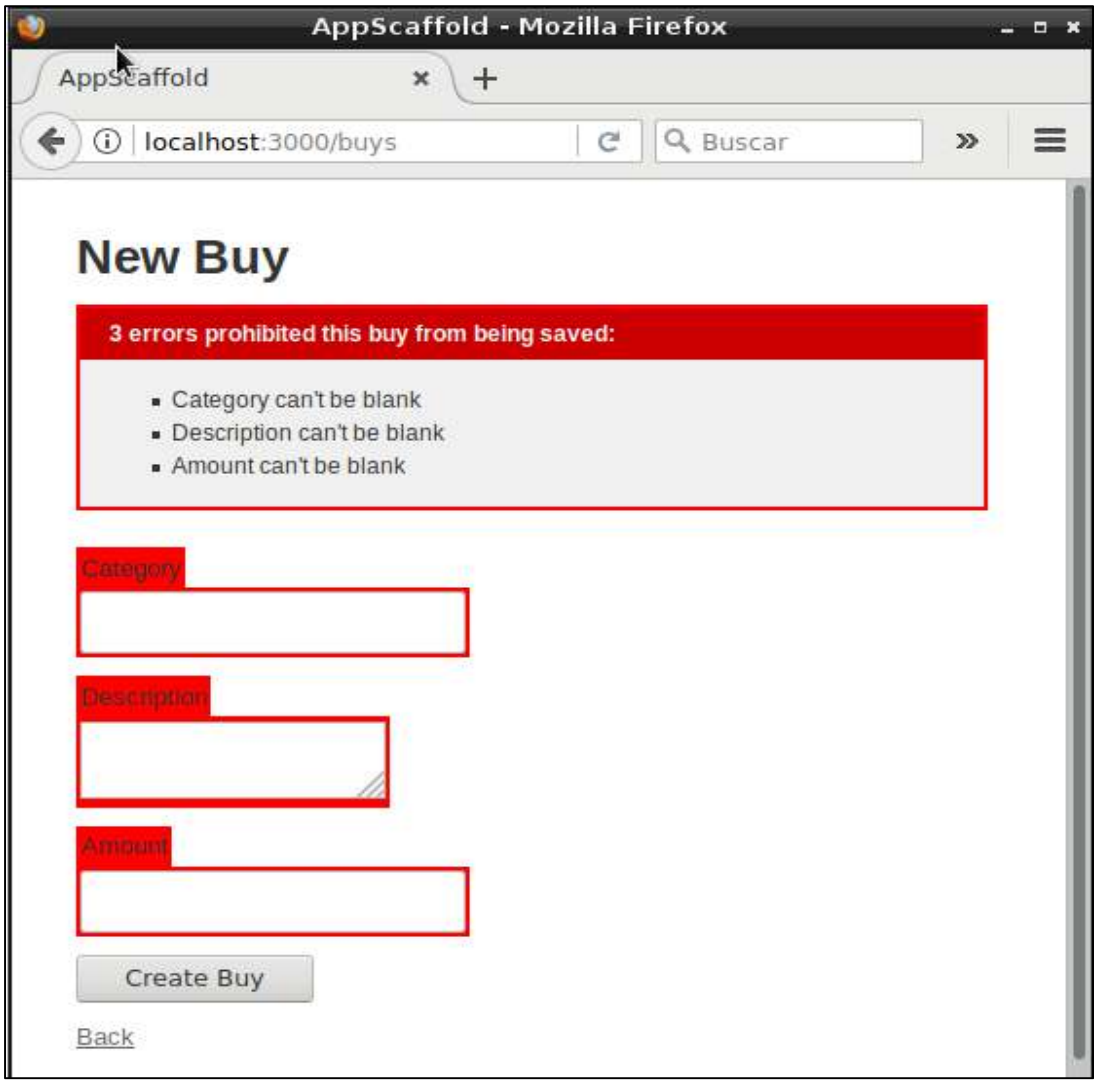

Figura 68 Campos del buy validados

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.
2. Crear un proyecto nuevo, utilice scaffold para generar el código y crear una tabla de nombre **Estudiante** con los campos nombres, apellidos, carrera, carnet, fecha de nacimiento, edad, celular; deberá configurar el archivo routes.rb para que la página principal de la aplicación sea el index generado por el scaffold, validar el campo celular para que solo admita número y que no permita campos vacíos. La aplicación deberá mostrar un formulario parecido al de la **figura 69,** se puede observar en la figura como el framework crea automáticamente las cajas de texto, dependiendo del tipo de dato que se le específica al generar el scaffold, como el campo fecha de nacimiento donde crea un imput tipo date_select en el formulario para seleccionar la fecha de nacimiento del estudiante.

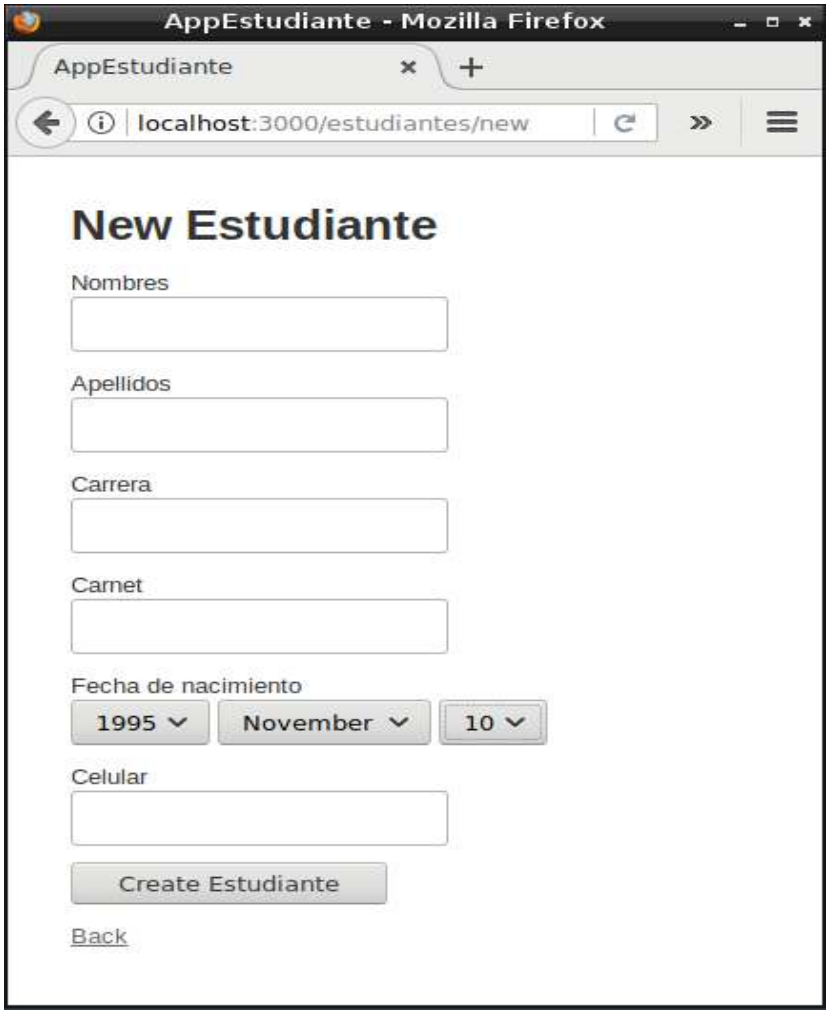

Figura 69 Ejercicio propuesto de scaffold

Guía 7: Ruby on Rails con MYSQL

Introducción

El framework RoR debido a la sencillez de configuración, permite al desarrollador cambiar de motores de base de datos o usar más de uno a la vez sin tener que hacer grandes cambios dentro de un proyecto.

En la presente guía el estudiante aprenderá a conectar Ruby on Rails con el motor de base datos MYSQL usando la gema 'mysql2', esto le sirve de pauta para probar Ruby on Rails con otros motores de base de datos, de igual forma el estudiante podrá aprender a configurar el escenario para hacer uso de gemas utilizadas en las pruebas, se introducirá en el desarrollo haciendo uso de técnicas de desarrollo ágil y conocerá la parte básica en el desarrollo en base a prueba y al comportamiento de una aplicación en Rails.

Objetivos

- Configura Ruby on Rails para usar MYSQL como motor de base de datos.
- Comprobar el funcionamiento de la aplicación usando MYSQL.
- Realizar validaciones de scaffold usando TDD.
- Comprobar el funcionamiento de la aplicación utilizando BDD.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • PhpMyadmin, mysql. • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Ruby on Rails org, RailsGuides, Getting Started with Rails. http://guides.rubyonrails.org/v3.2/getting_started.html
- Elicia. Scaffolding en RoR. https://gist.github.com/Elicia/8530046
- Thoughtbot. Shoulda Matcher. http://matchers.shoulda.io/docs/v3.1.1/

Desarrollo

1.Crear un nuevo proyecto en Rails.

```
$ rails new App_mysql -T
```

Al generar un nuevo proyecto se escribe el –T, para que el framework no configure la herramienta de pruebas, que se instala por defecto al momento de generar el proyecto.

2.Preparar el escenario de Test.

En esta sección se trabajará con la gema rspec y shoulda-matcher para la parte de los tests; para esto, lo primero es configurar el escenario de la aplicación.

2.1 Abrir el archivo **Gemfile** y agregar las siguientes líneas al grupo development, test.

```
gem 'rspec-rails'
gem 'shoulda-matchers'
gem 'capybara'
```

2.2 Ejecutar el siguiente comando en el terminal, para instalar las gemas agregadas anteriormente.

```
$ bundle install
```

2.3 Ejecutar el siguiente comando en el terminal, para crear la base de datos.

```
$rake db:create:all
```

2.4 Ejecutar el comando en el terminal, para generar los archivos de rspec.

```
$rails g rspec:install
```

El comando anterior generará automáticamente los archivos de rspec, junto con el directorio spec dentro de la aplicación.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rails g rspec:install
Running via Spring preloader in process 6433
      create  .rspec
      create  spec
      create  spec/spec_helper.rb
      create  spec/rails_helper.rb
```

Figura 70 Instalar rspec en el proyecto

2.5 Como parte de la configuración hay que agregar las siguientes líneas dentro del archivo **spec/rails_helper.rb** que fue generado anteriormente.

```
require "capybara/rspec"
require 'shoulda/matchers'
```

Estas líneas se deben de agregar en la parte superior, como se muestra en la figura 71.

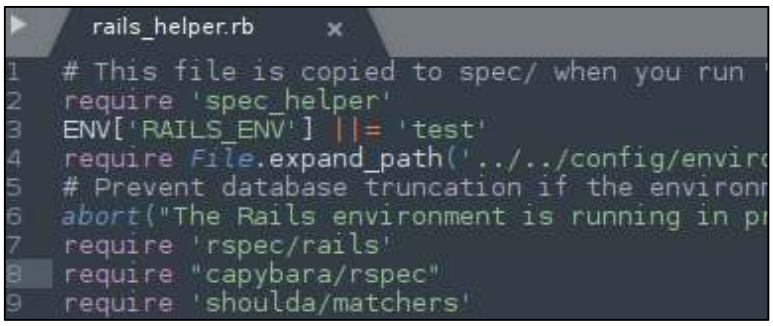

Figura 71 Agregar capybara y shoulda matcher al proyecto

2.6 Como en esta guía solo se harán test para los modelos. Abrir el archivo **Application.rb** que está dentro del directorio config y agregar el siguiente código dentro de la clase applications.

```
config.generators do |g|
  g.test_framework :rspec,
    :fixtures => true,
    :view_specs => false,
    :helper_specs => false,
    :routing_specs => false,
    :controller_specs => false,
    :request_specs => false
  end
```

Si observa el código solo fixtures es igual a true, debido a que rspec se puede utilizar para crear distintos tests dentro de la aplicación ya sea para los controladores, las rutas, los helpers, etc. Como se menciona anteriormente solo se trabajará con algunas pruebas para validar los modelos, y también el comportamiento de la aplicación con respecto a los modelos, es por eso que los demás se configuran como false para que no se generen archivos que se no se utilizarán.

2.7 Por ultimo solo queda escribir en siguiente código en el archivo **/spec/rails_helper.rb** para poder trabajar con la gema shoulda-matcher al momento de crear los tests. Agregarlo al final de todo el código que contiene el archivo.

```
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    # Choose a test framework:
    with.test_framework :rspec
    with.library :rails
  end
end
```

Una vez hecho esto, la aplicación esta lista para poder trabajar los tests con rspec y shoulda-matcher.

3.Trabajando con mysql

El framework RoR trabaja por defecto con sqlite3 para almacenar los datos, pero también permite trabajar con distintos gestores de base de datos, en este caso se trabajará con mysql.

3.1 Abrir el archivo Gemfile y agregar la gema ‘mysql2’.

```
gem 'mysql2'
```

3.2 Desde el terminal instalar la gema de mysql.

```
$ bundle install
```

Una vez con la gema de **mysql** instalada en el proyecto, lo siguiente sería configurar los datos de la base de datos que se creará y donde se guardarán los datos de la aplicación.

4.Configurar la base de datos. Modificar el archivo **config/database.yml** para que quede de la siguiente manera, en la sección **development**.

```
development:
  adapter: mysql2
  username: debian
  password: ruby1
  host: 127.0.0.1
  port: 3306
  database: RoR
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  timeout: 5000
```

Con la configuración anterior se ha establecido mysql como la base de dato de desarrollo y se ha dejado sqlite3 para el entorno de pruebas. Los parámetros que se han establecidos son los siguientes:

- adapter: se cambió a mysql2.
- username: usuario de base de dato.
- password: contraseña de base de dato.
- host: dirección ip del equipo donde está el motor de base dato.
- port: puerto de escucha de motor de base de datos.
- pool: se ha dejado por defecto, este valor define el número de conexiones atendidas al mismo tiempo.
- timeout: tiempo de espera por una conexión.

5. Crear base de datos

5.1 Generar la base de dato.

Se tiene que diferenciar que es lo que se quiere hacer, si crear una nueva base de datos o migrar los datos de un motor de base de datos a otro.

5.2 Crear nueva base de datos.

```
$ rake db:create
```

Puede verificar en el phpmyadmin que se ha creado la base de datos con el nombre **RoR** que se configuró en el archivo **database.yml**.

5.3 Migrar los datos, para ello la base de datos ya debe existir.

```
$ rake db:migrate
```

6.Para probar el comportamiento se está usando scaffold para generar código.

```
$ rails g scaffold User name:string password:string email:string
programador:boolean birthday:date work_time:time
```

7.Migrar los datos generados con scaffold y la base de datos de los tests.

```
$ rails db:migrate RAILS_ENV=test
```

8.Migrar los datos de la parte de desarrollo.

```
$ rails db:migrate RAILS_ENV=development
```

9.Ir al directorio db y abrir el archivo schema.rb que se ha generado anteriormente, el cual contiene el esquema actual de la base de datos donde se almacenarán los datos del usuario.

```
ActiveRecord::Schema.define(version: 20180227213204) do

  create_table "users", force: :cascade, options: "ENGINE=InnoDB DEFAULT CHARSET=utf8" do |t|
    t.string "name"
    t.string "password"
    t.string "email"
    t.boolean "programador"
    t.date "birthday"
    t.time "work_time"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

end
```

Figura 72 Esquema de base de datos del proyecto

10.Si observa, al momento de generar el código con scaffold, se ha creado un directorio models dentro del directorio **spec,** verificar como se muestra a continuación.

```
debian@debian:~/Proyectos_RoR/App_mysql$ ls spec/
models  rails_helper.rb  spec_helper.rb
```

Figura 73 Archivos del directorio spec

10.1 Dentro de este directorio **model** se ha creado el archivo **user_spec.rb** que se necesita para poder hacer test a los modelos. Eliminar o comentar la siguiente línea, que no se ocupará.

```
4
5
6     pending "add some examples to (or delete) #{__FILE__}"
7
```

Figura 74 Eliminar la línea del archivo user_spec.rb

11.Para empezar a crear los tests y validar el modelo user.

11.1 Agregar el código al archivo spec/**models/user_spec.rb** para hacer test al campo name y agregar validaciones.

```
context "name" do
    it { should validate_presence_of :name }
    it { should validate_length_of(:name).is_at_most(15) }
    it { should_not allow_value("123123").for(:name) }
    it { should allow_value("abcd").for(:name) }
  end
```

11.2 Ir al terminal y ejecutar el siguiente comando para hacer los tests.

```
$ rspec spec/models/user_spec.rb
```

Este comando ejecutará los tests que fueron agregados al archivo **user_spec.rb** y mostrará un conjunto de mensajes en rojo, que indican que es lo que se debe hacer para que los test pasen.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/models/user_spec.rb
FFF.

Failures:

  1) User name should validate that :name cannot be empty/falsy
     Failure/Error: it { should validate_presence_of :name }

       User did not properly validate that :name cannot be empty/falsy.
         After setting :name to ‹nil›, the matcher expected the User to be
         invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:7:in `block (3 levels) in <top (required)>'

  2) User name should validate that the length of :name is at most 15
     Failure/Error: it { should validate_length_of(:name).is_at_most(15) }

       User did not properly validate that the length of :name is at most 15.
         After setting :name to ‹"xxxxxxxxxxxxxxx"›, the matcher expected the
         User to be invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:8:in `block (3 levels) in <top (required)>'

  3) User name should not allow :name to be ‹"123123"›
     Failure/Error: it { should_not allow_value("123123").for(:name) }

       After setting :name to ‹"123123"›, the matcher expected the User to be
       invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:9:in `block (3 levels) in <top (required)>'

Finished in 0.04505 seconds (files took 2.48 seconds to load)
4 examples, 3 failures
```

Figura 75 Test para validar el campo name del modelo

12. Ir al archivo **user.rb** que se encuentra en el directorio **app/model** y agregar el siguiente código dentro de la clase.

```
validates_presence_of :name, presence: true
validates_format_of :name, :with => /\A[a-zA-Z]+\z/
validates :name, length: { maximum: 15 },on: :create
```

Con esto valida que el campo name del formulario user no quede vacío, que solo permita letras y que contenga un máximo de 15 caracteres.

12.1 Si ejecuta el comando de los tests nuevamente, podrá observar que ya no se muestran los mensajes en rojo.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/models/user_spec.rb
....

Finished in 0.04111 seconds (files took 2.76 seconds to load)
4 examples, 0 failures
```

Figura 76 Test del campo name en verde

13. Agregar siguiente código al archivo spec/**models/user_spec.rb** para hacer test al campo password del modelo.

```
context "Password" do
  it { should validate_presence_of :password }
  it { should validate_length_of(:password).is_equal_to(8) }
end
```

Al ejecutar el test, mostrará nuevamente los mensajes en color rojo.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/models/user_spec.rb
....FF

Failures:

  1) User Password should validate that :password cannot be empty/falsy
     Failure/Error: it { should validate_presence_of :password }

       User did not properly validate that :password cannot be empty/falsy.
         After setting :password to ‹nil›, the matcher expected the User to be
         invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:14:in `block (3 levels) in <top (required)>'

  2) User Password should validate that the length of :password is 8
     Failure/Error: it { should validate_length_of(:password).is_equal_to(8) }

       User did not properly validate that the length of :password is 8.
         After setting :password to ‹"xxxxxxx"›, the matcher expected the User
         to be invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:15:in `block (3 levels) in <top (required)>'

Finished in 0.06518 seconds (files took 2.46 seconds to load)
6 examples, 2 failures
```

Figura 77 Test del campo password en rojo

14. Agregar las validaciones en el archivo **app/model/user.rb,** correspondientes para el campo password del modelo.

```
validates_presence_of :password, presence: true
validates_length_of :password, is: 8
```

Este código lo que hace es limitar el campo password a 8 caracteres y no permitir que esté vacío.

14.1 Una vez escrito el código, volver a ejecutar el test para poder observar que ahora muestra el mensaje en verde, indicando que ha pasado los test a como se muestra en la siguiente figura.

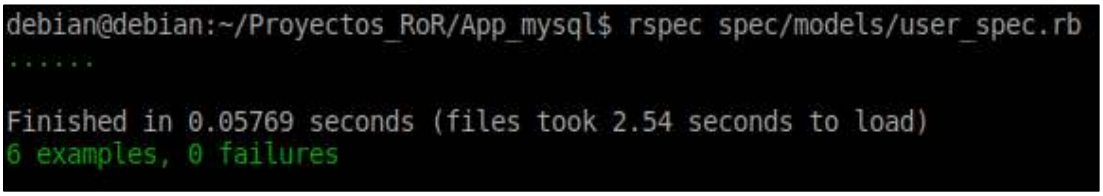

Figura 78 Test del campo password en verde

15. El ultimo test será para el campo email del modelo, agregar el código al archivo spec/models/**user_spec.rb**.

```
context "email" do
  it { should validate_presence_of :email }
  it { should validate_uniqueness_of :email }
  it { should_not allow_value("text").for(:email) }
  it { should allow_value("a@b.com").for(:email) }
end
```

Con estos tests, el campo email del modelo tiene que tener las validaciones para que la aplicación funcione correctamente y no acepte datos erróneos. La primera sentencia **it** le dice que no puede estar el campo vacío, la siguiente sentencia valida que el email sea único, luego se evalúa el formato del email, no puede ser solo texto y debe poseer el carácter @ para que lo tome como válido.

16.Ejecutar el test en el terminal y mostrará los mensajes en rojo nuevamente.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/models/user_spec.rb
......FFF.

Failures:

  1) User email should validate that :email cannot be empty/falsy
     Failure/Error: it { should validate_presence_of :email }

       User did not properly validate that :email cannot be empty/falsy.
         After setting :email to ‹nil›, the matcher expected the User to be
         invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:20:in `block (3 levels) in <top (required)>'

  2) User email should validate that :email is case-sensitively unique
     Failure/Error: it { should validate_uniqueness_of :email }

       User did not properly validate that :email is case-sensitively unique.
         After taking the given User, setting its :email to ‹"an arbitrary
         value"›, and saving it as the existing record, then making a new User
         and setting its :email to ‹"an arbitrary value"› as well, the matcher
         expected the new User to be invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:21:in `block (3 levels) in <top (required)>'

  3) User email should not allow :email to be ‹"text"›
     Failure/Error: it { should not allow_value("text").for(:email) }

       After setting :email to ‹"text"›, the matcher expected the User to be
       invalid, but it was valid instead.
     # ./spec/models/user_spec.rb:22:in `block (3 levels) in <top (required)>'

Finished in 0.11257 seconds (files took 2.46 seconds to load)
10 examples, 3 failures
```

Figura 79 Test del campo email en rojo

17.Agregar las validaciones al modelo user en el archivo **app/model/user.rb.**

```
validates_presence_of :email, presence: true
validates_uniqueness_of :email
validates_format_of :email, with: /@/
```

Ejecutar los tests y observar que finalmente todos los tests pasan y ya se han validado los campos del modelo user.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/models/user_spec.rb
..........

Finished in 0.13069 seconds (files took 2.45 seconds to load)
10 examples, 0 failures
```

Figura 80 Test del campo email en verde

18. Hasta el momento no se ha trabajado con la parte web del framework; como se sabe, cuando se utiliza scaffold para generar el código, este ya ha generado las vistas para el modelo user.

18.1 Ir al directorio **view** y observar que existen todas las vistas necesarias para el correcto funcionamiento del modelo user.

Figura 81 Archivo de las vistas generados por scaffold

18.2 El siguiente paso es configurar el index de la aplicación, para eso ir al archivo **config/routes.rb** y escribir el código para el index sea la página principal del proyecto. El archivo routes quedará como se muestra en la figura 82.

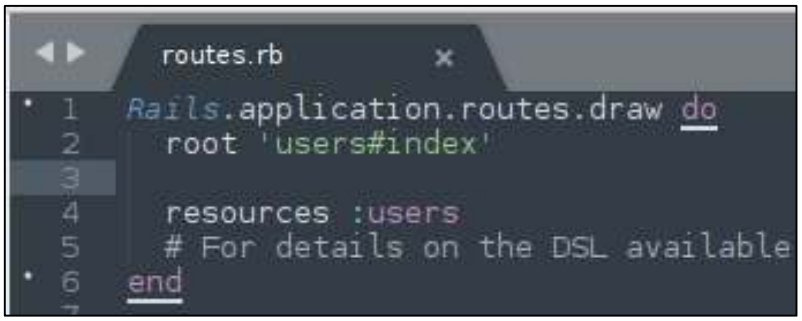

Figura 82 Archivo routes.rb del proyecto

19. Para probar la aplicación, debe arrancar el servidor en el terminal, ejecutando el siguiente comando.

```
$ rails s
```

20.Prueba en la aplicación web

20.1 Abrir el navegador en la dirección localhost:3000

Figura 83 Index de la aplicación

Como se observa en la figura anterior, ha cargado el archivo index como la página principal de la aplicación, de igual forma se observa que no contiene ningún dato almacenado hasta el momento.

20.2 Para crear nuevo usuario y agregar datos, presionar click en **New User** y mostrará el formulario para ingresar los datos.

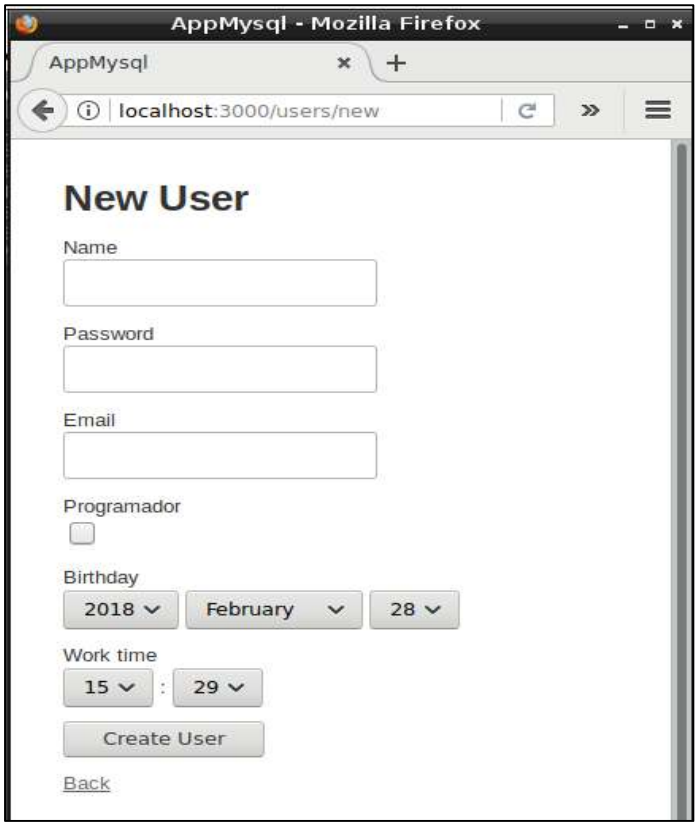

Figura 84 Formulario para agregar un nuevo usuario

20.3 Una vez abierto el formulario, se podrá poner a prueba las validaciones que se agregaron anteriormente, si presiona el botón create user observará que muestra unos mensajes en relación a las validaciones de los campos del modelo user. Estos mensajes son generador automáticamente por el framework.

Figura 85 Validaciones en la web

20.4 Al llenar los datos del formulario correctamente, ya permite guardar los datos.

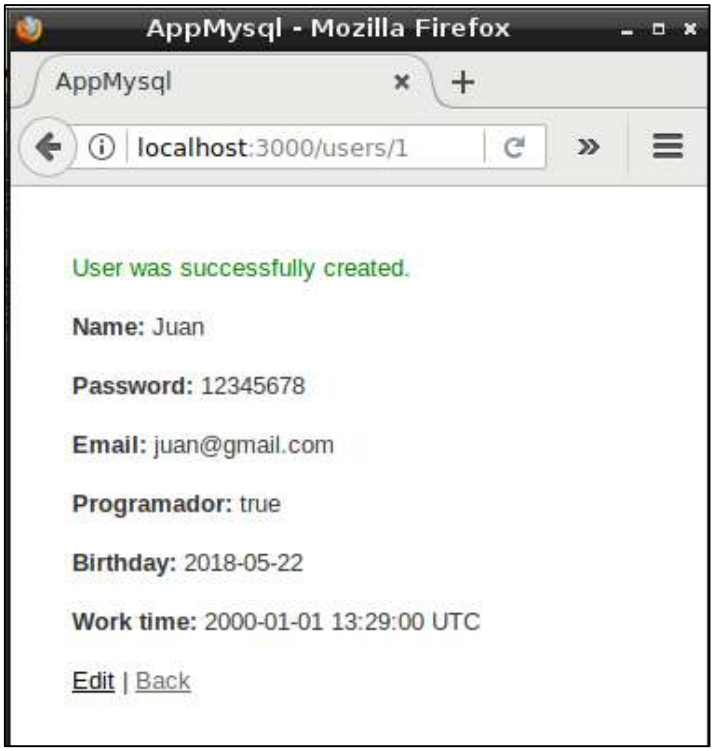

Figura 86 Usuario creado correctamente

20.5 Verificar que se haya creado correctamente el usuario en mysql, abrir el phpmyadmin y hacer un select a la tabla user, de la base de datos en la que se está trabajando.

id	name	password	email	programador	birthday	work_time
1	Juan	12345678	juan@gmail.com	1	2018-05-22	13:29:00

Figura 87 Datos almacenados en phpmyadmin

21.Pruebas por comportamiento BDD.

Para trabajar con el comportamiento de la aplicación y comprobar que las validaciones anteriormente agregadas se comportan de la manera correcta sin necesidad de hacer uso del navegador web, se puede hacer uso de la gema **capybara**, que ayuda a probar aplicaciones web simulando como un usuario real interactúa con la aplicación, esta trabaja en conjunto con rspec.

21.1 Para hacer testing primero crear el directorio en donde se guardan los features.

```
$ rails g rspec:feature user
```

Este comando creará un nuevo directorio y un archivo para test de los features dentro del directorio **spec**.

```
debian@debian:~/Proyectos_RoR/App_RoR$ rails g rspec:feature staticpages
Running via Spring preloader in process 3661
      create  spec/features/staticpages_spec.rb
```

Figura 88 Generar features para el test

21.2 Abrir el archivo .rspec que está en la raíz de la aplicación y agregar la siguiente línea.

```
-f documentation
```

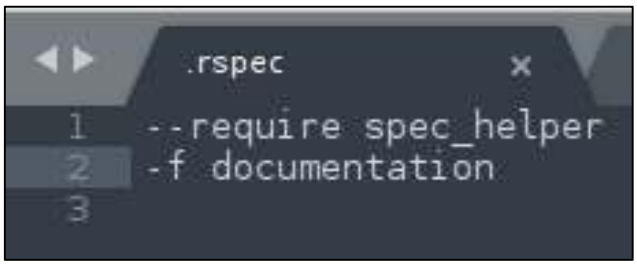

Figura 89 Muestra del archivo .rspec

21.3 Abrir el archivo **spec/features/user_spec.rb** y agregar el siguiente código, para hacer test al comportamiento al momento de crear un nuevo usuario.

```
feature "User want to create a new user successfully" do
scenario "Create a new user account" do
   visit new_user_path
   fill_in "Name", :with => "Pedro"
   fill_in "Password", :with => "12345678"
   fill_in "Email", :with => "pedro@gmail.com"
   click_button "Create User"
   expect(page).to have_text("User was successfully created")
 end
end
```

En esta parte se interactúa con la aplicación como si se hiciera uso del navegador web, en las líneas 4, 5 y 6 hace referencia a los campos de texto que se ha validado, en este caso se introducen datos como nombre password y email de manera correcta, esto hará que el test pase sin ningún problema.

21.4 Ejecutar el test con rspec, escribiendo el siguiente comando en el terminal.

```
$ rspec spec/features/users_spec.rb
```

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/features/users_spec.rb

Users
  User want to create a new user successfully
    Create a new user account

Finished in 0.33531 seconds (files took 2.25 seconds to load)
1 example, 0 failures
```

Figura 90 Test al comportamiento en verde

Al ejecutar el test no muestra ningún mensaje en rojo, debido a que los datos en el escenario creado en el archivo **spec/features/users_spec.rb** fueron pasado de manera correcta, respetando las validaciones anteriormente agregadas al modelo.

22. Otra prueba que se puede realizar en relación al comportamiento, es cuando el usuario presione click en el link back se debe mostrar la página principal de la aplicación donde se muestran todos los usuarios almacenados.

22.1 Copiar el siguiente código en el archivo spec/**features**/**user_spec.rb**

```
feature "User want to see all users created" do
scenario "User is on index page" do
   visit new_user_path
   click_link "Back"
   expect(page).to have_content("Users")
 end
end
```

Lo que hace es simular el compartimiento de la página cuando el usuario presiona click en link Back y regresa a la página principal de la aplicación.

22.2 Ejecutar el test nuevamente y podrá observar que se ejecuta sin ningún problema.

```
debian@debian:~/Proyectos_RoR/App_mysql$ rspec spec/features/users_spec.rb

Users
  User want to create a new user successfully
    Create a new user account
  User want to see all users created
    User is on index page

Finished in 0.36305 seconds (files took 2.27 seconds to load)
2 examples, 0 failures
```

Figura 91 Test de comportamiento completos

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.
2. Crear una aplicación en Rails utilizando mysql; configurar una base de datos de nombre Datos_estuadiente y utilice scaffold para generar el código y crear una tabla **Estudiante** con los campos nombres, apellidos, carrera, carnet y validar todos los campos necesarios.

Extra: para las validaciones hacer uso de TDD, haciendo uso las distintas gemas anteriormente utilizadas en la guías. Y agregar código css a las vistas para obtener un mejor diseño.

Guía 8: Ruby on Rails con TDD

Introducción

Test Driver Development (TDD) conocido como desarrollo en base a pruebas, es una de las principales técnicas de desarrollo ágil más utilizadas para trabajar con Ruby on Rails, se centra básicamente en escribir un pequeño test para un código que no existe todavía.

En esta guía el estudiante configurará un proyecto para poder trabajar con el desarrollo en base a pruebas y poder realizar una aplicación básica tomada de (Hartl, s.f.), se instalará un conjunto de gemas necesarias para el correcto funcionamiento del proyecto, dentro de las cuales se podría mencionar Rspec que es la gema principal para realizar las pruebas. De igual forma se introducirá más en el desarrollo de aplicaciones web utilizando el framework Ruby on Rails.

Objetivos

- Configurar el escenario para hacer test.
- Hacer uso de la herramienta Rspec.
- Crear una aplicación en RoR

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Sublimetext • Rspec	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Michael Hartl. Ruby on Rails tutorial (Rail5), Learn Web Development with Rails. https://www.railstutorial.org/book/
- Rspec. Rspec-rails. https://github.com/rspec/rspec-rails
- Rspec expectations. https://relishapp.com/rspec/rspec-expectations/docs

Desarrollo

1. Generar un nuevo proyecto.

```
$ rails new App_RoR -T
```

Al generar un nuevo proyecto se escribe el –T, para que el framework no configure la herramienta de pruebas, que se instala por defecto al momento de generar el proyecto en Rails.

1.1. Agregar las gemas que se utilizarán en el proyecto, dentro grupo development, test.

```
gem "rspec-rails", "~> 3.7"
gem 'capybara'
gem 'database_cleaner'
```

1.2. El grupo development, test quedará como se muestra en la siguiente figura.

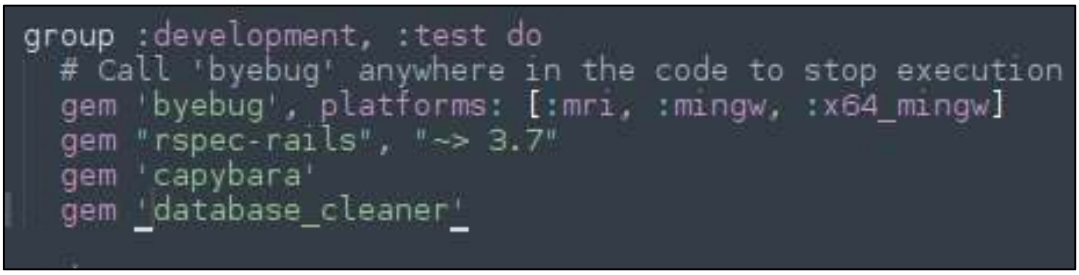

Figura 92 Agregar gemas en el grupo development, test. Gemfile

1.3. Ejecutar el siguiente comando en el terminal para instalar las gemas.

```
$ bundle install
```

1.4. Ejecutar el siguiente comando para crear los directorios que rspec necesita para trabajar.

```
$ rails g rspec:install
```

2. Configuración de Rspec.

2.1. Abrir el archivo config/**application.rb** y agregar el siguiente código dentro de la clase, en el mismo archivo.

```
config.generators do |g|
    g.test_framework :rspec,
        fixtures: true,
        view_specs: false,
        helper_specs: false,
        routing_specs: false,
        controller_specs: true,
        request_specs: false
end
```

2.2. Abrir el archivo rails_helper.rb y agregar debajo de la línea **require 'rspec/rails'**.

```
require "capybara/rspec"
```

2.3. Ejecutar el comando y verificar que no muestra ningún mensaje en rojo.

```
$ rspec
```

3. Static pages

3.1. Para crear páginas estáticas escribir los siguiente en el terminal.

```
$ rails generate controller StaticPages home help
```

3.2. Verificar la configuración del archivo routes.rb.

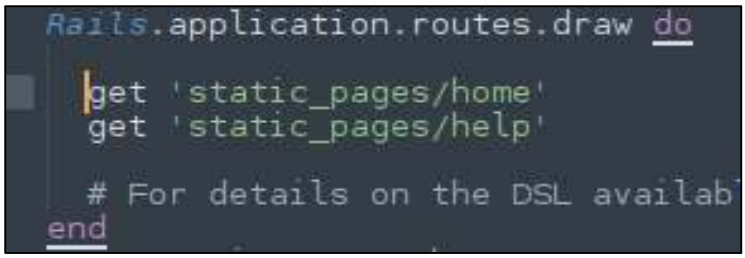

Figura 93 Archivo routes.rb

3.3. Verificar que se han creado las vistas en **app/views/static_pages/.**

4. Modificar las vistas.

4.1. Agregar al archivo **home.html.erb.**

```
<h1>Sample App</h1>
<p>
  This is the home page for the
  <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
  sample application.
</p>
```

4.2. Modificar el archivo **help.html.erb.**

```
<h1>Help</h1>
<p>
  Get help on the Ruby on Rails Tutorial at the
  <a href="http://www.railstutorial.org/help">Rails Tutorial help page</a>.
  To get help on this sample app, see the
  <a href="http://www.railstutorial.org/book"><em>Ruby on Rails Tutorial</em>
  book</a>.
</p>
```

5. Hacer testing con rspec.

5.1. El primer paso será ejecutar el comando db:migrate en el terminal para crear el esquema de pruebas con el que se ejecutarán los tests.

```
$ rails db:migrate
```

5.2. Probar hacer testing escribiendo en el terminal el comando.

```
$ rspec spec/controllers/static_pages_controller_spec.rb
```

```
debian@debian:~/Proyectos_RoR/App_TDD$ rspec spec/controllers/static_pages_controller_spec.rb
..

Finished in 0.02432 seconds (files took 4.42 seconds to load)
2 examples, 0 failures
```

Figura 94 Primer test

5.3. Agregar siguiente código dentro del archivo **static_pages_controller_spec.rb** del directorio **spec/controllers/.**

```
describe "GET #about" do
  it "returns http success" do
    get :about
    expect(response).to have_http_status(:success)
  end
end
```

5.4. Ejecutar el comando rspec y ver que muestra el test en rojo.

```
$ rspec spec/controllers/static_pages_controller_spec.rb
```

```
Failures:

  1) StaticPagesController GET #about returns http success
     Failure/Error: get :about

     ActionController::UrlGenerationError:
       No route matches {:action=>"about", :controller=>"static_pages"}
     # ./spec/controllers/static_pages_controller_spec.rb:21:in `block (3 level
s) in <top (required)>'

Finished in 0.03441 seconds (files took 5.03 seconds to load)
3 examples, 1 failure

Failed examples:

rspec ./spec/controllers/static_pages_controller_spec.rb:20 # StaticPagesContro
ller GET #about returns http success
```

Figura 95 Nuevo test en rojo

5.5. Editar el archivo routes.rb y agregar lo siguiente.

```
get  'static_pages/about'
```

5.6. Ejecutar el test de nuevo.

El mensaje que muestra, es debido a que no existe la acción about que intenta alcanzar el método get.

```
Failures:

  1) StaticPagesController GET #about returns http success
     Failure/Error: get :about

     AbstractController::ActionNotFound:
       The action 'about' could not be found for StaticPagesController
     # ./spec/controllers/static_pages_controller_spec.rb:21:in `block (3 levels) in <top (required)>'

Finished in 0.04139 seconds (files took 4.42 seconds to load)
3 examples, 1 failure

Failed examples:

rspec ./spec/controllers/static_pages_controller_spec.rb:20 # StaticPagesController GET #about returns http success
```

Figura 96 Ejecución del nuevo test; no existe la acción about

5.7. Agregar el método en el archivo **controllers/static_pages_controller.rb** y luego ejecutar el test.

```
def about
end
```

```
Failures:

  1) StaticPagesController GET #about returns http success
     Failure/Error: get :about

     ActionController::UnknownFormat:
       StaticPagesController#about is missing a template for this request format and va
riant.

       request.formats: ["text/html"]
       request.variant: []

       NOTE! For XHR/Ajax or API requests, this action would normally respond with 204
No Content: an empty white screen. Since you're loading it in a web browser, we assume
that you expected to actually render a template, not nothing, so we're showing an error
 to be extra-clear. If you expect 204 No Content, carry on. That's what you'll get from
 an XHR or API request. Give it a shot.
     # ./spec/controllers/static_pages_controller_spec.rb:21:in `block (3 levels) in <t
op (required)>'

Finished in 0.0522 seconds (files took 4.76 seconds to load)
3 examples, 1 failure

Failed examples:

rspec ./spec/controllers/static_pages_controller_spec.rb:20 # StaticPagesController GET
```

Figura 97 Ejecución del nuevo test; no existe la vista

Como se observa en la figura anterior, el mismo test dice el error y que necesita hacer para poder pasar el test y que no muestre los mensajes en color rojo, en este caso no encuentra la vista para la acción creada anteriormente.

5.8. Crear una vista para ese método. Crear el archivo **about.html.erb** en el directorio app/**views/static_pages**/ y agregar el código html.

```
<h1>About</h1>
<p>
  The <a href="http://www.railstutorial.org/"><em>Ruby on Rails
  Tutorial</em></a> is a
  <a href="http://www.railstutorial.org/book">book</a> and
  <a href="http://screencasts.railstutorial.org/">screencast series</a>
  to teach web development with
  <a href="http://rubyonrails.org/">Ruby on Rails</a>.
  This is the sample application for the tutorial.
</p>
```

5.9. Para comprobar que todo funciona, ejecutar el test nuevamente.

```
debian@debian:~/Proyectos_RoR/App_RoR$ rspec spec/controllers/static_pages_controller_spec.rb
...

Finished in 0.0328 seconds (files took 4.57 seconds to load)
3 examples, 0 failures
```

Figura 98 Nuevo test en verde

6. Configurar el título de la página.

6.1. Para que el título de la página cambie por cada una de las vistas a la que se accede en la aplicación, copiar el código en el archivo app/helpers/**application_helper.rb** dentro del módulo.

```
def full_title(page_title = '')
  base_title = "Rails_App"
  if page_title.empty?
    base_title
  else
    page_title + " | " + base_title
  end
end
```

Este método se agrega en los **helpers** debido a que es un método que será a utilizado por las distintas vistas de la aplicación, a diferencia de los métodos creados en los controladores que solo son accedidos por cada una de las vistas respectivas a cada método.

6.2. En el archivo app/**views/layouts/application.html.erb** modificar la línea de la etiqueta **tittle** por la siguiente.

```
<title><%= full_title(yield(:title)) %></title>
```

Deberá quedar a como se muestra en la figura 99.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title><%= full_title(yield(:title)) %></title>
```

Figura 99 Encabezado de la aplicación

7. Modificar los archivos de las vistas.

7.1. En el archivo **help.html.erb**, agregar la siguiente línea en la parte superior del código.

```
<% provide(:title, "Help") %>
```

7.2. En el archivo **about.html.erb**, agregar la siguiente línea en la parte superior de todo el código.

```
<% provide(:title, "About") %>
```

7.3. Navegar en la dirección http://localhost:3000/static_pages/about y verificar que cambia el título de la página.

Figura 100 Página about

7.4. Navegar en la dirección http://localhost:3000/static_pages/help y verificar que cambia el título de la página.

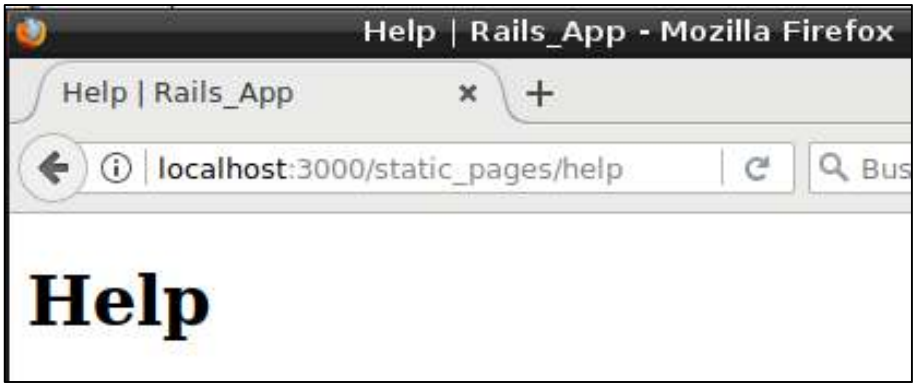

Figura 101 Página help

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.
2. Copiar el código del recuadro, en el archivo **static_pages_controller_spec.rb** del directorio **spec/controllers/.**

```
describe "GET #contact" do
  it "returns http success" do
    get :contact
    expect(response).to have_http_status(:success)
  end
end
```

3. Ejecutar el test y deberá mostrar los siguientes mensajes.

```
debian@debian:~/Proyectos_RoR/App_RoR$ rspec spec/controllers/static_pages_controller_spec.rb
...F

Failures:

  1) StaticPagesController GET #contact returns http success
     Failure/Error: get :contact

     ActionController::UrlGenerationError:
       No route matches {:action=>"contact", :controller=>"static_pages"}
     # ./spec/controllers/static_pages_controller_spec.rb:28:in `block (3 levels) in <top (required)>'

Finished in 0.03702 seconds (files took 4.57 seconds to load)
4 examples, 1 failure

Failed examples:

rspec ./spec/controllers/static_pages_controller_spec.rb:27 # StaticPagesController GET #contact returns http success
```

Figura 102 Test para la página contact

4. Realizar los mismos pasos que realizo con la página **about** y hacer que el test pase.

```
debian@debian:~/Proyectos_RoR/App_RoR$ rspec spec/controllers/static_pages_controller_spec.rb
....

Finished in 0.04033 seconds (files took 4.6 seconds to load)
4 examples, 0 failures
```

Figura 103 Test en color verde de la página contact

5. Navegar en la dirección http://localhost:3000/static_pages/contact.

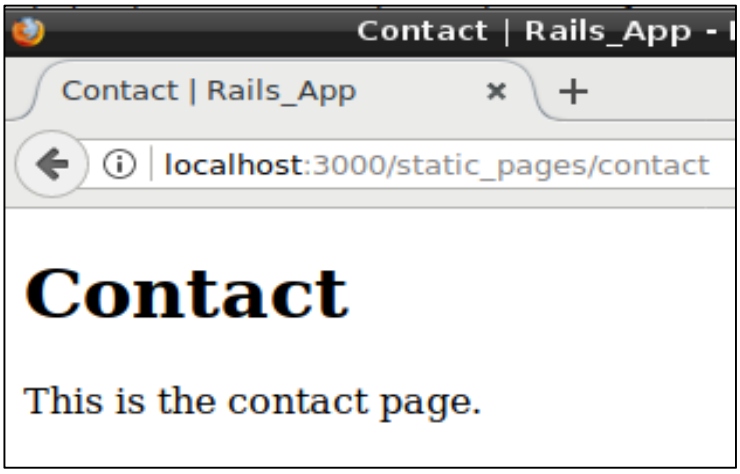

Figura 104 Página contact

6. Configurar el archivo config/**routes.rb,** para que la página de inicio del proyecto sea el archivo home que se encuentra dentro del directorio **static_pages**, realizar la prueba navegando en la dirección **localhost:3000,** como se muestra en la figura 105.

Figura 105 Index de la aplicación

Guía 9: Integración bootstrap en RoR.

Introducción

En esta guía el estudiante podrá integrar una nueva gema llamada bootstrap al proyecto en Rails, utilizada principalmente para el diseño en las vistas de la aplicación, que también servirá para integrar código css a todas las vistas para poder obtener un mejor diseño. En continuación con la guía anterior podrá familiarizarse más con el lenguaje ERB que utiliza el framework en las vistas y conocer un poco sobre Sass, que es un lenguaje que mejora a lo que es css, permitiendo anidar o unir código que tiene relación entre sí.

Objetivos

- Instalar la gema bootstraap.
- Integrar y configurar bootstrap al proyecto en Rails.
- Agregar css a las vistas.
- Conocer acerca de Sass.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Michael Hartl. Ruby on Rails tutorial (Rail5), Learn Web Development with Rails. https://www.railstutorial.org/book/
- Twitter, Inc. bootstrap. https://rubygems.org/gems/bootstrap
- Hampton Catlin. Sass Basics. http://sass-lang.com/guide

Desarrollo

1. En continuación con el proyecto de la guía 8, empezar a trabajar con las vistas. El primer paso es modificar el archivo **application.html.erb** dentro del directorio (app/**views/layouts/**).

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= full_title(yield(:title)) %></title>
    <%= csrf_meta_tags %>
    <%= stylesheet_link_tag    'application', media: 'all',
                                'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
    <%= render 'layouts/shim' %>
  </head>
  <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
      <%= render 'layouts/footer' %>
    </div>
  </body>
</html>
```

El código anterior se puede observar que se hace uso del método **render** para mostrar cada uno de los archivos a los cuales se dirige, para que esto funcione y pueda llamar a los archivos, se debe de crear lo archivos conocidos como **PARTIAL.**

2. Parciales (PARTIAL).

El trabajo de la etiqueta render es buscar los archivos **Partial** dentro del directorio layouts, el siguiente paso será crear cada uno de los archivos dentro del directorio antes mencionado, hay que tomar en cuenta que los archivos Partial siempre inician con guión bajo (_), y esto hace posible identificar los Partial en un directorio.

2.1. Crear un archivo llamado **_shim.html.erb,** dentro del directorio app/views/layouts/ y agregar el siguiente código.

```
<!--[if lt IE 9]>
  <script src="//cdnjs.cloudflare.com/ajax/libs/html5shiv/r29/html5.min.js">
  </script>
<![endif]-->
```

2.2. Crear un archivo llamado **_header.html.erb,** en el mismo directorio que el anterior y agregar el siguiente código.

```
<header class="navbar navbar-fixed-top navbar-inverse">
  <div class="container">
    <%= link_to "sample app", '#', id: "logo" %>
    <nav>
      <ul class="nav navbar-nav navbar-right">
        <li><%= link_to "Home", '#' %></li>
        <li><%= link_to "Help", '#' %></li>
        <li><%= link_to "Contact", '#' %></li>
      </ul>
    </nav>
  </div>
</header>
```

2.3. Y por último crear el archivo **_footer.html.erb** y agregar el siguiente código.

```
<footer class="footer">
  <small>
    The <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
    by <a href="http://www.michaelhartl.com/">Michael Hartl</a>
  </small>
  <nav>
    <ul>
      <li><%= link_to "About", '#' %></li>
      <li><a href="http://news.railstutorial.org/">News</a></li>
    </ul>
  </nav>
</footer>
```

En la creación de los archivos anteriores, se hace uso de etiquetas propias del lenguaje ERB, por ejemplo: **link_to,** que crea un hipervínculo hacia algún sitio. De igual forma se hace uso de clases propias de bootstrap que son las utilizadas para agregar estilos a la aplicación.

3. Editar el archivo el app/views/**Static_pages/home.html.erb** y agregar el código.

```
<div class="center jumbotron">
  <h1>Welcome to the Sample App</h1>

  <h2>
    This is the home page for the
    <a href="http://www.railstutorial.org/">Ruby on Rails Tutorial</a>
    sample application.
  </h2>

  <%= link_to "Google", '#', class: "btn btn-lg btn-primary" %>
</div>

<%= link_to image_tag("rails.png", alt: "Rails logo"),
            'http://rubyonrails.org/' %>
```

En el código se utiliza la etiqueta **image_tag**, que busca dentro de la carpeta app/**assets/images/** la imagen de nombre **rails.png**, debido a que la imagen no existe, mostrará un error si se quiere visualizar esa vista, lo que se debe hacer es descargar la imagen y almacenarla en el directorio antes mencionado

3.1. Escribir el siguiente comando en el terminal.

```
$ curl -o app/assets/images/rails.png -OL railstutorial.org/rails.png
```

Con el comando anterior, descarga la imagen desde la url y la ubica en el directorio donde se almacenan todas las imágenes que se incluyen en el proyecto RoR, para probar que la página principal de la aplicación funciona correctamente, solo hay que iniciar el navegador en el terminal.

```
$ rails s
```

3.2. Navegar en la dirección localhost:3000, podrá observar que la vista home carga perfectamente y también se observa que el diseño no es muy amigable.

Figura 106 Index de la aplicación sin css

4. Bootstrap

4.1. Para integrar bootstrap en una aplicación RoR lo primero es agregar la gema en el archivo **Gemfile**.

```
gem 'bootstrap-sass', '3.3.7'
```

4.2. Instalar la gema, escribiendo el siguiente.

```
$ bundle install
```

5. El comando **rails generate** crea automáticamente archivos css por cada controlador que se genera en el proyecto, pero en este caso se creará un solo archivo de nombre **custom.scss** dentro del directorio **app/assets/stylesheets/**, en el cual se ubicará todo el código css del proyecto. La extensión **scss** se utiliza cuando se trabaja con Sass.

5.1. Para incluir bootstrap agregar al archivo custom.scss.

```
@import "bootstrap-sprockets";
@import "bootstrap";
```

5.2. Reiniciar el servidor, y recargar el navegador.

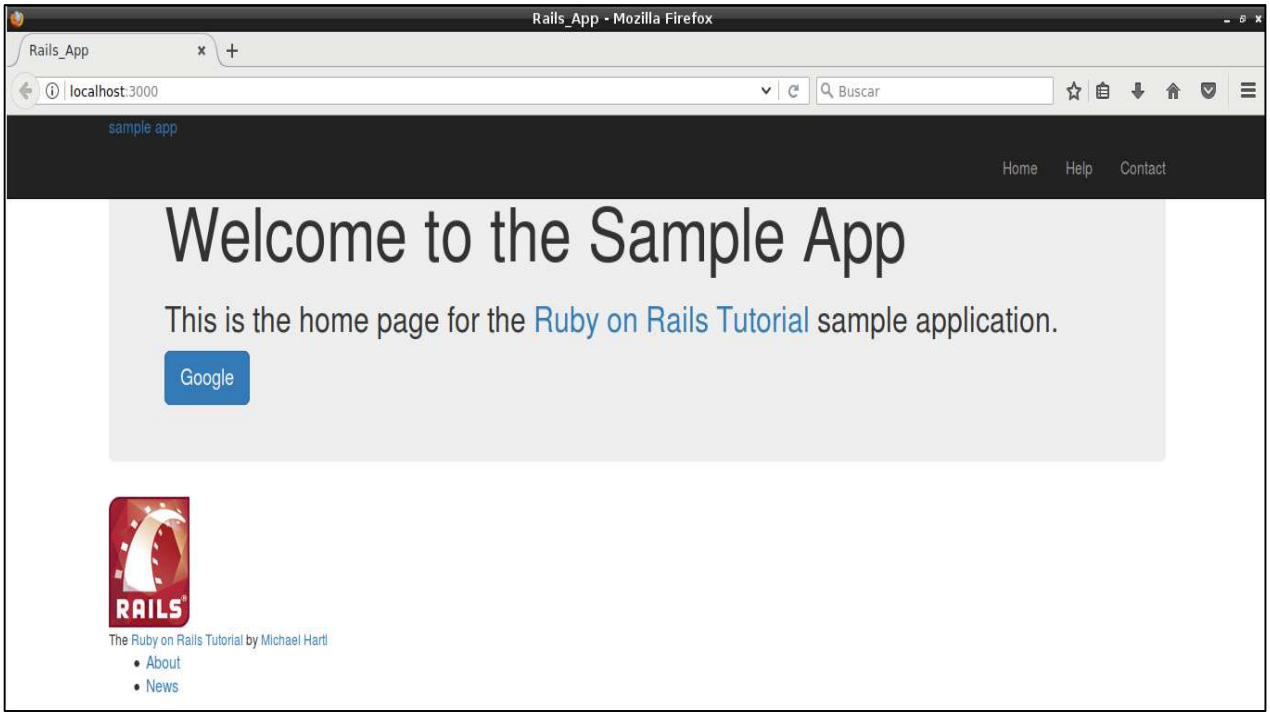

Figura 107 Index con bootstrap integrado al proyecto

Como se observa en la figura 107, ya se ha integrado al proyecto la gema bootstrap, el siguiente paso será empezar a agregar código css para darle un diseño más vistoso.

5.3. Agregar el siguiente código css al archivo custom.scss y actualizar el navegador.

```
body {
  padding-top: 60px;
}

section {
  overflow: auto;
}

textarea {
  resize: vertical;
}

.center {
  text-align: center;
}

.center h1 {
  margin-bottom: 10px;
}
```

Al momento de recargar el navegador, se podrá observar como la vista se va ajustando poco a poco.

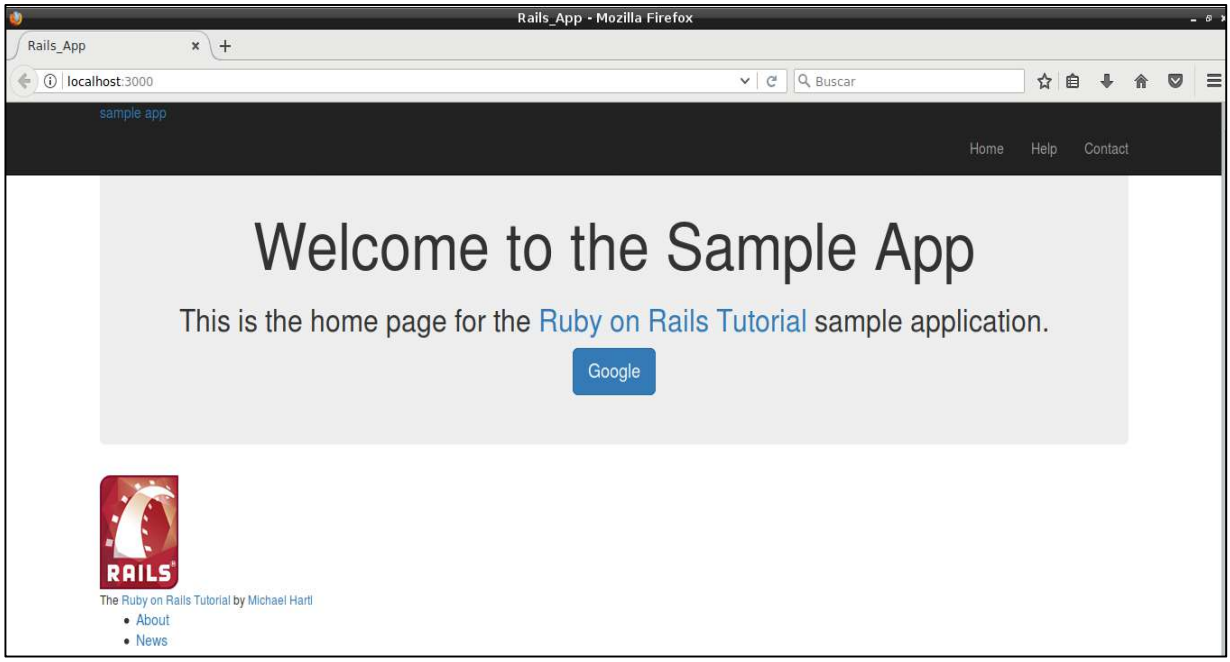

Figura 108 Aplicando css al body de la aplicación

5.4. Continuando en el archivo custom.scss, agregar el código css para las letras y actualizar el navegador para observar los cambios.

```
/* typography */
h1, h2, h3, h4, h5, h6 {
  line-height: 1;
}

h1 {
  font-size: 3em;
  letter-spacing: -2px;
  margin-bottom: 30px;
  text-align: center;
}

h2 {
  font-size: 1.2em;
  letter-spacing: -1px;
  margin-bottom: 30px;
  text-align: center;
  font-weight: normal;
  color: #777;
}

p {
  font-size: 1.1em;
  line-height: 1.7em;
}
```

El código anterior es principalmente empleado para darle formato a las letras dentro de los cuales se puede mencionar: tipo de letra, tamaño color, margen entre otros detalles.

Como se observa en la siguiente figura, los cambios aplicando css se han ido realizando con éxito.

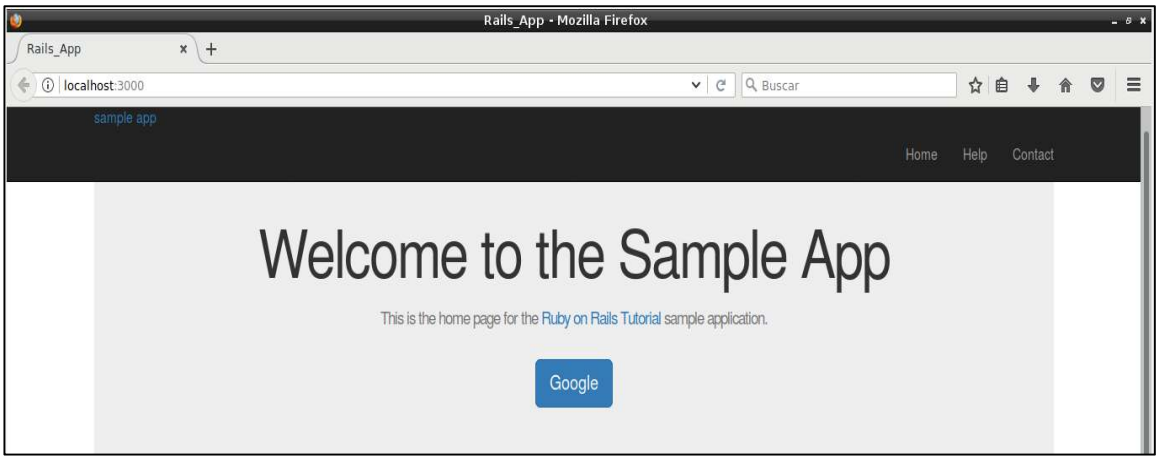

Figura 109 Aplicando css a las letras en la aplicación

5.5. Aplicar css al header de la aplicación, en el mismo archivo agregar:

```
/* header */
#logo {
  float: left;
  margin-right: 10px;
  font-size: 1.7em;
  color: #fff;
  text-transform: uppercase;
  letter-spacing: -1px;
  padding-top: 9px;
  font-weight: bold;
}

#logo:hover {
  color: #fff;
  text-decoration: none;
}
```

Observar la parte superior de la aplicación en el encabezado se ha modificado.

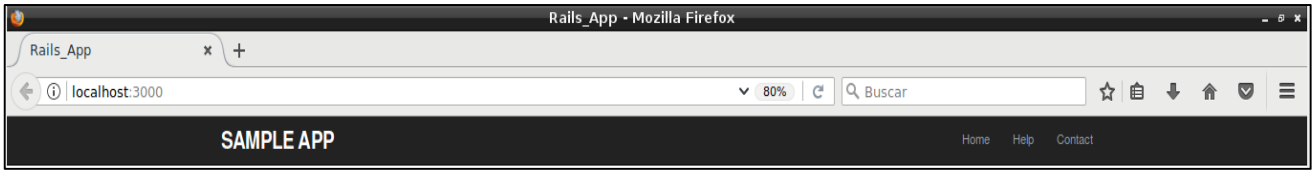

Figura 110 Aplicando css al header de la aplicación

6. Al finalizar la aplicación quedará como se muestra en la siguiente figura.

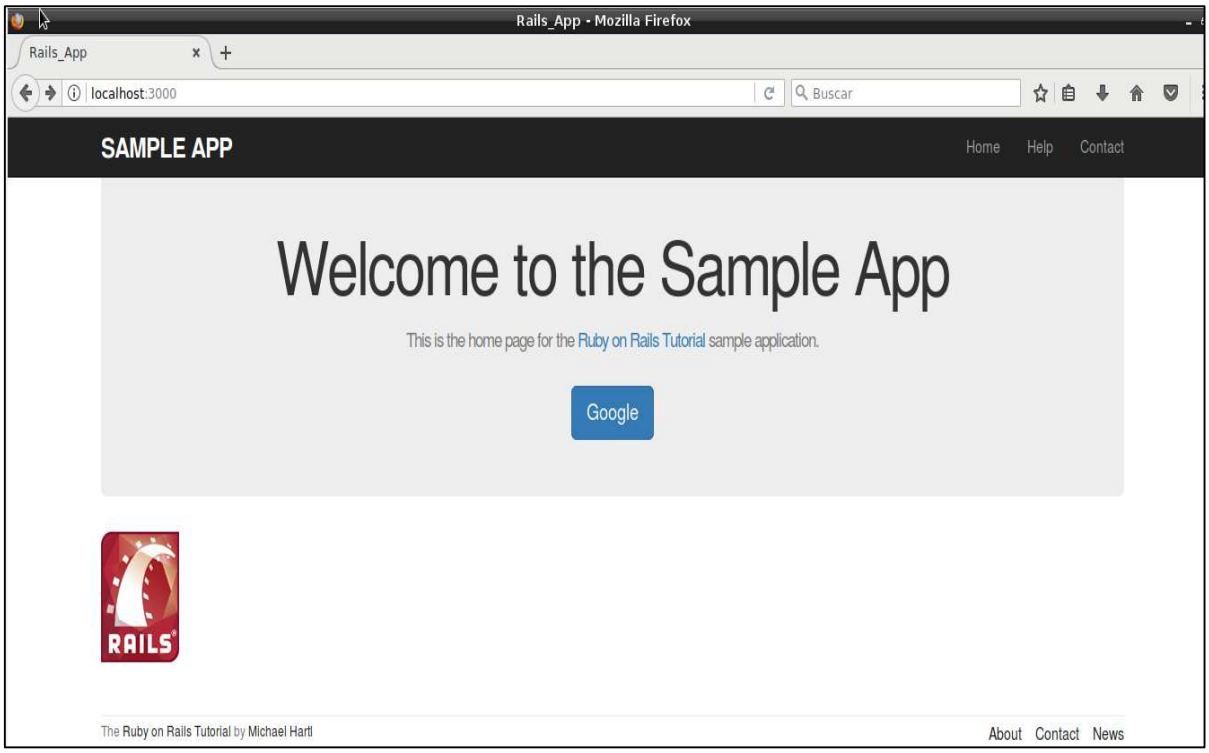

Figura 111 Aplicando css al footer de la aplicación

7. Links dentro de la aplicación.

En Rails se puede utilizar rutas con nombres en la configuración del archivo routes.rb; el método root lo que hace es tomar como raíz a un método o acción de la aplicación, el objetivo de crear rutas con nombre es que permite referirse a una ruta a través de un nombre y no de una url especifica.

7.1. Para crear las rutas con nombre modificar el archivo **routes.rb.**

```
root 'static_pages#home'
get  '/help',    to: 'static_pages#help'
get  '/about',   to: 'static_pages#about'
get  '/contact', to: 'static_pages#contact'
```

En el código anterior todas las peticiones GET http que van dirigida hacia una url, enrutan a una acción específica dentro del controlador de Static_pages. Esto permite crear las rutas con nombres para cada una de los métodos.

7.2. Rutas con nombres creadas.

Page	URL	Named route
Home	/	root_path
About	/about	about_path
Help	/help	help_path
Contact	/contact	contact_path

Figura 112 Rutas con nombre creadas en la aplicación

Como se observa en la figura 112, para la página home la ruta es **root_path,** esto es debido a que en el archivo routes.rb se configuró esa acción como la raíz de la aplicación.

7.3. Ejecutar el comando rails routes en el terminal y observar las rutas de cada una de las acciones de la aplicación.

```
$ rails routes
```

```
debian@debian:~/Proyectos_RoR/App_tets_TDD$ rails routes
 Prefix Verb URI Pattern            Controller#Action
   root GET  /                      static_pages#home
   help GET  /help(.:format)        static_pages#help
  about GET  /about(.:format)       static_pages#about
contact GET  /contact(.:format)     static_pages#contact
```

Figura 113 Rutas de la aplicación

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.

2. Con ayuda de la **figura 112** en la que se muestran las rutas, modificar los archivos **Partial** de las vistas que están dentro del directorio **layouts** y agregar las rutas con nombre en cada una de las etiquetas **link_to**, para poder navegar dentro de la aplicación en las distintas vistas que contiene.

Extra: modificar el contenido css del archivo custom.scss, cambiándolo de código css a código **sass**.

Proyecto 1: Desarrollo de una aplicación al estilo instagram.

Introducción

Con la implementación y realización de esta guía, se pretende que el estudiante realice de manera guiada una pequeña aplicación web similar a Instagram, haciendo uso de gemas muy útiles en el desarrollo de aplicación en Rails como son: paperclip; para publicación de imágenes, devise; para la gestión de usuarios, con la finalidad que éste consolide las competencias que ha venido adquiriendo con las asignaciones anteriores.

Objetivos

- Consolidar los conocimientos sobre Ruby on Rails adquiridos en las asignaciones previas.
- Desarrollar una aplicación en Rails desde 0.
- Conocer acerca de la gema devise.

Tiempo

- Tres sesiones de clases

Requerimientos

Software	Hardware
Sistema debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Thoughtbot. Paperclip. github.com/thoughtbot/paperclip
- Beginner Programmers. Make an instagram like site in under 30 minutes. www.youtube.com/watch?v=MpFO4Zr0EPE
- Plataformatec. Devise. github.com/plataformatec/devise

Desarrollo

En este proyecto, es en su mayoría guiado, todos los pasos descritos a continuación les llevará a desarrollar una aplicación web con la funcionalidad de crear usuarios y que los usuarios publiquen imágenes con su respectiva descripción.

1. Como primer paso e inicio del desarrollo de la aplicación se debe crear un proyecto en este caso con el nombre Instagram.

   1.1.Creación del nuevo proyecto.

```
$ rails new Instagram
```

   1.2.Una vez terminado el proceso de creación del proyecto, ingresar al directorio.

```
$ cd Instagram
```

   1.3.Para visualizar el resultado iniciar el servidor de Rails.

```
$ rails s
```

   1.4.En el navegador ir a la dirección localhost:3000, si todo sea realizado correctamente cargará el mensaje de bienvenida como el mostrado en la **figura 114.**

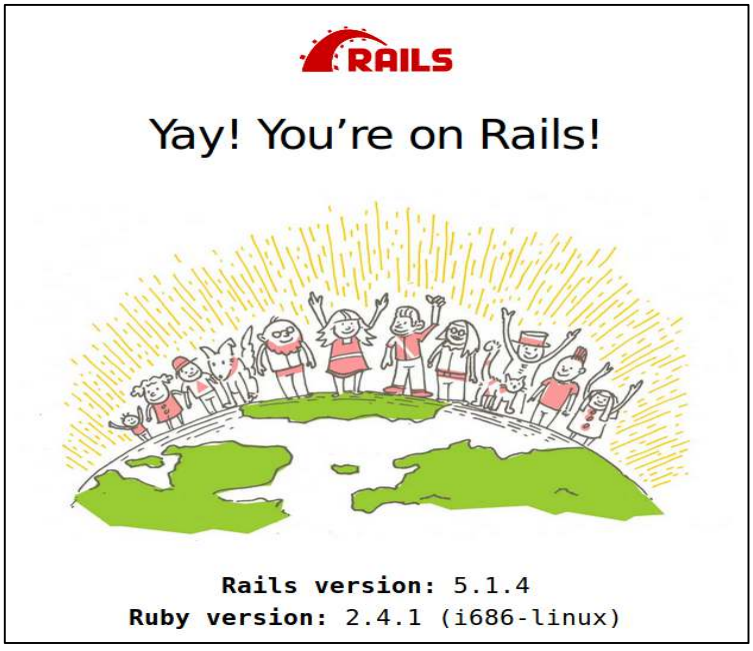

Figura 114 Página de inicio por defecto

2. Instalación de la gema devise.

Devise es una gema que sirve para que los usuario de un sitio web creen usuario, inicien sesión, cierren su respectiva sesión así como la posibilidad de recuperar contraseñas, modificar sus perfiles etc.

2.1.Para incluir devise, agregar la gema en el archivo **Gemfile**.

```
gem 'devise'
```

2.2.Desde el terminal, ubicado en la ruta del proyecto instalar la gema.

```
$ bundle install
```

3. Crear estructura de usuarios con devise. En los siguientes pasos se instalara la gema como parte del proyecto y luego se creará el modelo que la maquetación de cómo se crearán los usuarios.

3.1.Instalar devise en el Instagram.

```
$ rails generate devise:install
```

3.2.Generar el modelo para los usuarios.

```
$ rails generate devise User
```

3.3.Realizar migración para la base de datos. Migrar es una característica de Active Record que permite tener un esquema de base de dato a través del tiempo, es decir cada vez que se genera un cambio se realiza una migración lo cual genera un nuevo esquema con el nuevo cambio por lo que será fácil regresar a un esquema anterior si algo no funciona o no era lo que se deseaba.

```
$ rake db:migrate
```

3.4.Correr el servidor para ver lo que se ha generado.

```
$ rails s
```

3.5. En el navegador ir a la url **localhost:3000/users/sign_in**. En esta vista se muestra que se ha generado un formulario para iniciar sesión al igual que para las otras funcionalidades entre ellas la de crear usuarios.

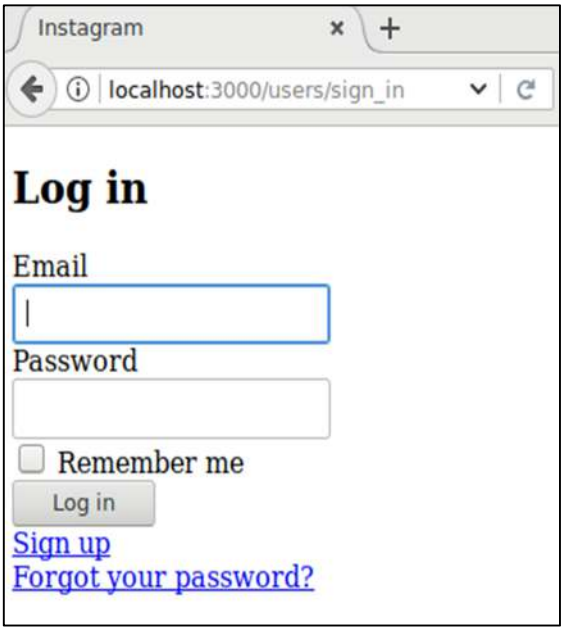

Figura 115 Vista de login de usuario

4. Poner estilo al Instagram a través la gema de bootstrap-sass.

4.1. Igual que la instalación de la gema devise, en el archivo **Gemfile** incluir la siguiente la gema **bootstrap-sass'** la cual permite poner estilo a la aplicación y usar todas bondades de **bootstrap 3**.

```
gem 'bootstrap-sass','~>3.3.5'
```

4.2. Instalar las gemas desde el terminal.

```
$ bundle install
```

4.3. En el archivo app/assets/javascripts/stylesheets/**application.css** agregar las siguientes líneas. Con lo cual se está configurando la aplicación para que exporte e interprete los estilos bootstrap y cambiar la extensión al archivo anterior a **application.scss**

```
@import "bootstrap-sprockets";
@import "bootstrap";
```

4.4. En el app/assets/javascripts/**application.js** agregar lo siguiente. Con lo cual se está configurando la aplicación, para el uso de los JavaScript de bootstrap.

```
//=require bootstrap-sprockets
```

5. Generando vistas, modelos y controladores**.**

5.1.Copiar el siguiente código en **app/view/application.html.erb**.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Instagram</title>
    <%= stylesheet_link_tag "application", media: "all","data-turbolinks-track" => true %>
    <%= javascript_include_tag "application", "data-turbolinks-track" => true %>
    <%= csrf_meta_tags %>
</head>
<body>
    <!-- Navigation -->
<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
    <div class="container">
            <!-- Brand and toggle get grouped for better mobile display -->
                <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
            target="#bs-example-navbar-collapse-1">
                        <span class="sr-only">Toggle navigation</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                </button>
                </div>
<!-- Collect the nav links, forms, and other content for toggling -->
            <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
                <ul class="nav navbar-nav">
                    <li><a href="#">About</a></li>
                    <li><a href="#">Services</a></li>
                   <li><a href="#">Contact</a></li>
                </ul>
            </div>
<!-- /.navbar-collapse -->
    </div>
<!-- /.container -->
</nav>
<!-- Page Content -->
<div class="container">
    <div class="row">
            <div class="col-lg-12 text-center">
                <%= yield %>
            </div>
    </div>
<!-- /.row -->
</div>
<!-- /.container -->
    <style>body{padding-top: 70px;}</style>
</body>
</html>
```

5.2.Generar el modelo Post. El cual tendrá como único campo **description** que será de tipo **text.**

```
$ rails generate model Post description:text
```

5.3.Una vez teniendo el modelo generar el controlador de nombre Posts con tres vistas las cuales serán new para crear nuevos post, index para el inicio y show para ver un post en particular.

```
$ rails generate controller Posts new index show
```

5.4.Cambiar las rutas de la aplicación en el archivo **app/config/routes.rb,** de tal manera que el index que se acaba de generar sea la primera vista al cargar la aplicación.

```
Rails.application.routes.draw do
root 'posts#index'
 resources :posts
devise_for   :users
end
```

5.5.Iniciar el servidor.

```
$ rails s
```

5.6.En el navegador dirigirse a la dirección localhost:3000. Cargará la vista index con la apariencia que se ejemplifica en la figura 3 ese diseño que parece se está cargando desde la vista **app/view/application.html.erb**. que se agregó en el paso 5.1 y carga el contenido de index debido a **<%= yield %>** que redendiriza las otras vistas bajo ésta

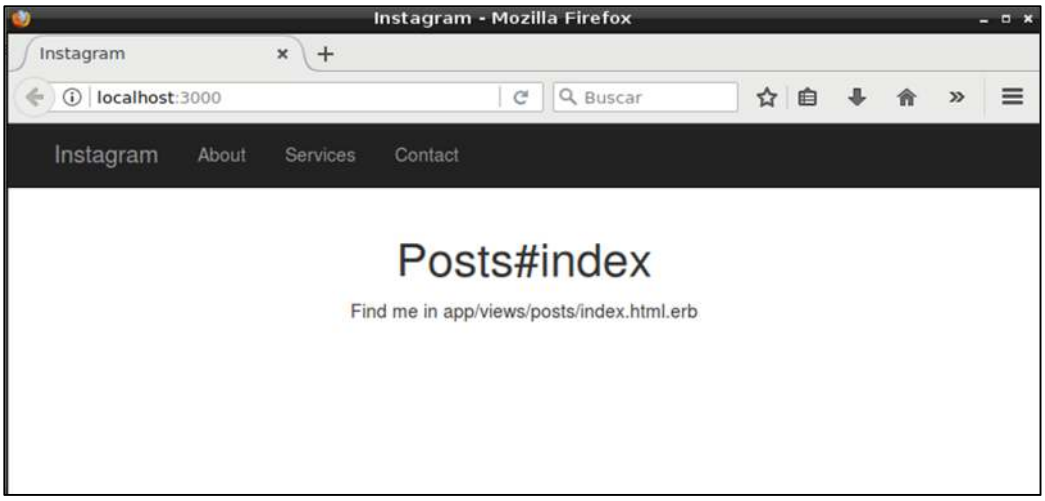

Figura 116 Vista index como página de inicio

6. Configuración de link_to en la vista.

6.1.Consultar las rutas. Para ello usar el comando routes que permite visualizar todas las urls y el acceso a las diferentes acciones de los controladores

```
$ rake routes
```

6.2.Se debe obtener una lista similar a la siguiente figura.

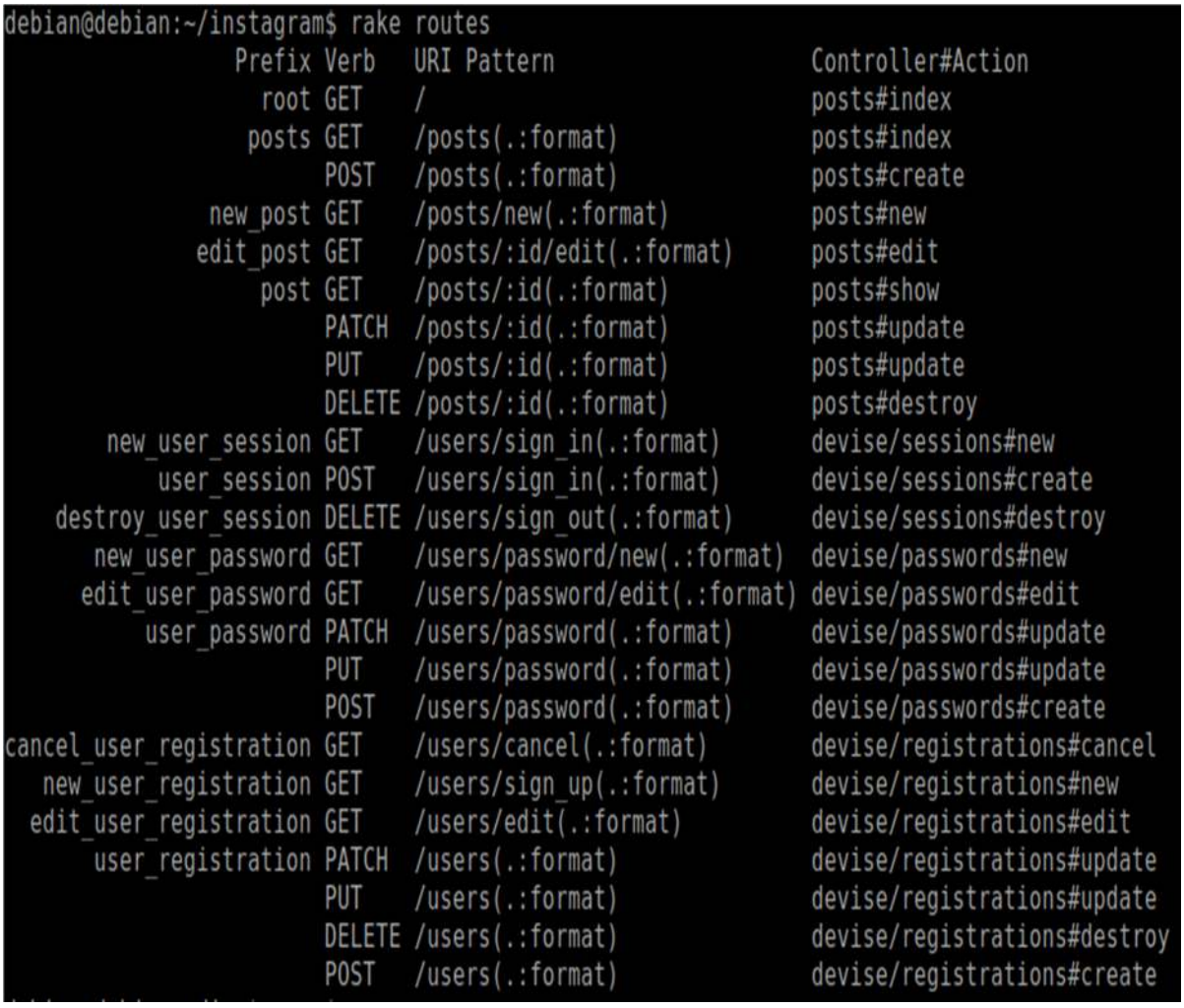

Figura 117 Lista de las rutas de la aplicación

7. Modificará el archivo **app/view/application.html.erb** y modificar el código de **<ul class="nav navbar-nav">** por el siguiente. Lo que se logra es configurar los link_to de acuerdo a las acción que va a realizar cada link para ello se está usando las url obtenidas en el paso anterior.

```
<ul class="nav navbar-nav">
    <li><%= link_to “New Post”, new_post_path %></li>
    <%- if current_user %>
    <li>
    <%= link_to "Logout", destroy_user_session_path, method: :delete,confirm:"Are you sure?" %>
    </li>
    <% else %>
            <li><%= link_to “Login”, new_user_session_path %></li>
            <li><%= link_to “Register”, new_user_registration_path %></li>
    <% end%>
</ul>
```

7.1. Verificación de las vistas, ver cada una de las vistas para comprobar que se estén cargando correctamente las acciones configuradas. En la vista **index** es donde se mostrarán todos los post publicados. Para acceder solo ir en el navegador a **localhost:3000**.

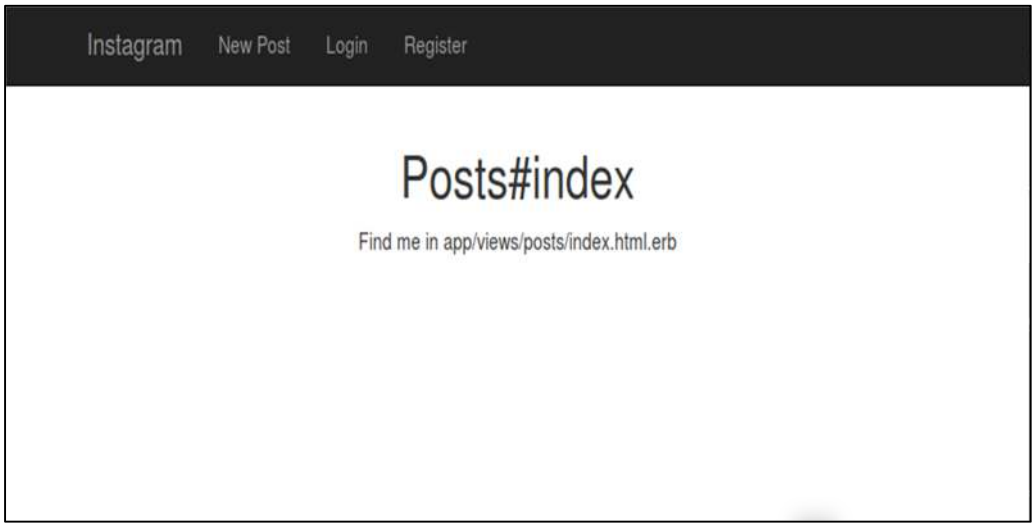

Figura 118 Vista index

7.2.Ir a la dirección localhost:300/posts/new, en la vista **New,** es donde se crearán los post para ser publicados.

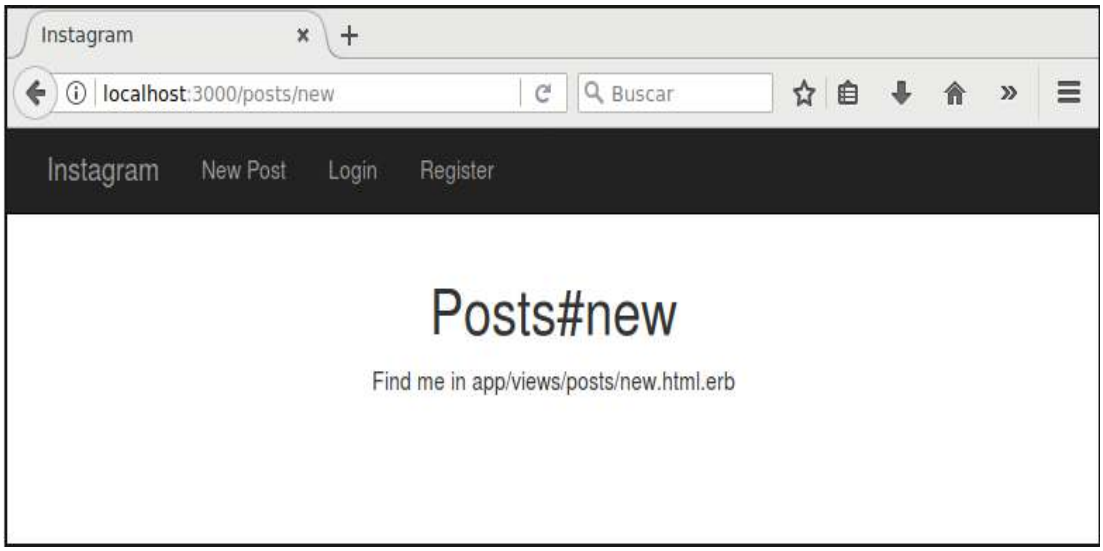

Figura 119 Vista new

7.3.Ir a la dirección localhost:3000/users/sign_in, esta vista **Login** es la que permitirá a los usuarios registrados iniciar sesión en la aplicación.

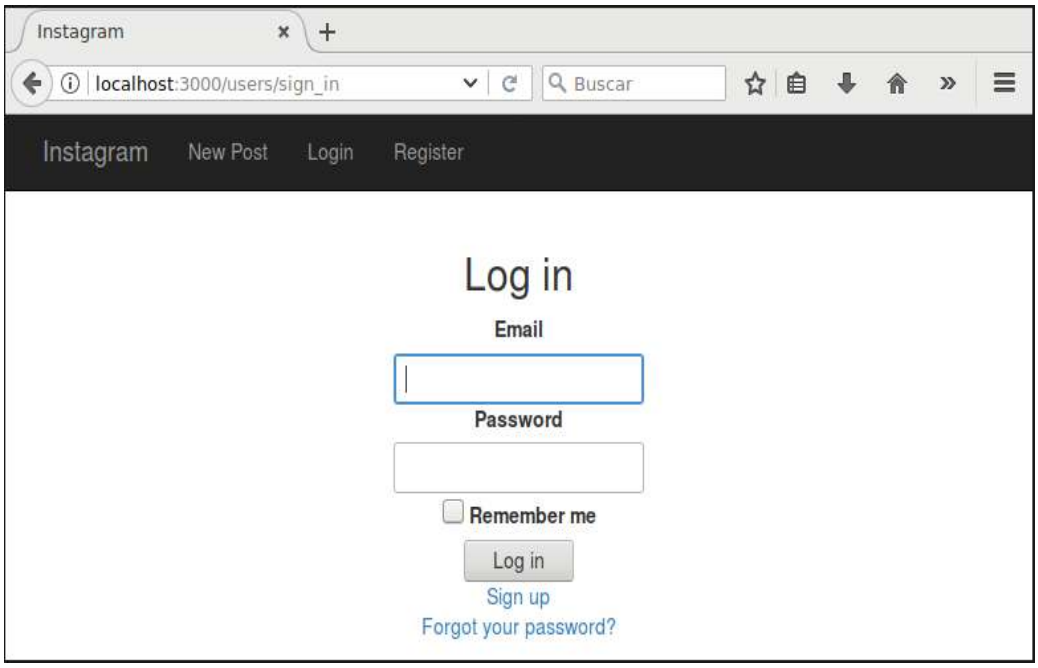

Figura 120 Vista Login

7.4.Ir a la dirección localhost:3000/users/sign_up, en la vista **Register** es donde se dan de alta a los nuevos usuarios.

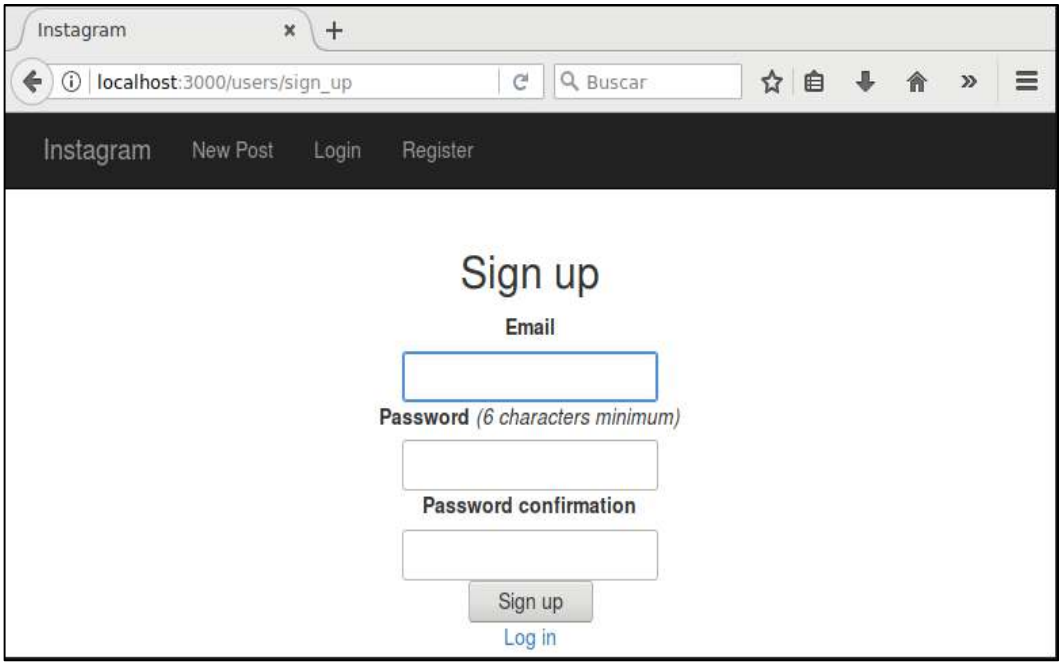

Figura 121 Vista para registrar un nuevo usuario

8. Configuración de la vista New Post.

8.1. Editar el archivo **app/views/posts/new.html.erb**. En el siguiente código se crea el formulario a este punto solo con el campo descripción.

```
<h2>New Post</h2>
 <hr>
    <%= form_for @post do |f| %>
    <%= f.label :description %>
      <%= f.text_area :description %>
      <br>
      <%=  f.submit %>
    <%end%>
```

8.2.Configurar el controlador de los Post, en el archivo **posts_controller.rb,** del directorio **app/controllers.rb/**, agregar el método new al controlador con el siguiente código.

```
def new
   @post = Post.new
end
```

8.3.Ir a la dirección localhost:3000/posts/new y ver que ya se ha creado un formulario para agregar un nuevo post como se muestra en la siguiente figura.

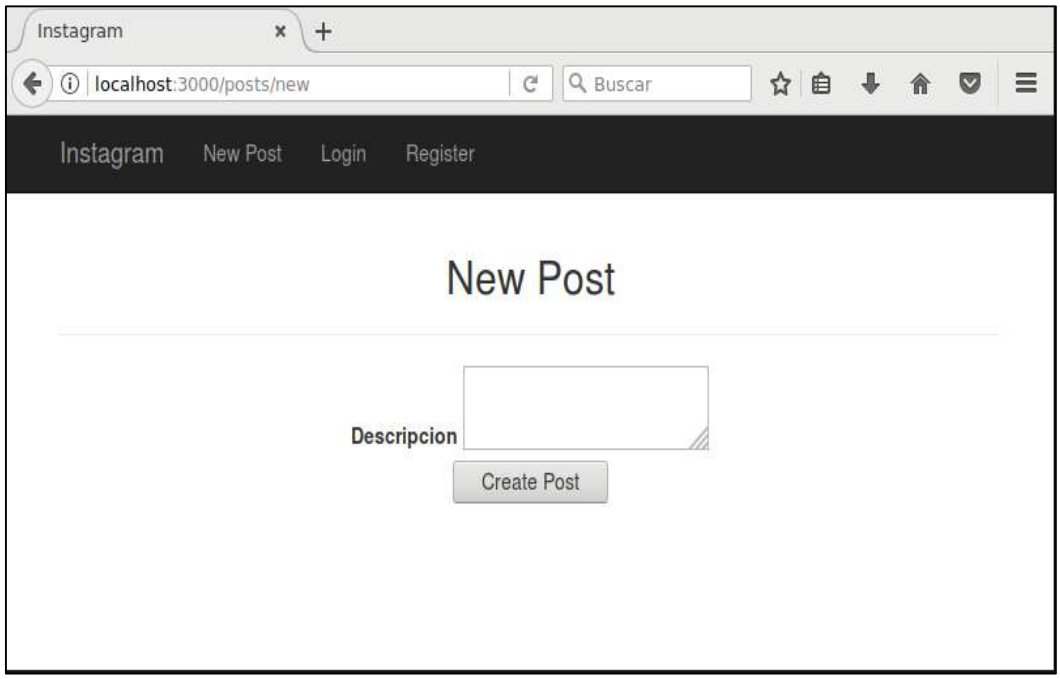

Figura 122 Formulario para crear un nuevo post

9. Mostrar notificación de actividades cuando un usuario se registra.

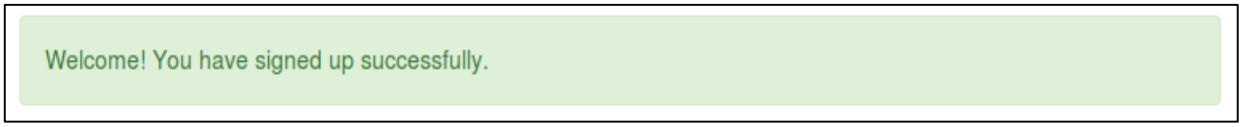

Figura 123 Notificación de un usuario nuevo registrado

10. En el archivo **app/view/application.html.erb,** en el div que tiene la clase "**row**" agregar el siguiente código. Que lo que hace es mostrar alertas tipo notificaciones como el de la **Figura 123**.

```
<div class="row">
    <% flash.each do |a,b| %>
    <dir class="alert alert-success"><%= b%> </dir>
   <%end%>
   <div class="col-lg-12 text-center">
   <%= yield %>
   </div>
</div>
```

11. Instalación de las gemas paperclip y aws-sdk.

11.1.Agregar en la gema al proyecto en el archivo **Gemfile.**

```
gem "paperclip","~>5.0.0"
gem 'aws-sdk', '< 2.0'
```

11.2.Instalar las gemas.

```
$ bundle install
```

12. Crear la estructura de paperclip que permitirá manejar todo lo referente a la publicación de imágenes

```
$ rails g paperclip post image
```

13. Al momento de desarrollar esta guía, se ha encontrado una incompatibilidad de paperclip con Rails 5.1 al intentar hacer migración, para solucionarlo hay que especificar con que versión de Rails se está trabajando, para ello:

13.1.En el directorio **/db/migrate/**, buscar el último fichero de migración creado, agregar al final la versión, como se muestra a continuación la versión en color rojo.

```
class AddAttachmentImageToPosts < ActiveRecord::Migration[5.1]
    def self.up
      change_table :posts do |t|
      t.attachment :image
    end
   end
   def self.down
     remove_attachment :posts, :image
   end
end
```

13.2.Migrar los datos.

```
$ rake db:migrate
```

14. Configurar modelo, vista y controlador paperclip.
    14.1. Configurar el modelo app/models/**post.rb,** en este archivo copiar el código a continuación, en el cual se establecen las dimensiones de las imágenes, una ruta por omisión y valida que el contenido a subir sea de tipo imagen.

```
    has_attached_file :image, styles: {large: "600x600>", medium: "300x300>",
thumb: "150x150>" }, default_url: "/images/:style/missing.png"
    validates_attachment_content_type :image, content_type: /\Aimage\/.*\z/
```

15. Modificar el archivo **app/views/posts/new.html.erb.** Crear el formulario que se mostrará, para publicar los post.

```
<h2>New Post</h2>
  <hr>
   <%= form_for @post, html: { multipart: true } do |f| %>
      <%= f.label :image %>
      <%= f.file_field :image%>
     <br>
      <%= f.label :description %>
      <%= f.text_area :description %>
      <br>
      <%=  f.submit %>
<%end%>
```

15.1. Editar el controlador en app/controllers.rb/**posts_controller.rb** y agregar los métodos a utilizar.

```
def index
 @post = Post.all
end

def show
 @posts = Post.find(params[:id])
end

def create
  @post = Post.new(permit_post)
    if @post.save
      flash[:success] = "Success!"
       redirect_to post_path(@post)
     else
       flash[:error] = @post.errors.full_messages
      redirect_to new_post_path
    end
end

private
def permit_post
   params.require(:post).permit(:image,:description)
end
```

El método **index** será invocado por la página de inicio y lo que hace es listar todos los post que han sido creados, el método show muestra un post en particular para identificarlo recibe como parámetro el id de ese post. El método créate es el encargado generar el nuevo post

16. Configuración de la vista.

16.1. Agregar el siguiente código en app/views/posts/**show.html.erb**. El cual permite que se visualice un pos en particular.

```
<%= image_tag(@post.image.url(:large)) %>
<br>
<%= @post.description %>
<div class="link">
    <%= link_to "Destroy", @post , method: :delete,confirm:"Are you sure?"  %>
    <%= link_to "Edit",edit_post_path(@post) %>
</div>
```

17. Para visualizar todos los post, editar el archivo app/views/posts/**index.html.erb** y agregar el siguiente código ERB. Con lo cual se mostrarán todos los post generados.

```
<% @posts.each do |x| %>
<div>
        <%= image_tag x.image.url(:medium) %>
    <br>
        <%= x.description %>
  <div class="link">
  <%= link_to "Destroy", x , method: :delete,confirm:"Are you sure?"  %>
      <%= link_to "Edit", edit_post_path(x) %>
      <%= link_to "Show", x %>
  </div>
</div>
<% end %>
```

18. Crear el método **destroy** dentro del archivo **posts_controller.rb** del directorio **/app/controllers/,** agregar el siguiente código debajo del método **create**.

```
def destroy
  @post = Post.find(params[:id])
  if @post.destroy
     redirect_to posts_path
     flash[:success] = "The post is destroy!"
  else
    redirect_to posts_path
  end
end
```

19. Funcionamiento.

19.1.Crear un nuevo post.

Al momento de crear un nuevo Post, el formulario permite elegir una imagen almacenada y una descripción de ésta, para ser posteada.

Figura 124 Creación de un nuevo Post

19.2.Un post ya publicado, en la vista **show**.

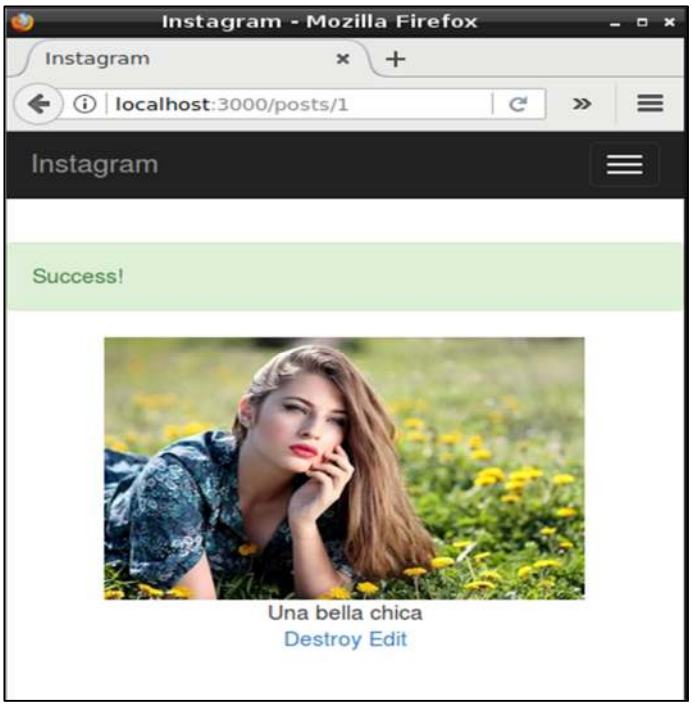

Figura 125 Post creado correctamente

20. Validaciones.

A este punto la aplicación es muy funcional, pero hay acciones que no se han tomado en cuenta, como es no permitir la creación de post sin contenido, que los usuarios estén logueados para publicar un post, poner un mínimo de caracteres en la descripción, entre otras, para ello Ruby on Rails permite realizar un sin número de validaciones para los modelos que permiten o deniegan dichas acciones.

20.1. Post solo visibles cuando el usuario haya iniciado sesión. Editar **/app/controllers/posts_controller.rb** (Poner justo después de la declaración de la clase).

```
before_action :authenticate_user!
```

20.2. Resultado de la primera validación se puede observar en la siguiente figura.

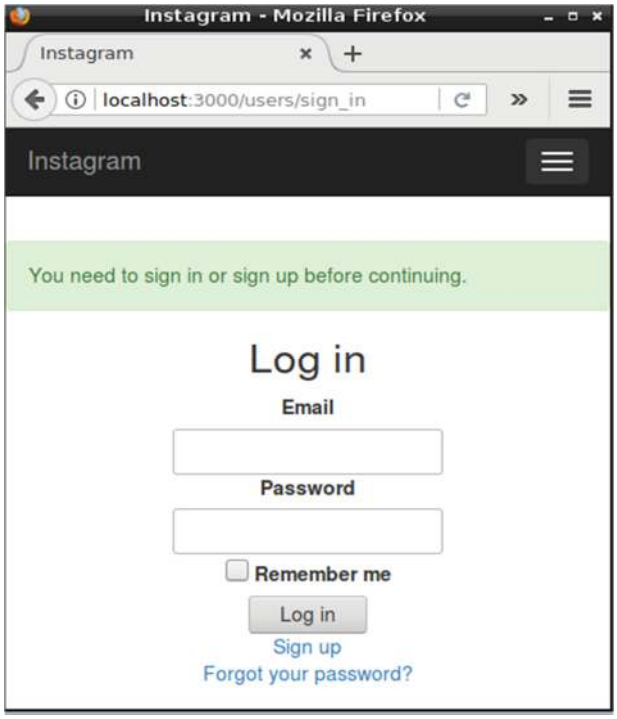

Figura 126 No se permite crear Post sino está logueado

20.3. No permitir publicar post vacíos, ni con descripción menor a diez caracteres. Para ello editar **/app/models/post.rb** y agregar el siguiente código.

```
validates :image, presence: true
validates :description, presence: true , length: { minimum: 10 }
```

En la siguiente figura se muestra el resultado de la segunda validación.

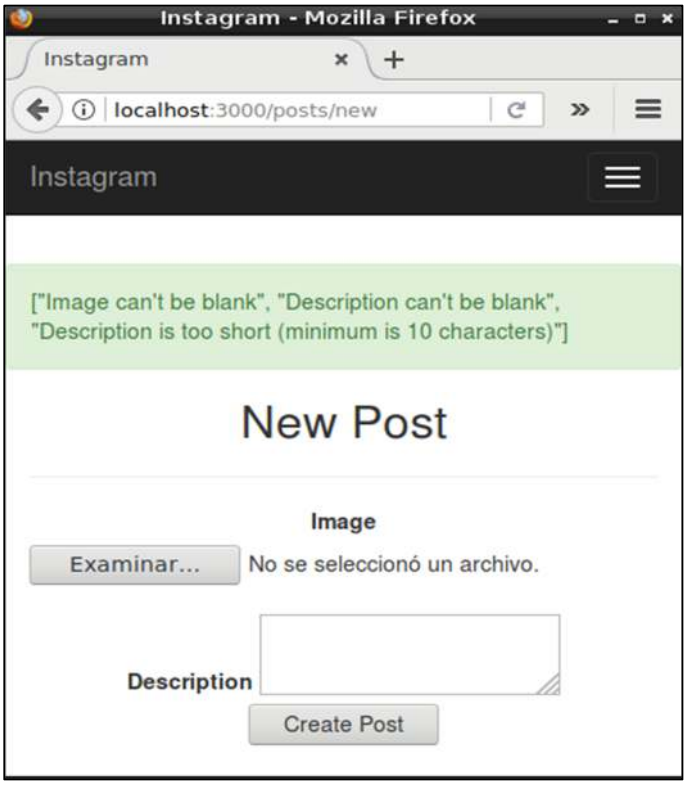

Figura 127 Validaciones al momento de crear un nuevo Post

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.
2. Aplicar **css** a las vistas según su criterio.
3. Realizar **validación**, que solo el usuario dueño del post pueda borrarlo.
4. Realizar la funcionalidad **Edit** para ello se debe crear la vista y los controladores necesarios como se realizó con las otras funcionalidades.

Proyecto 2: Blog

Introducción

En esta guía el estudiante podrá llevar a la práctica muchos de los conceptos antes estudiados, en base al desarrollo de aplicaciones web utilizando el framework Ruby on Rails, pondrá en uso los conocimientos adquiridos en el desarrollo de las asignaciones anteriores; creará un blog básico en el cual se ejecute acciones CRUD dentro del funcionamiento del proyecto, dichas funcionalidades se irán agregando en el transcurso de la guía.

Objetivos

- Poner en práctica conocimientos antes adquiridos.
- Crear un pequeño blog que ejecute acciones CRUD.
- Aprender a integrar al proyecto una plantilla bootstrap.

Tiempo

- Tres sesiones de clases

Requerimientos

Software	Hardware
Sistema debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Rails 5.4.1 • Nodejs • Sublimetext	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- Ruby on Rails org. Getting started with Rails. http://guides.rubyonrails.org/getting_started.html
- Ruby on Rails org. Ruby on Rails guides. http://guides.rubyonrails.org/

Desarrollo

1. Crear un proyecto nuevo.

1.1. Ubicar en el directorio, donde se va almacenar el proyecto.

```
$ cd /home/debían/Proyectos_RoR
```

1.2. Generar un nuevo proyecto.

```
$ rails new blog
```

1.3. Entrar a la carpeta del proyecto.

```
$ cd blog/
```

1.4. Iniciar el servidor para comprobar que arranca sin ningún problema.

```
$ rails server
```

1.5. Abrir el navegador en la dirección.

```
http://localhost:3000/
```

Figura 128 Página de inicio de la aplicación

2. Página de inicio.

2.1. Para crear la página de inicio del proyecto, generar un controlador.

```
$ rails generate controller welcome index
```

Rails generará automáticamente los archivos y directorios como se muestra en la figura 129.

```
debian@debian:~/Proyectos_RoR/blog$ rails generate controller welcome index
Running via Spring preloader in process 10219
      create  app/controllers/welcome_controller.rb
       route  get 'welcome/index'
      invoke  erb
      create    app/views/welcome
      create    app/views/welcome/index.html.erb
      invoke  test_unit
      create    test/controllers/welcome_controller_test.rb
      invoke  helper
      create    app/helpers/welcome_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/welcome.coffee
      invoke    scss
      create      app/assets/stylesheets/welcome.scss
```

Figura 129 Primer controlador de la aplicación

Dentro de los archivos más importantes que se han generador esta: el controller ubicado en (app/controllers/**welcome_controller.rb**), el index ubicado en (app/views/welcome/**index.html.erb**).

2.2. Abrir el editor de texto el archivo (app/views/welcome/**index.html.erb**), borrar lo que contiene, agregar la siguiente línea y guardar los cambios.

```
<h1>Hello, Rails!</h1>
```

2.3. Configurar la página de inicio, para que inicie con el index creado, deberá configurar el archivo routes (config/**routes.rb**) el cual contiene:

```
routes.rb
1  Rails.application.routes.draw do
2    get 'welcome/index'
3
4    # For details on the DSL available
5  end
6
```

Figura 130 Archivo routes.rb

2.3.1. Para que la aplicación inicie con el index creado, agregar la siguiente línea al archivo routes.rb.

```
root 'welcome#index'
```

2.3.2. Guarde los cambios realizados y recargar el navegador.

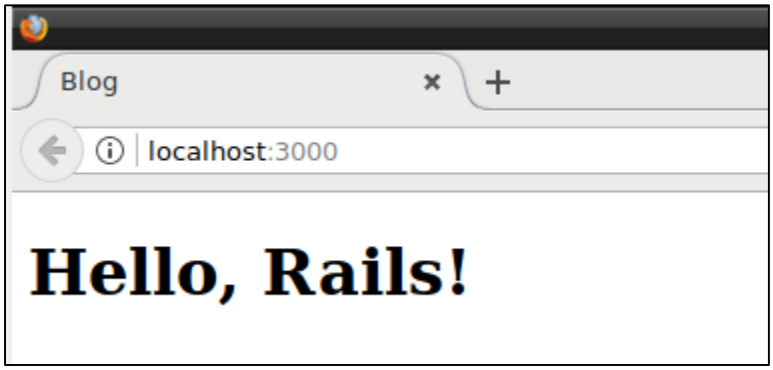

Figura 131 Página de inicio configurada

3. Operaciones CRUD

En esta sección creará un nuevo recurso (resource). Un recurso es el término usado para una colección de objetos similares como artículos, animales o personas; Se puede crear, leer, actualizar y eliminar a esto se le conoce como operaciones CRUD.

3.1. Método new.

3.1.1. Generar el controlador del article

```
$ rails generate controller articles
```

Creará automáticamente un conjunto archivos y mostrará la ruta de estos.

```
debian@debian:~/Proyectos_RoR/blog$ rails generate controller articles
Running via Spring preloader in process 10502
      create  app/controllers/articles_controller.rb
      invoke  erb
      create    app/views/articles
      invoke  test_unit
      create    test/controllers/articles_controller_test.rb
      invoke  helper
      create    app/helpers/articles_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/articles.coffee
      invoke    scss
      create      app/assets/stylesheets/articles.scss
```

Figura 132 Controlador articles generado

Al abrir el archivo (app/controllers/**articles_controller.rb**), se podrá observar que contiene el controlador vacío, dentro de este se crearán los métodos CRUD del proyecto:

Figura 133 Archivo article_controller.rb

3.1.2. Crear el método new manualmente, dentro del controlador, agregar el código en el archivo (app/controllers/**articles_controller.rb**).

```
def new
end
```

3.1.3. Crear el archivo **new.html.erb** en el directorio (**app/views/articles/**), copiar la dentro del archivo creado la siguiente línea y guarda.

```
<h1>New Article</h1>
```

3.1.4. Editar el archivo (config/routes.rb), agregar el método resources al artículo debajo del método root, lo que hace es configurar totas las rutas para el controlador articles.

```
resources :articles
```

3.1.5. Ir a la dirección http://localhost:3000/articles/new y verificar que cargar la página correctamente.

Figura 134 Página new

3.1.6. Crear un formulario en el archivo (app/views/articles/**new.html.erb**).

```
<%= form_for :article, url: articles_path do |f| %>
  <p>
  <%= f.label :title %><br>
  <%= f.text_field :title %>
  </p>
  <p>
  <%= f.label :text %><br>
  <%= f.text_area :text %>
  </p>
  <p>
  <%= f.submit %>
  </p>
<% end %>
```

3.1.7. Guardar los cambios y actualizar el navegador, podrá ver que se ha generado un formulario.

Figura 135 Formulario para crear un nuevo artículo

3.2. Método create.

3.2.1. Debajo del método new creado anteriormente en el archivo (app/controllers/**articles_controller.rb**), declarar un nuevo método llamado create para eso agregar el siguiente código.

```
def create
render plain: params[:article].inspect
end
```

3.2.2. Crear un modelo para almacenar los datos de los artículos.

```
$ rails generate model Article title:string text:text
```

```
Running via Spring preloader in process 10650
      invoke  active_record
      create    db/migrate/20180320221149_create_articles.rb
      create    app/models/article.rb
      invoke    test_unit
      create      test/models/article_test.rb
      create      test/fixtures/articles.yml
```

Figura 136 Archivos generador por el modelo

3.2.3. Migrar la base datos.

```
$ rails db:migrate
```

Al escribir el comando anterior en el terminal, se mostrará un mensaje como el de la siguiente figura.

```
debian@debian:~/Proyectos_RoR/blog$ rails db:migrate
== 20171129203046 CreateArticles: migrating ===================
-- create_table(:articles)
   -> 0.0123s
== 20171129203046 CreateArticles: migrated (0.0130s) ==========
```

Figura 137 Migración de la base de datos

3.3. Guardar datos.

Para almacenar los datos, editar el método create y agregar el método article_params en el archivo (app/controllers/**articles_controller.rb**).

```
def create
    @article = Article.new(article_params)
    @article.save
     redirect_to @article
end
private
 def article_params
   params.require(:article).permit(:title, :text)
 end
```

El archivo deberá quedar como se muestra en la siguiente figura, una parte importante de notar es que el método article_params es tipo privado, el cual deberá quedar siempre en la parte inferior del archivo debido a que es el único método de tipo privado que se utilizará.

Figura 138 Archivo article_controller; método article_params

3.4. Método show

3.4.1. Crear el método **show** (app/controllers/**articles_controller.rb**), al igual que los anteriores.

```
def show
    @article = Article.find(params[:id])
end
```

3.4.2. Crear el archivo **show.html.erb,** dentro del directorio **app/views/articles/**.

```
<p>
 <strong>Title:</strong>
   <%= @article.title %>
</p>
<p>
 <strong>Text:</strong>
 <%= @article.text %>
</p>
```

3.4.3. Guardar los cambios, iniciar el servidor y navegar en la dirección (http://localhost:3000/articles/new).

Figura 139 Crear un nuevo artículo

3.4.4. Crear un artículo y guardarlo para verificar que todo funciona correctamente.

Figura 140 Artículo creado correctamente

3.5. Método index

Para mostrar todos los artículos creados y almacenados hay que crear el método index.

3.5.1. Dentro del archivo (app/controllers/**articles_controller.rb**), crear el método llamado index, de la misma manear que se han creado los métodos anteriores.

```
def index
  @articles = Article.all
end
```

3.5.2. Crear el archivo **index.html.erb** en el directorio **app/views/articles/**, y agregar el contenido erb del index.

```
<h1>Listing articles</h1>
  <table>
<tr>
    <th>Title</th>
    <th>Text</th>
 </tr>
  <% @articles.each do |article| %>
    <tr>
       <td><%= article.title %></td>
       <td><%= article.text %></td>
    </tr>
 <% end %>
```

3.5.3. Ir a la dirección http://localhost:3000/articles, comprobar que se muestra los artículos creados.

Figura 141 Artículos creados

3.6. Agregar link a los archivos existentes para navegar dentro del proyecto, abrir el archivo (app/views/**welcome/index.html.erb**), agregar la siguiente la línea.

```
<%= link_to 'My Blog', controller: 'articles' %>
```

3.6.1. Abrir el archivo (app/views/**articles/index.html.erb**) y agregar lo siguiente debajo de la etiqueta de cierre **table**.

```
<%= link_to 'New article', new_article_path %>
<%= link_to 'Back', root_path %>
```

3.6.2. Abrir el archivo (app/views/articles/**new.html.erb**), agregar el código al finalizar el form.

```
<%= link_to 'Back', articles_path %>
```

3.6.3. Abrir el archivo (app/views/articles/**show.html.erb**) agregar al final.

```
<%= link_to 'Back', articles_path %>
```

3.7. Validar algunos campos.

3.7.1. Abrir el archivo (app/**models/article.rb**), se puede observar que la clase no contiene nada, dentro de esa clase validar que el titulo contenga al menos 5 caracteres.

```
validates :title, presence: true,
    length: { minimum: 5 }
```

3.7.2. Abrir el archivo (app/**controllers/articles_controller.rb**), editar el método new y create de la siguiente manera.

```
def new
    @article = Article.new
end
def create
    @article = Article.new(article_params)
    if @article.save
        redirect_to @article
    else
      render 'new'
    end
end
```

3.7.3. Editar el archivo (app/views/**articles/new.html.erb**), para mostrar un mensaje de alerta cuando se quiera guardar un artículo sin título.

```
<h1>New Article</h1>
<%= form_for :article, url: articles_path do |f| %>

  <% if @article.errors.any? %>
    <div id="error_explanation">
          <h2>
       <%= pluralize(@article.errors.count, "error") %> prohibited
                  this article from being saved:
        </h2>
        <ul>
          <% @article.errors.full_messages.each do |msg| %>
          <li><%= msg %></li>
          <% end %>
       </ul>
    </div>
   <% end %>
      <p>
        <%= f.label :title %><br>
    <%= f.text_field :title %>
     </p>
      <p>
     <%= f.label :text %><br>
    <%= f.text_area :text %>
       </p>
    <p>
       <%= f.submit %>
    </p>
<% end %>
```

3.7.4. Ir a la URL (http://localhost:3000/articles), verificar que existen los link, de igual manera si guarda un artículo sin título mostrará un mensaje de error.

Figura 142 Link agregados en la vista

3.8. Método Update

3.8.1. Crear el método edit dentro del archivo **articles_controller.rb**.

```
def edit
  @article = Article.find(params[:id])
end
```

3.8.2. Crear un archivo llamado **edit.html.erb** dentro del directorio (app/views/**articles**) y agregar el siguiente código.

```
<h1>Editing article</h1>

<%= form_for :article, url: article_path(@article), method: :patch do |f| %>
<% if @article.errors.any? %>
    <div id="error_explanation">
     <h2>
       <%= pluralize(@article.errors.count, "error") %> prohibited
            this article from being saved:
     </h2>
     <ul>
       <% @article.errors.full_messages.each do |msg| %>
         <li><%= msg %></li>
        <% end %>
     </ul>
    </div>
<% end %>
<p>
  <%= f.label :title %><br>
  <%= f.text_field :title %>
</p>
<p>
   <%= f.label :text %><br>
    <%= f.text_area :text %>
</p>
<p>
    <%= f.submit %>
</p>
<% end %>
<%= link_to 'Back', articles_path %>
```

3.8.3. Agregar el método Update dentro archivo articles_controller.rb del directorio (app/controllers/).

```
def update
    @article = Article.find(params[:id])
    if @article.update(article_params)
      redirect_to @article
        else
       render 'edit'
    end
end
```

3.8.4. Editar el archivo index (app/views/**articles/index.html.erb**), cambiar el contenido de la tabla.

```
<table>
   <tr>
    <th>Title</th>
    <th>Text</th>
    <th colspan="2"></th>
    </tr>
    <% @articles.each do |article| %>
              <tr>
             <td><%= article.title %></td>
             <td><%= article.text %></td>
    <td><%= link_to 'Show', article_path(article) %></td>
    <td><%= link_to 'Edit', edit_article_path(article) %></td>
              </tr>
    <% end %>
</table>
```

3.8.5. Editar el archivo **show.html.erb** (app/views/articles/).

```
<p>
 <strong>Title:</strong>
   <%= @article.title %>
</p>
<p>
 <strong>Text:</strong>
 <%= @article.text %>
</p>
<%= link_to 'Edit', edit_article_path(@article) %> |
<%= link_to 'Back', articles_path %>
```

3.8.6. Guardar los cambios realizados anteriormente y actualizar el navegador (http://localhost:3000/articles), verificar que funciona correctamente.

Figura 143 Link show y edit en las vista

3.9. Crear plantillas parciales (**PARTIAL**).

3.9.1. Para eliminar la información duplicada de los formularios en los archivos show y edit, se creará una plantilla parcial que será compartida por ambos archivos; Crear un archivo llamado **_form.html.erb** en el directorio (app/views/articles/) y agregar el siguiente código.

```
<%= form_for @article do |f| %>
  <% if @article.errors.any? %>
    <div id="error_explanation">
      <h2>
       <%= pluralize(@article.errors.count, "error") %> prohibited
          this article from being saved:
          </h2>
              <ul>
                    <% @article.errors.full_messages.each do |msg| %>
            <li><%= msg %></li><% end %>
          </ul>
    </div>
   <% end %>
<p>
  <%= f.label :title %><br>
  <%= f.text_field :title %>
</p>
<p>
  <%= f.label :text %><br>
  <%= f.text_area :text %>
</p>
<p>
  <%= f.submit %>
</p>
<% end %>
```

3.9.2. Editar el archivo **new.html.erb** (app/views/articles/new.html.erb), como se muestra en siguiente cuadro. Como se observa se hace un llamado al form desde este archivo.

```
<h1>New article</h1>
<%= render 'form' %>
<%= link_to 'Back', articles_path %>
```

3.9.3. Editar el archivo **edit.html.erb** (app/views/articles/edit.html.erb).

```
<h1>Edit article</h1>
<%= render 'form' %>
<%= link_to 'Back', articles_path %>
```

3.10. Método borrar

Crear el método Destroy dentro del archivo **articles_controller.rb** y agregar el siguiente código (app/controllers/articles_controller.rb).

```
def destroy
  @article = Article.find(params[:id])
  @article.destroy
  redirect_to articles_path
end
```

3.10.1.Editar el archivo app/views/**articles/index.html.erb** para agregar el link de Destroy debajo del link de **edit**.

```
<td>
<%= link_to 'Destroy', article_path(article), method: :delete,data: { confirm: 'Are you sure?' } %>
</td>
```

El archivo **index.html.erb** (app/views/articles/index.html.erb), quedará como se muestra a continuación:

index.html.erb

```
<h1>Listing articles</h1>
  <table>
    <tr>
    <th>Title</th>
    <th>Text</th>
    <th colspan="2"></th>
      </tr>
<% @articles.each do |article| %>
      <tr>
    <td><%= article.title %></td>
    <td><%= article.text %></td>
    <td><%= link_to 'Show', article_path(article) %></td>
    <td><%= link_to 'Edit', edit_article_path(article) %></td>
    <td>
    <%= link_to 'Destroy', article_path(article), method: :delete,data: { confirm: 'Are you sure?' } %>
    </td>
    </tr>
    <% end %>
</table>

<%= link_to 'New article', new_article_path %>
<%= link_to 'Back', root_path %>
```

Figura 144 Index del directorio articles

3.10.2.Guardar todos los cambios realizados y actualizar el navegador en la dirección http://localhost:3000/articles, podrá observar que ya cuenta con la opción Destroy para poder eliminar un artículo creado.

Figura 145 Link Destroy agregado en la vista

3.10.3.Si presiona click en el link **Destroy** mostrará un mensaje de alerta para eliminar el artículo guardado, como se muestra a continuación.

Figura 146 Mensaje alerta de Destroy

Al final el archivo articles_controller.rb quedará como se muestra en la siguiente figura, como observa solo es método article_params es de tipo privado.

```ruby
class ArticlesController < ApplicationController

    def new
     @article = Article.new
     end

     def create
        @article = Article.new(article_params)
       if @article.save
                    redirect_to @article
       else
         render 'new'
      end
     end

     def show
          @article = Article.find(params[:id])
     end

     def index
         @articles = Article.all
     end

     def edit
      @article = Article.find(params[:id])
     end

     def update
       @article = Article.find(params[:id])
     if @article.update(article_params)
         redirect_to @article
             else
          render 'edit'
      end
     end

     def destroy
         @article = Article.find(params[:id])
         @article.destroy
         redirect_to articles_path
    end

   private
   def article_params
    params.require(:article).permit(:title, :text)
   end
end
```

Figura 147 Archivo articles_controller completo

4. Segundo modelo.

4.1. Se procederá a crear un segundo modelo, para generar una tabla y poder almacenar los comentarios que se le agregaran a cada artículo.

4.1.1.Generar el nuevo modelo.

```
$ rails generate model Comment commenter:string body:text article:references
```

Automáticamente genera los siguientes archivos y directorios.

```
debian@debian:~/Proyectos_RoR/blog$ rails generate model Comment commenter:stri
ng body:text article:references
Running via Spring preloader in process 11329
      invoke  active_record
      create    db/migrate/20180321030731_create_comments.rb
      create    app/models/comment.rb
      invoke    test_unit
      create      test/models/comment_test.rb
      create      test/fixtures/comments.yml
```

Figura 148 Archivos generados del segundo modelo

4.1.2.Migrar la base de datos.

```
$ rails db:migrate
```

Al ejecutar el comando anterior, se obtiene la siguiente salida en el terminal.

```
debian@debian:~/Proyectos_RoR/blog$ rails db:migrate
== 20180321030731 CreateComments: migrating ===========================
-- create_table(:comments)
   -> 0.0049s
== 20180321030731 CreateComments: migrated (0.0052s) ==================
```

Figura 149 Migración de segundo modelo

4.1.3.Establecer una relación entre los modelos que se han creado en el proyecto; Editar el archivo (app/**models/article.rb**) y agregar el código.

```
has_many :comments
```

El archivo **article.rb** quedará como se muestra en la siguiente figura:

article.rb

```
1  class Article < ApplicationRecord
2
3  has_many :comments
4  validates :title, presence: true,
5      length: { minimum: 5 }
6  end
```

Figura 150 Archivo article.rb

4.1.4.Editar el archivo routes.rb (config/**routes.rb**).

NOTA: Eliminar la línea **resources :articles** y agregar el siguiente código al archivo.

```
resources :articles do
    resources :comments
end
```

Las rutas se escriben de esta manera, debido a que cada comentario depende de un artículo almacenado.

4.1.5.Generar el controlador para los comentarios.

```
$ rails generate controller Comments
```

```
debian@debian:~/Proyectos_RoR/blog$ rails generate controller Comments
Running via Spring preloader in process 11878
      create  app/controllers/comments_controller.rb
      invoke  erb
      create    app/views/comments
      invoke  test_unit
      create    test/controllers/comments_controller_test.rb
      invoke  helper
      create    app/helpers/comments_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/comments.coffee
      invoke    scss
      create      app/assets/stylesheets/comments.scss
```

Figura 151 Creación del controlador para los comentarios

4.1.6.Para generar las vistas de los comentarios, crear una plantilla parcial para los comentarios de nombre **_comment.html.erb,** dentro del directorio (app/views/**comments/**) y agregar el siguiente código.

```
<p>
 <strong>Commenter:</strong>
 <%= comment.commenter %>
</p>
<p>
  <strong>Comment:</strong>
  <%= comment.body %>
</p>
<p>
<%= link_to 'Destroy Comment', [comment.article, comment],method: :delete,data: { confirm: 'Are you sure?' } %>
</p>
```

4.1.7.Crear un nuevo archivo parcial para el formulario, con el nombre **_form.html.erb** en el directorio (app/**views/comments/**).

```
<%= form_for([@article, @article.comments.build]) do |f| %>
  <p>
    <%= f.label :commenter %><br>
    <%= f.text_field :commenter %>
  </p>
  <p>
    <%= f.label :body %><br>
    <%= f.text_area :body %>
  </p>
  <p>
    <%= f.submit %>
  </p>
<% end %>
```

4.1.8.Editar el archivo **show.html.erb** del directorio (app/**views/articles/**) y modificara el código por el del siguiente.

```
<p>
  <strong>Title:</strong>
  <%= @article.title %>
</p>
<p>
   <strong>Text:</strong>
    <%= @article.text %>
</p>
  <h2>Comments</h2>
   <%= render @article.comments %>
   <h2>Add a comment:</h2>
<%= render 'comments/form' %>
<%= link_to 'Edit', edit_article_path(@article) %> |
<%= link_to 'Back', articles_path %>
```

4.1.9. Editar el archivo **comments_controller.rb** del directorio /app/**controllers/** y crear los métodos **create** y **destroy**, para poder generar y eliminar comentarios en los artículos, de igual forma se agrega el método comment_params, que es de tipo privado, todo el código se debe de agregar dentro de la clase que existe en el archivo.

```
def create
  @article = Article.find(params[:article_id])
  @comment = @article.comments.create(comment_params)
   redirect_to article_path(@article)
end

def destroy
   @article = Article.find(params[:article_id])
   @comment = @article.comments.find(params[:id])
  @comment.destroy
   redirect_to article_path(@article)
end

private
   def comment_params
     params.require(:comment).permit(:commenter, :body)
   end
```

4.1.10. Para eliminar los comentarios asociados a cada uno de los artículos almacenados en la aplicación, editar el archivo article.rb del directorio (**app/models/**) y agregar el siguiente código.

```
dependent: :destroy
```

El archivo quedará como se muestra en la siguiente figura.

Figura 152 Archivo article.rb con todas las validaciones

4.1.11. Verificar el funcionamiento del proyecto, guarde los cambios realizados y actualizar el navegador, si presiona click en el link show observará que se pueden crear comentarios a cada uno de los artículos

Figura 153 Muestra de los comentarios en la vista

5. Agregar una plantilla bootstrap al proyecto.

Para este segmento, deberá descargar una carpeta del siguiente link https://goo.gl/UcUKn9, contiene los archivos de la platilla bootstrap que se va a agregar al proyecto.

5.1. Copiar todos los archivos que están dentro (**Plantilla_bootstrap/css**), en la carpeta **stylesheets** del proyecto (blog/app/**assets/stylesheets**).

5.2. Copiar todos los archivos que están dentro (**Plantilla_bootstrap/js**) en la carpeta **javascripts** del proyecto (blog/app/**assets/ javascripts**).

5.3. Copiar la imagen que está dentro (**Plantilla_bootstrap/imagenes**) en la carpeta **images** del proyecto (blog/app/**assets/images**).

5.4. Copiar la carpeta **fonts** que está dentro (**Plantilla_bootstrap/**) en la carpeta **assets** del proyecto (blog/**app/assets/**).

5.5. Remplazar el contenido archivo **index.html.erb** de la aplicación blog/app/**views/welcome/index.html.erb,** por el contenido del archivo **index.html.erb** de la plantilla que se está agregando al proyecto (**Plantilla_bootstrap/index.html.erb**), guardar los cambios y actualizar el navegador.

5.6. Actualizar el navegador en la dirección http://localhost:3000/.

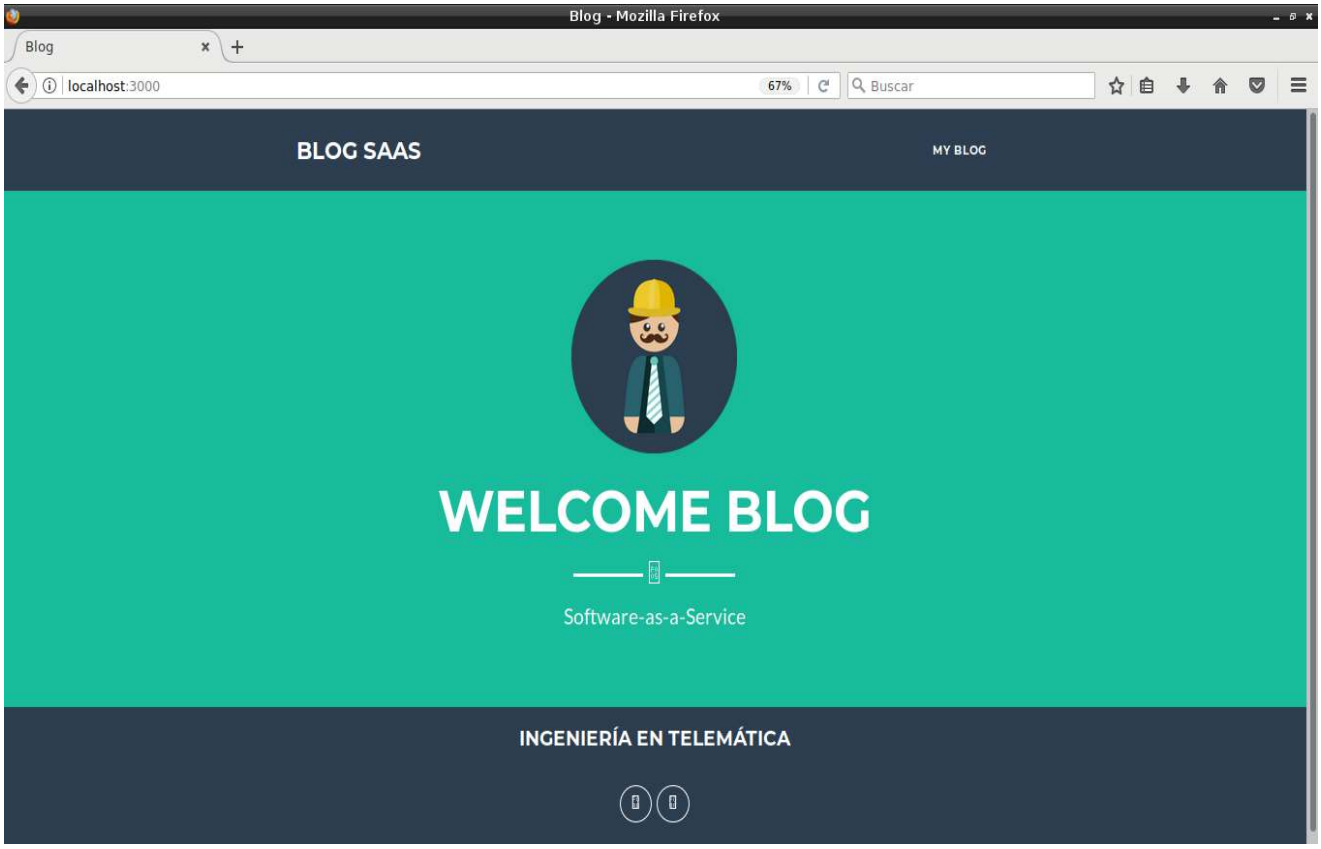

Figura 154 Index de la aplicación con plantilla bootstrap

NOTA: Como se puede observar, aún no se visualizan los fonts, para hacer que esto funcione realizar los siguientes cambios.

Figura 155 Error al cargar los fonts en la aplicación

5.7. Editar el archivo **application.rb** (config/application.rb) y agregar la siguiente línea dentro de la clase.

```
config.assets.paths << Rails.root.join("app", "assets", "fonts")
```

El archivo application.rb, deberá quedar como se muestra a continuación.

```
application.rb
1  require_relative 'boot'
2  require 'rails/all'
3
4  Bundler.require(*Rails.groups)
5
6  module Blog
7    class Application < Rails::Application
8      config.assets.paths << Rails.root.join("app", "assets", "fonts")
9      config.load_defaults 5.1
10   end
11 end
```

Figura 156 Archivo application.rb

5.8. Entrar al archivo **font-awesome.css.erb,** que se encuentra en el directorio /app/assets/**stylesheets/font-awesome.css.erb** y remplazar el código de **@font-face,** el cual aparece en la parte superior a partir de la línea 7 a como se muestra en la figura 30.

```
7  @font-face {
8    font-family: 'FontAwesome';
9    src: url('../fonts/fontawesome-webfont.eot?v=4.7.0');
10   src: url('../fonts/fontawesome-webfont.eot?#iefix&v=4.7.0') format('embedded-opentype'),
11   url('../fonts/fontawesome-webfont.woff2?v=4.7.0') format('woff2'),
12   url('../fonts/fontawesome-webfont.woff?v=4.7.0') format('woff'),
13   url('../fonts/fontawesome-webfont.ttf?v=4.7.0') format('truetype'),
14    url('../fonts/fontawesome-webfont.svg?v=4.7.0#fontawesomeregular') format('svg');
15   font-weight: normal;
16   font-style: normal;
17 }
```

Figura 157 Código de font que se debe remplazar

Remplazarlo por el siguiente código:

```
@font-face {
font-family: "FontAwesome";  src: url('<%= asset_path("fontawesome-webfont.eot") %>');
src: url('<%= asset_path("fontawesome-webfont.eot") + "?#iefix" %>') format('eot'),
url('<%= asset_path("fontawesome-webfont.woff") %>') format('woff'),
url('<%= asset_path("fontawesome-webfont.ttf") %>') format('truetype'),
url('<%= asset_path("fontawesome-webfont.svg") + "#FontAwesome" %>') format('svg');
font-weight: normal;
font-style: normal;
}
```

5.9. Reiniciar el servidor de Rails y actualizar el navegador en la dirección http://localhost:3000/, verificar que ya se puede visualizar correctamente los iconos.

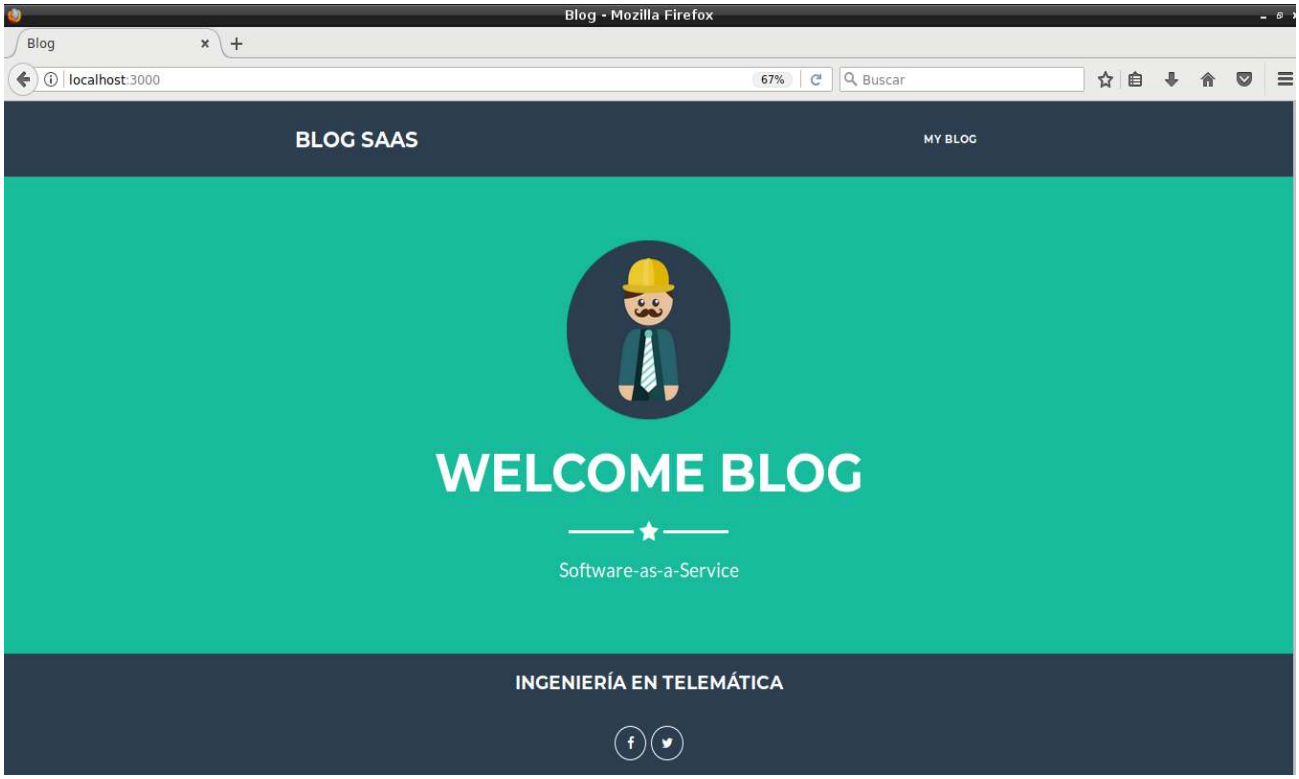

Figura 158 Index de la aplicación listo

Ejercicios propuestos para ser entregados al docente

1. Modifique los demás archivos de las vistas del proyecto, poyándose del archivo **index.html.erb** del directorio (app/**view/welcome/index.html.erb),** para obtener un diseño igual en toda la aplicación.
2. Verificar el correcto funcionamiento de todas las vistas, se pretende obtener vistas similares a las de la siguientes figuras.

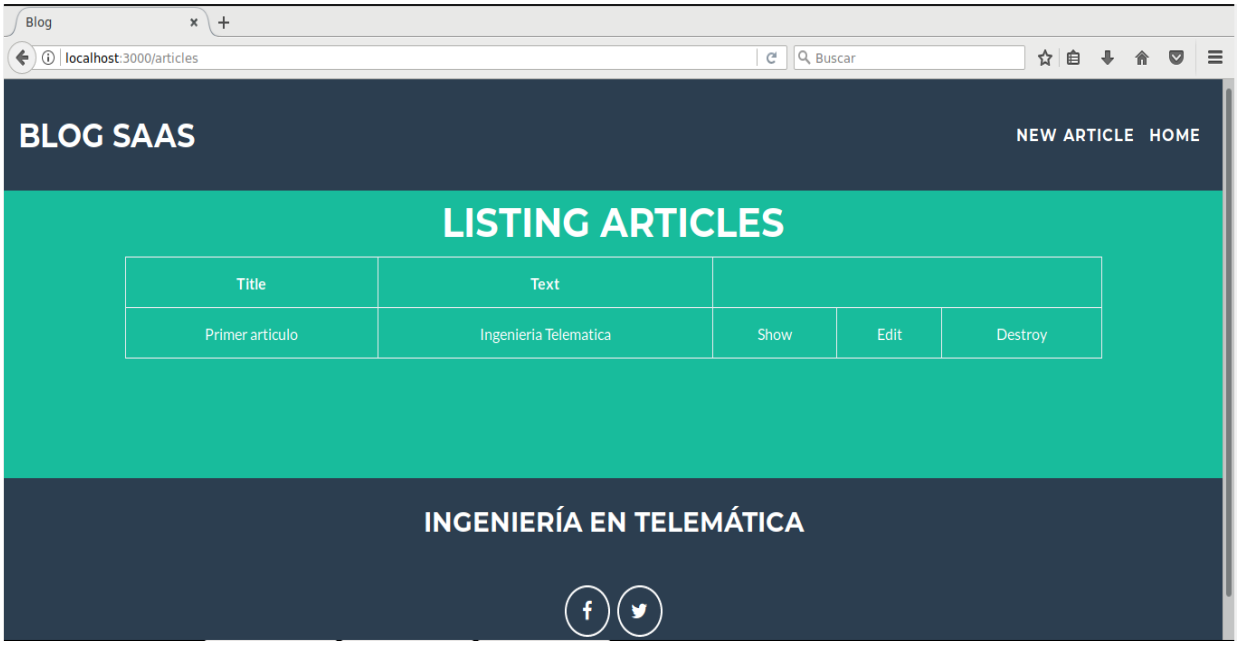

Figura 159 Muestra de las vistas de la aplicación

Figura 160 Vista de nuevo artículo con la plantilla

Extra: Agregar a la aplicación la gema **devise** para generar un Login y poder crear usuarios, realice las validaciones necesarias para que solo los usuarios propietarios de los artículos, los puedan eliminar.

Capítulo V:

Aspectos finales

8. Conclusiones

Finalizando con el presente trabajo monográfico y cumpliendo con la finalidad de los objetivos, se ha llegado a las siguientes conclusiones:

- Se elaboraron un total de 9 guías prácticas y 2 proyecto finales, para el componente curricular Software como un Servicio de la carrera Ingeniería en Telemática, las cuales pueden servir de apoyo tanto a los docentes como a los estudiantes.
- Se ha definido un formato específico utilizado en cada de una de las guías, en las cuales se plantean una serie de ejercicios guiados; posteriormente al final de cada una, se proponen actividades para los estudiantes en donde pongan en práctica lo aprendido en la realización de los ejercicios.
- De igual forma se seleccionaron las herramientas y versiones de software a utilizar para poder elaborar cada una de las guías, entre los cuales se puede mencionar la utilización del framework Ruby on Rails utilizado mayormente para el desarrollo de aplicaciones web, siendo este la principal herramienta para llevar a cabo cada uno de los enunciados propuestos en las guías antes mencionadas.
- Así también se ha logrado idenficar una secuencia exponencial respecto a la temática que se debían abordar en las guías, los cuales fueron tomados de los temas planteados en la microgramación del componente software como un servicio, para que los estudiantes puedan alcanzar los suficientes conocimientos, habilidades y actitudes que se pretende con la elaboración de cada una de las guías presentadas y así poder adquirir las distintas competencias que se plantean en la microprogramación antes mencionada.

Tomando en cuenta las conclusiones antes mencionadas se puede afirmar que se ha logrado elaborar un documento de guías prácticas sencillo, que los estudiantes podrán usar de apoyo para la clase de software como un servicio y así lograr comprender de una mejor manera el componente.

9. Recomendaciones

Al personal docente:

Debido a que el componente curricular Software como un Servicio, no contaba con una documentación de guías para el desarrollo de los laboratorios se recomienda hacer uso del presente documento, utilizando como base las guías propuestas en el mismo, las cuales son el resultado de dicho trabajo.

A los estudiantes:

Usar el presente documento como herramienta y guía en la iniciación en el desarrollo de aplicaciones con el framework Ruby on Rails.

Dar soluciones a cada una de las guías propuestas de manera secuencial, para ir adquiriendo el conocimiento e incorporar ampliaciones o mejoras en las soluciones de dichas guías de manera que las competencias desarrolladas se vayan consolidando.

Mejoras a este trabajo:

Ruby on Rails es un framework amplio, interesante y con una comunidad de desarrolladores que incorporar actualizaciones de manera constante, se considera de gran importancia que se realicen trabajos posteriores para ampliar los temas que no se lograron abordar en el presente documento, al igual que posibles actualizaciones o mejoras a las guías propuestas.

Una de las posibles ampliaciones a este trabajo sería, desarrollo de una aplicación en Ruby on Rails y hacer un “deployment” o despliegue de la aplicación utilizando Heroku, que es una plataforma de servicio en la nube, inicialmente desarrollada para soportar lenguaje Ruby. Esto con la intensión de aumentar las competencias a adquirir por parte de los estudiantes, al poder obtener mayores conocimientos en un ambiente de desarrollo de aplicaciones y servicios más cercano a la realidad.

10. Referencias

Junta de Castilla y León. . (2010). *Cloud Computing.*

ABC tecnología. (16 de 2 de 2015). *ABC tecnología*. Obtenido de http://www.abc.es/tecnologia/consultorio/20150216/abci--201502132105.html

Aguilar, L. J. (2012). Computación en la nube. *Revista del Instituto Español de Estudios Estratégicos*.

Armando Escalante y Borko Furht. (2010). *Handbook of cloud computing.* New York: Springer Science+Business Media,.

Armando Fox y David Patterson. (2012). *Engineering Long Lasting Software.*

Avila Mejía, O. (2011). *Computación en la nube.*

Best MVC Pratices. (s.f.). *yikiframework*. Obtenido de http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices

Chelinsky, D. (2010). *The RSpec Book.* Dallas, Texas.

GoF. (s.f.). *Design Patterns Elements of Reusable Object-Oriented Software.*

Guijardo, L. F. (7 de Diciembre de 2010). *A un click de las TIC*. Obtenido de https://aunclicdelastic.blogthinkbig.com/%C2%BFque-es-bpaas-%C2%BFque-necesidades-empresariales-resuelve/

Hansson, D. H. (Febrero de 2006). Interview with David Heinemeier Hansson from Ruby on Rails.

Hartl, M. (s.f.). *Ruby on Rails Tutorial*. Obtenido de https://www.railstutorial.org/book

Lucia El Asri. (6 de Junio de 2016). *KATADE*. Obtenido de http://katade.com/2016/06/10/gemas-imprescindibles-programar-ruby-on-rails/

Model View Controller History. (26 de Diciembre de 2014). Obtenido de http://wiki.c2.com/?ModelViewControllerHistory

Network working group. (Enero de 2005). *RFC_URI.* Obtenido de https://www.ietf.org/rfc/rfc3986.txt

Poveda, M. E. (2015). *Microprogramación Software como un Servicio.* León.

puppet. (s.f.). *Puppet*. Obtenido de https://puppet.com/docs/puppet/5.0/lang_template_erb.html

Rails . (s.f.). *Rails, Active Record validations*. Obtenido de http://guides.rubyonrails.org/active_record_validations.html

Rails. (s.f.). *RailGuideMigrations*. Obtenido de http://guides.rubyonrails.org/active_record_migrations.html

Rails. (s.f.). *RailsGuide_Routing.* Obtenido de http://guides.rubyonrails.org/routing.html

Rails. (s,f de s,f de s,f). *Railsguide*. Obtenido de http://guides.rubyonrails.org/getting_started.html

RailsGuide_Active_basic. (s.f.). Obtenido de http://www.guiasrails.es/active_record_basics.html

RESTAPI. (s.f.). *RESTAPI*. Obtenido de https://restfulapi.net/

Ruby. (s.f). *Ruby*. Obtenido de http://www.ruby-lang.org/es/about/

Simple Example of MVC (Model View Controller) Design Pattern for Abstraction. (8 de Abirl de 2008). Obtenido de https://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design

Steve Fox. (12 de Noviembre de 2016). *behaviour-driven.org.* Obtenido de http://behaviour-driven.org/

Stewart, B. (29 de 11 de 2001). *Linuxdevcenter*. Obtenido de http://www.linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html

TIOBE. (Febrero de 2018). *TIOBE*. Obtenido de https://www.tiobe.com/tiobe-index/

Wikipedia. (s.f.). *Wikipedia*. Obtenido de https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

11. Anexos

Anexo 1 Cronograma de actividades