



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Aplicación Web de bases de datos usando el Framework Ruby on Rails

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Gestión

Autor: Ismael Bautista Perales

Director: José Vicente Busquets Mataix

Julio 2014

Resumen

Este proyecto está enfocado a la venta online de libros para estudiantes de la ETSINF en la UPV. En esta aplicación web se pueden hacer reservas de libros, dar de alta nuevos ítems en la base de datos accediendo como usuario administrador, gestionar los usuarios de administración y gestionar los pedidos. Además, la aplicación envía correo electrónico al usuario que decida comprar uno o varios de los ejemplares. Como último dato, esta aplicación está internacionalizada (i18n), tanto en inglés como en español, para que pueda ser usada tanto por alumnos españoles, como por alumnos internacionales.

Palabras clave: ruby, rails, ROR, framework, aplicación web, librería.

Tabla de contenidos

- [1. Introducción](#)
- [2. Lenguaje base: Ruby. Definición y características.](#)
- [3. Framework: Ruby on Rails](#)
 - [3.1 Qué es un framework](#)
 - [3.2 Definición de Rails](#)
 - [3.3 MVC](#)
 - [3.4 ¿Por qué Rails? Características.](#)
- [4. Base de datos empleada: MySQL](#)
 - [4.1 Definición](#)
 - [4.2 ¿Por qué MySQL? Características](#)
- [5. IDE Utilizado: Sublime Text](#)
- [6. Control de versiones: Git.](#)
 - [6.1 ¿Qué es el software de control de versiones?](#)
 - [6.2 Definición.](#)
 - [6.3 Git Bash y Gitk.](#)
- [7. El comercio electrónico](#)
 - [7.1 Definición e historia](#)
 - [7.2 Datos de e-commerce en España](#)
- [8. La aplicación](#)
 - [8.1 Análisis](#)
 - [8.1.1 Análisis de requisitos](#)
 - [8.1.2 Casos de uso](#)
 - [8.1.3 Diagramas de flujo](#)
 - [8.1.4 Modelado de la aplicación](#)
 - [8.2 Instalación y configuración de Rails](#)
 - [8.3 Conector MySQL y gema mysql2](#)
 - [8.4 Creación del proyecto](#)
 - [8.5 Scaffolding](#)
 - [8.6 Migraciones](#)
 - [8.7 Rails Server \(Webrick\)](#)
 - [8.8 Tests](#)
 - [8.9 Mailer](#)
 - [8.10 Ajax y jQuery](#)
 - [8.11 Bcrypt](#)
 - [8.12 Internacionalización](#)
- [9. Referencias](#)

1. Introducción

El objetivo de este proyecto es implementar un sistema de gestión de venta de libros en la Escuela de Informática de la UPV, basándose en el framework Ruby on Rails.

En las páginas siguientes abordaré la explicación, de una manera extensa a la par que concisa, de todos los elementos utilizados para realizar la aplicación, para acabar con la explicación en profundidad del proyecto en sí, las opciones de Rails que han sido utilizadas y los pasos para programar la aplicación, de principio a fin.

El uso de Ruby on Rails implica el uso de otras aplicaciones, necesarias para el total funcionamiento de la aplicación:

- Ruby. Lenguaje de programación del lado del servidor. Es el lenguaje en el que se basa el framework Rails.
- MySQL. Se usa este Sistema de Gestión de Base de Datos relacionales.
- Netbeans. Se ha usado esta aplicación como IDE, para realizar la programación de la misma.

Asimismo, para la elaboración de esta memoria se han utilizado los siguientes programas:

- Microsoft Word 2010 y LibreOffice. Se han utilizado estos dos procesadores de texto para esta redacción.
- MySQL Workbench. Programa utilizado para realizar el esquema de la base de datos, modelado y demás.
- yUML. Herramienta online para la generación de UML's, casos de uso y diagramas de actividad.

2. Lenguaje base: Ruby.

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar creando Ruby en 1993, y lo presentó en sociedad en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos. Su implementación oficial es distribuida bajo una licencia de software libre.

Como he comentado antes, el lenguaje fue creado por Yukihiro "Matz" Matsumoto en febrero del año 1993, y presentado al público en 1995. El nombre de Ruby fue impuesto como una broma aludiendo al lenguaje Perl (perla). La última versión estable (a día de hoy, y en la que está basada este proyecto en Rails) es la 2.0.0.

Ruby está orientado a objetos, esto es, todos los tipos de datos son un objeto (clases y tipos incluidos). Toda función es un método. Las variables son referencias a objetos. Además, soporta herencia con enlace dinámico, aunque por el contrario no soporta herencia múltiple. También soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre).

La sintaxis de este lenguaje es bastante similar a la de Perl y Python.

Las principales características de este lenguaje son:

- Orientación a objetos.
- Manejo de excepciones.
- Cuatro niveles de ámbito de variable: global, clase, instancia y local.
- Similitud con Perl en expresiones regulares nativas.
- Fácil portabilidad.
- Amplia librería.
- En tiempo de ejecución soporta alteración de objetos.



3. Framework: Ruby on Rails

3.1 ¿Qué es un Framework?

Un framework es un esquema para el desarrollo y/o implementación de una aplicación. Es decir, un framework nos facilita la programación debido a que muchas funciones que deberían ser escritas a mano en muchas líneas ya vienen implementadas en sus librerías. Nos evita escribir grandes cantidades de código y facilita la interpretación del código (siempre y cuándo se tengan nociones del framework que se está utilizando).

Generalmente, los frameworks están ligados a un lenguaje en concreto, aunque no siempre es así. En este caso Ruby on rails está ligado a Ruby.

Utilizar un framework tiene algunas ventajas, como:

- Facilidad para el programador a la hora de diseñar la estructura global de la aplicación. El framework proporciona esta estructura, la cuál hay que ir programando.
- Facilita la colaboración, ya que está todo más estandarizado y definido según el framework. Es un código menos personal y más automatizado.
- Hay un mayor número de utilidades y librerías adaptadas a frameworks en concreto.
- Aunque al desarrollar una aplicación, se necesita un cierto tiempo y costes iniciales de aprendizaje, a largo plazo se facilita el desarrollo de la aplicación y el mantenimiento de la misma.

3.2 Definición de Rails

Ruby on Rails, también conocido como Rails, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC), el cual abordaremos en el siguiente punto. Al igual que su padre

Ruby, hace uso de la metaprogramación, lo que facilita la legibilidad de su sintaxis.

Rails se distribuye a través de gemas (RubyGems), que es el gestor de paquetes que proporciona: un formato estándar y autocontenido (gem) para poder distribuir programas o bibliotecas en Ruby, una herramienta destinada a gestionar la instalación de éstos, y un servidor para su distribución.

Las gemas son plugins añadidos a nuestros proyectos en Rails, que nos permiten nuevas funcionalidades, nuevas funciones predefinidas o nuevas herramientas para el desarrollo. El listado de gemas se encuentra en la web de RubyForge.

Como servidor, en este caso, utilizaremos WEBrick, incluido en Ruby, del cuál hablaremos más en profundidad en puntos posteriores.

El principal fundamento de Ruby on Rails es el DRY (“Don't repeat yourself”, No te repitas). Esto quiere decir que las definiciones solo deberían realizarse una vez.

En este proyecto vamos a usar la versión 4.0.0 de Rails, ya que me ha parecido la más estable dentro de las últimas actualizaciones realizadas.

3.3 MVC

MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado y donde se separe la lógica de negocio de la interfaz de usuario, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.



Su fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores.

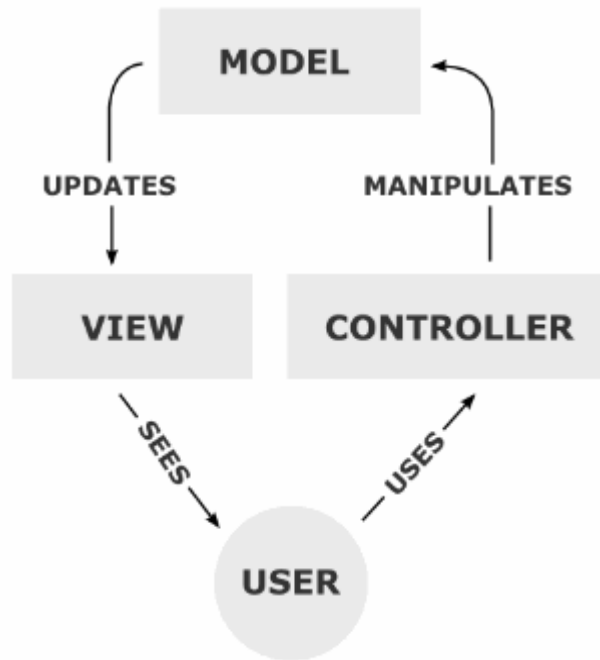


Figura 1: Modelo-Vista-Controlador

Modelos

Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos normalmente estarán en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán las correspondientes consultas (SELECT, UPDATE, INSERT..).

Vistas

Las vistas contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, es decir, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML.

Controladores

Hace de intermediario entre la vista y el modelo, es decir, sirve de enlace entre modelos y vistas para implementar las diversas necesidades del desarrollo. Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etcétera.

El flujo de control que sigue el MVC es el siguiente:

- Se produce la interacción con la interfaz de usuario (click en botón, enlace, por ejemplo).
- El controlador recibe la notificación de la acción solicitada y gestiona el evento (trata el evento de entrada) a través, generalmente, de un gestor de eventos (handler).
- El controlador accede al modelo, actualizándolo o modificándolo si fuera necesario según la acción solicitada por el usuario.
- La vista obtiene sus datos del modelo y genera la interfaz apropiada.
- La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.

3.4 ¿Por qué Rails? Características.

Algunas razones por las que usar Rails en la actualidad para desarrollar webs son:

- Facilidad de uso y aprendizaje. Previo aprendizaje de Ruby, Rails es muy intuitivo y relativamente sencillo. Con respecto al uso, con Rails, como con otros tantos frameworks, se pueden realizar cosas muy complejas en un intervalo de tiempo bastante corto respecto al uso de lenguajes tradicionales.
- Es de código libre y extensible. Día a día se va mejorando por programadores de todo el mundo.
- Uso de ActiveRecord. Con otros lenguajes hay que aprender hasta tres lenguajes diferentes de programación para tener un proyecto bien armado (un lenguaje base, otro de Templating y otro para la base de datos). En Rails



todo gira entorno a Ruby, y nada más. Para las bases de datos se utiliza ActiveRecord, que es una interfaz para la manipulación de objetos dentro de bases de datos para Rails que funciona, también, en torno a Ruby.

- Testing Rspec. Gracias a esta utilidad, los desarrolladores pueden testear su código sin necesidad de tener que iniciar el navegador web y comprobar la aplicación en sí.
- Comunidad Rails. Rails tiene una comunidad muy activa en la red. Hay infinidad de foros y de webs dedicadas a este framework, que sirven de mucha ayuda a la hora de emprender un proyecto, sobre todo si se es principiante.

Las principales características de Rails son las siguientes:

- Orientado a objetos.
- Open Source.
- Tipos dinámicos.
- Manejo de excepciones.
- Altamente portable.
- Hilos de ejecución simultáneos (Green threads).
- No necesita compilador.
- Alteración de objetos en tiempo de ejecución.
- Introspección, reflexión y metaprogramación.
- Facilidad de integración.
- Simplicidad.

4. Base de datos: MySQL

4.1 Definición

Es un sistema administrativo relacional de bases de datos (RDBMS, por sus siglas en inglés, Relational Database Management System). Este tipo de bases de datos puede ejecutar desde acciones tan básicas, como insertar y borrar registros, actualizar información ó hacer consultas simples, hasta realizar tareas tan complejas como la aplicación lo requiera.

MySQL es un servidor multi-usuarios muy rápido y robusto de ejecución de instrucciones en paralelo, es decir, que múltiples usuarios distribuidos a lo largo de una red local o Internet podrán ejecutar distintas tareas sobre las bases de datos localizadas en un mismo servidor.

Utiliza el lenguaje SQL (Structured Query Language) que es el estándar de consulta a bases de datos a nivel mundial.

MySQL ha estado disponible desde 1996, pero su desarrollo data desde 1979 y ha ganado 3 años consecutivos el premio Linux Journal Reader's Choice Award. Actualmente está disponible en código abierto.

4.2 ¿Por qué usar MySQL? Características.

Las principales características del software de base de datos MySQL son:

- **Interioridades y portabilidad**

- Escrito en C y C++
- Funciona en diferentes plataformas.
- APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente multiple CPUs si están disponibles.
- Proporciona sistemas de almacenamiento transaccional y no transaccional.



- Un sistema de reserva de memoria muy rápido basado en threads.
- Tablas hash en memoria, que son usadas como tablas temporales.
- Las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible. Normalmente no hay reserva de memoria tras toda la inicialización para consultas.
- El código MySQL se prueba con Purify (un detector de memoria perdida comercial) así como con Valgrind, una herramienta GPL.

- Sentencias y funciones

- Soporte completo para operadores y funciones en las cláusulas de consultas SELECT y WHERE.
- Los nombres de funciones no colisionan con los nombres de tabla o columna.
- Puede mezclar tablas de distintas bases de datos en la misma consulta.

- Seguridad

- Un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está cifrado cuando se conecta con un servidor.

- Otros

- Gran escalabilidad. Soporta a grandes bases de datos (se conocen usuarios que hacen uso de MySQL Server con 60.000 tablas y 5.000.000.000.000 de registros. Se permiten hasta 64 índices por tabla.
- Conectividad. Los clientes pueden conectarse con el servidor MySQL usando sockets TCP/IP. En Unix, los clientes se pueden conectar usando ficheros socket Unix. Además, los servidores Windows soportan conexiones con memoria compartida, y hay gran variedad de conectores.

Además de las principales características nombradas en el punto anterior, en las que se explican las principales ventajas y características de MySQL, cabe destacar que MySQL es la base de datos de código abierto número uno del mundo, es la base de datos número uno para Web y es una excelente base de

datos embebida. Más de 3.000 ISVs y OEMs, incluyendo 8 de los 10 mayores, y 17 de los 20 principales proveedores de software de todo el mundo confían en MySQL como base de datos de sus productos.



5. IDE: Sublime Text.

Las principales características del IDE usado en esta aplicación son:

- Mini mapa: Es una preview de la estructura de nuestro código que puede ser colocada a un lado del tab o bien puede ser ocultada. Es muy útil para desplazarse por el archivo cuando conocemos bien la estructura del mismo, y sobre todo cuándo contiene muchas líneas de código.
- Multi Selección.
- Multi Cursor: Al usar multi selección Sublime Text nos crea n cursores con los que podemos escribir texto de forma arbitraria en n posiciones diferentes en paralelo.
- Multi Layout: Viene con siete configuraciones de layout donde podemos elegir editar en una sola ventana o hacer split de hasta cuatro ventanas verticales o cuatro ventanas en grid.
- Soporte nativo para muchísimos de lenguajes: Soporta de forma nativa infinidad de lenguajes.
- Búsqueda Dinámica: Se puede hacer búsqueda de expresiones regulares o normal por archivos, por proyectos, por directorios, por una conjunción de ellos o por todo a la vez.
- Auto completado y marcado de llaves: Podemos ir a la llave que cierra o abre un bloque de forma sencilla
- Acceso rápido a línea o archivo.

6. Control de versiones: Git

6.1 ¿Qué es el software de control de versiones?

Se llama control de versiones a la gestión de los diversos cambios que se producen sobre los elementos de algún producto, principalmente software, o sobre su configuración. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el susodicho productp en un momento dado de su desarrollo o modificación. Aunque un sistema de control de versiones puede realizarse de forma manual, se suele realizar con herramientas que facilitan esta gestión, las cuáles son conocidas como sistemas de control de versiones o SVC (del inglés System Version Control). Estos sistemas hacen más fácil e intuitiva la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

Algunos ejemplos de sistemas de control de versiones son: CVS, Subversion, SourceSafe, o el que voy a usar en este proyecto o del que luego profundizaré, Git.

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente.

6.2 Definición.

Git es un sistema de control de versiones diseñado para manejar proyectos muy grandes con velocidad y eficiencia, pero igual de apropiado para repositorios pequeños; es especialmente popular con la comunidad open source, sirviendo como plataforma de desarrollo para proyectos como el Kernel Linux, Ruby on Rails, WINE o X.org.

Git cae en la categoría de herramientas de manejo de código fuente distribuido, similar por ejemplo a Mercurial o Bazaar. Cada directorio de trabajo de Git es un repositorio completo con historial y capacidades totales



de tracking de revisiones, independiente de acceso de red o un servidor central. Aun así, Git es extremadamente rápido y eficiente con el espacio.

Git es un proyecto Open Source cubierto por la GNU General Public License v2. Originalmente escrito por Linus Torvalds y mantenido por Junio C Hamano.

Este programa de control de versiones aporta muchas ventajas, entre las que se destacan las siguientes:

- Es distribuido. Esto tiene dos implicaciones muy importantes:

Independencia. Tú eres un repositorio, no necesitas ninguna información de ningún otro lado para realizar las operaciones. Es decir, no necesitas para nada red.

Mejor integración y comunicación en grupos de desarrollo grandes, pues cada desarrollador puede compartir cambios con el resto sin depender de un servidor central.

- Rapidez.

- Git sigue y almacena contenido, no ficheros como lo hace svn. Esto tiene importantes repercusiones, una positiva y otra quizá menos: la menos positiva es que no soporta almacenar directorio vacíos y la más positiva es la detección transparente de movimiento de ficheros (renombrado, copias, movimientos, etc...).

- Nuevo concepto, el index. El índice es un punto intermedio entre nuestros ficheros reales en el directorio de trabajo y el repositorio. Permite ser usado como un punto intermedio en el que almacenar nuestro cambios. En él podemos guardar nuestras modificaciones de forma temporal y recuperarlas posteriormente.

6.3 Git Bash y Gitk.

Tras la instalación de Git, una de las opciones de que disponemos es Git Bash.

Un bash, desde el punto de vista informático, es un programa informático que tiene la función de interpretar órdenes, basado en la shell de Unix. Suele

ser el interprete de comandos preestablecido en la mayoría de las distribuciones de Linux.

Git Bash es una consola desde la cuál se interpretan los comandos de Git. Además, en esta consola se pueden ejecutar los comandos más habituales de Linux (ls, cd, mkdir, rm, pwd, etc.).

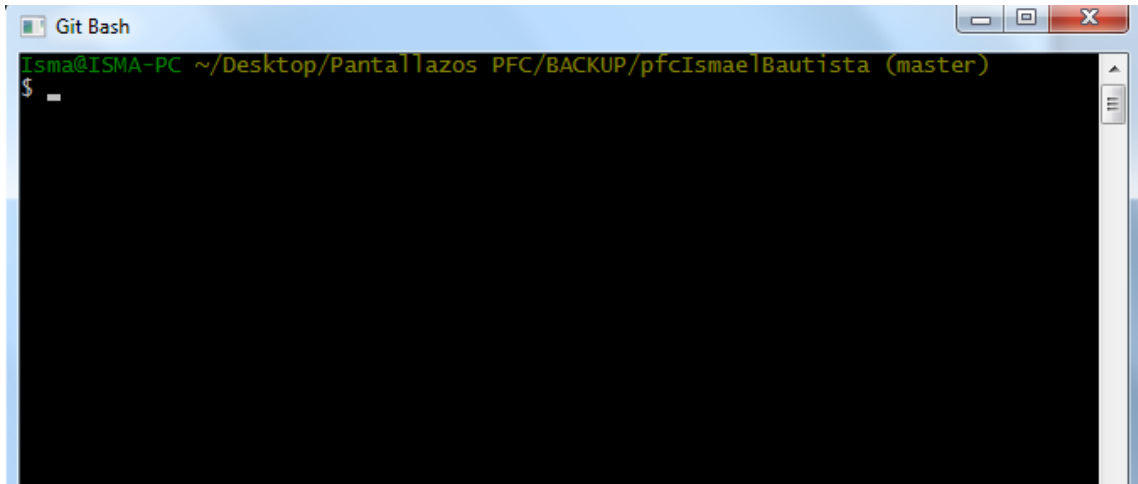


Figura 2. Git Bash

Git dispone de herramienta gráfica para visualizar el histórico de confirmaciones. Es gitk, un programa Tcl/Tk que se distribuye junto con Git. Gitk es básicamente un git log visual, y acepta casi todas las opciones de filtrado que acepta git log.

Al teclear gitk en el prompt, visualizaremos el estado de nuestro proyecto:

Aplicación Web de bases de datos usando el Framework Ruby on Rails

The screenshot shows the Gitk application interface. At the top, there's a menu bar with 'Archivo', 'Editar', 'Vista', and 'Ayuda'. Below it, a commit history list shows several commits by 'Ismael Bautista' with dates ranging from 2014-06-15 to 2014-07-06. The selected commit is 'Commit previo a entrega a Director PFC 27/06. Cambio en pagos'. Below the history, there's a search bar and a diff view for the file 'app/models/order.rb'. The diff shows changes to the 'TYPE_OF_PAYMENT' array, adding 'Tarjeta de crédito' and 'Paypal' to the existing 'Metálico'.

```
gitk: pfdismaelBautista
Archivo  Editar  Vista  Ayuda
● master Internacionalizacion finalizada. Aplicacion finalizada
● Commit previo a entrega a Director PFC 27/06. Cambio en pagos
● Nuevas vistas de Acerca de y Contacto. Mas modificaciones de estilos y pequenyos detalles. Arreglados pequenyos fallos
● Diversos cambios diseo. Imagen BG. Cambio estilos. Pequenyos detalles.
● Modificacion estilos de la pagina. Imagenes con botones. Creacion Controller Main y cambio en las rutas para que sea pagina principal. Cambio en todos los controladores de la ruta store_url por s
● Tareas de logueado hechas. Botones de log in y log out. Anyadir usuarios y autentificacion. Limitacion de acceso. Menu mas amplio para usuario administrador
● Formulario de venta. Envio por finn xD de email automatico al recibir el pedido
● Cositas con Ajax y JQuery
● Cositas con Ajax y JQuery
● Cambios. Manejo de errores, boton de borrar del carrito
● Adelantos en la Cart. Combinacion de items. Pendiente manejo de errores

SHA1 ID: c592def9d87222e10d870f0dce36a1f754544079  Row 2 / 13

Buscar << >>  revisión que contiene:  Exacto  Todos los campos

Diferencia  Versión antigua  Versión nueva  Líneas de contexto: 3  Ignora cambios de espaciado  Line diff
Autor: Ismael Bautista <ismael.bautista87@gmail.com> 2014-06-27 16:25:36
Committer: Ismael Bautista <ismael.bautista87@gmail.com> 2014-06-27 16:25:36
Padre: 9556ac6a94f97b09cccc9a0bc03ce74a47cfc1c5 (Nuevas vistas de Acerca de y Contacto. Mas modificaciones de estilos y pequenyos deta
Hija: 3259e36d75db04d3fa6f0918702f8dd41119014 (Internacionalizacion finalizada. Aplicacion finalizada)
Rama: master
Sigue-a:
Precede-a:

Commit previo a entrega a Director PFC 27/06. Cambio en pagos

----- app/models/order.rb -----
index 630401e..b1e1986 100644
@@ -2,7 +2,7 @@ class Order < ActiveRecord::Base
  #Los items que pertenecen a un pedido se destruyen cuando este se destruye
  has_many :line_items, dependent: :destroy

-  TYPE_OF_PAYMENT = ["Metálico", "Tarjeta de crédito", "Paypal"]
+  TYPE_OF_PAYMENT = ["Metálico", "Paypal"]

  #VALIDAMOS QUE ESTEN LOS DATOS
  validates :name, :address, :email, presence: true
```

Figura 3. Gitk. Árbol de proyecto

Como se observa, podemos observar el histórico de confirmaciones en la parte superior. En la parte inferior se muestran las modificaciones introducidos en cada modificación, según sea seleccionada en la parte superior.

7. El comercio electrónico

7.1 Definición e historia.

El comercio electrónico, o “e-commerce”, es la transacción o compra de productos, servicios, o similares, a través de medios electrónicos, principalmente de Internet.

Ello incluye intercambio de bienes, servicios e información electrónica. Incluye también las actividades de promoción y publicidad de productos y servicios, campañas de imagen de las empresas, marketing en general, facilitación de los contactos entre los agentes de comercio, soporte post-venta, seguimiento e investigación de mercados, concursos electrónicos y soporte para compartir negocios.

El comercio electrónico se caracteriza por tener tres capas, las cuáles se complementan e interrelacionan entre sí:

- Capa Logística. Integración de las cadenas logísticas de aprovisionamiento y distribución
- Capa Transaccional. Capa que hace posible el intercambio de información en formato electrónico.
- Capa Financiera. Capa que hace referencia a los medios de pago, asociada a los intercambios de información, bienes y servicios.

7.2 Datos de e-commerce en España

Principales claves del “e-commerce” en España y su comparativa con el resto de Europa:

- 1) Porcentaje de compras: el 31% de los españoles compró algún producto a través de Internet para consumo privado durante 2012. Sin embargo, España se mantiene por debajo de la media europea, que en 2012 fue del 45%.



2) Devoluciones: es un punto que lidera España. Es el segundo país en donde más consumidores están al tanto de los tiempos de devolución de que disponen en la venta digital (81% frente al 69% europeo).

3) Garantía online: otro punto fuerte para nuestro país. España es el tercero en saber la validez de las garantías (77%, frente al 56% de la media UE).

4) Incoherencia: los compradores online españoles no saben qué hacer si reciben algún artículo que no han solicitado, ya que sólo un 15% responde correctamente a esta pregunta, la mitad que la media europea.

5) Protección ante abusos: el 75% de los españoles confía en las organizaciones de consumidores para proteger sus derechos, la misma cifra que a escala europea;

6) Centros europeos especiales: tan solo un 9% de españoles conoce los Centros Europeos para el Consumidor y un 16% recurrió al mecanismo de arbitraje para resolución de disputas (ADR, por sus siglas en inglés).

Otros datos:

- Sectores empresariales con mayor demanda online:

Hostelería y turismo lideran la lista de sectores más demandados. Mientras, los productos con mayor crecimiento en la demanda digital son la ropa, el material deportivo y los bienes del hogar.

- Crecimiento de PayPal:

Las plataformas de pago mantienen una tendencia creciente gracias al crecimiento de la confianza de los consumidores y la pérdida del miedo a las compras online.

- Procesos de compra:

El inconveniente más frecuente que se han encontrado los compradores online ha sido el incumplimiento en los plazos de entrega

- Productos:

Los internautas se quejan de las descripciones en las características de los productos, que en ocasiones difieren de la realidad.

8. La aplicación

8.1 Análisis.

8.1.1 Análisis de requisitos

El sistema a implementar se basa en una página de venta de libros para un público universitario, congregado en la Escuela de Informática de la Universidad Politécnica de Valencia. En esta página, cualquier alumno puede comprar libremente cualquier ejemplar, sin necesidad de registro ni log-in. Asimismo, la aplicación web dispone de un usuario web (o varios) que se encargan de la gestión interna de la misma.

- Cualquier alumno puede comprar cuantos libros quiera. Se identificará en un formulario, con su nombre, apellidos y correo electrónico.
- Ningún usuario que no sea administrador podrá realizar labores de usuario administrador en la aplicación web.

El usuario administrador podrá dar de alta nuevos ítems para la venta, modificarlos o eliminarlos; también podrá gestionar los pedidos entrantes y crear nuevos usuarios administrador, si lo necesitara, otorgándoles un nombre de usuario y contraseñas únicos.

El sistema recoge los datos de una base de datos relacional, manipulable por el administrador y modificable, testeada comprobando todos los campos de una manera robusta. No se pueden corromper las tuplas de una manera sencilla con un conocimiento medio/alto de informática.

Cuándo el cliente compre y realice el pago del producto, actualmente recibirá un correo electrónico en el e-mail indicado en el formulario de la compra, en el que se detallarán los artículos obtenidos.

8.1.2 Casos de uso

De los requisitos explicados en el punto anterior se extraen los siguientes casos de uso:

8.1.2.1 Caso de uso del administrador al iniciar sesión

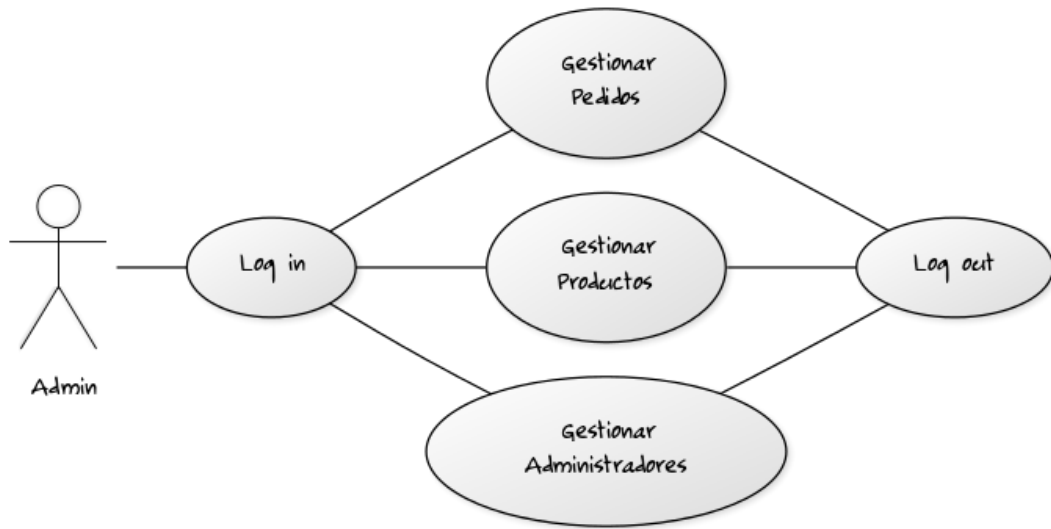


Figura 4. Caso de uso del administrador.

8.1.2.2 Caso de uso del cliente en la aplicación

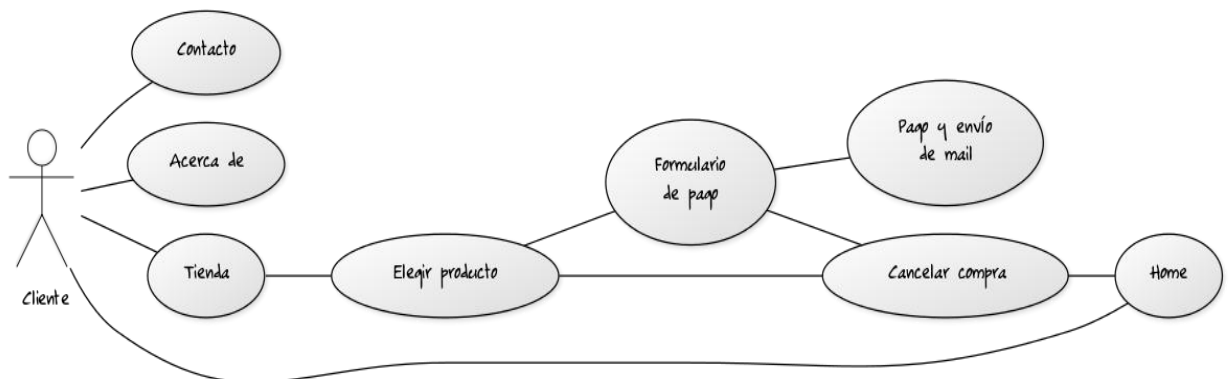


Figura 5. Caso de uso del cliente.

8.1.3 Diagramas de flujo

De los casos de uso se derivan los siguientes diagramas de flujo:

8.1.3.1 Diagrama de flujo del comprador en nuestra web

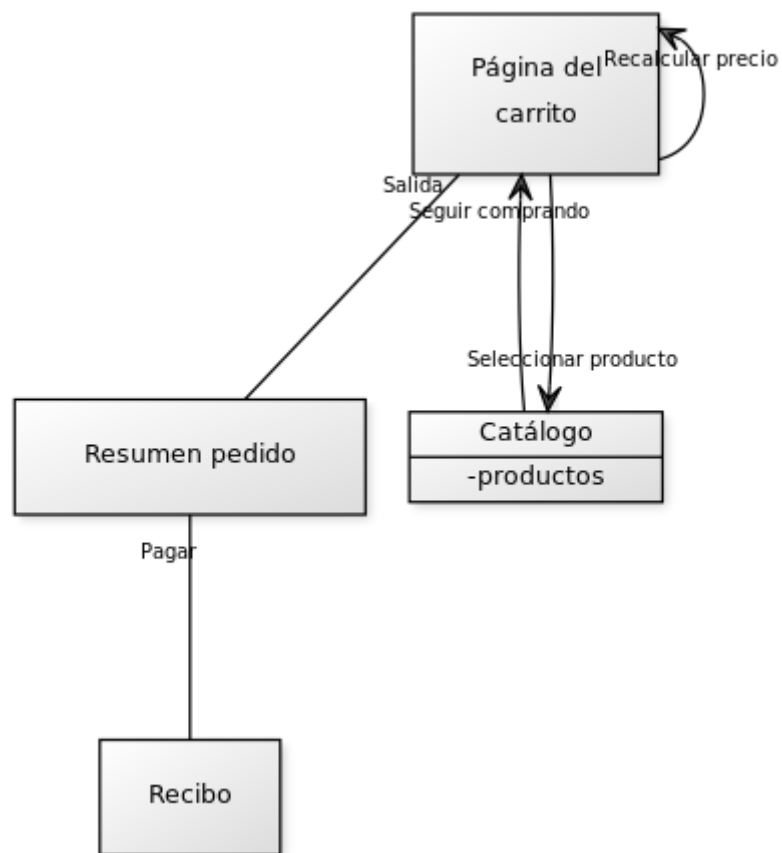


Figura 6. Diagrama de flujo del comprador.

8.1.3.2 Diagrama de flujo de las páginas del vendedor

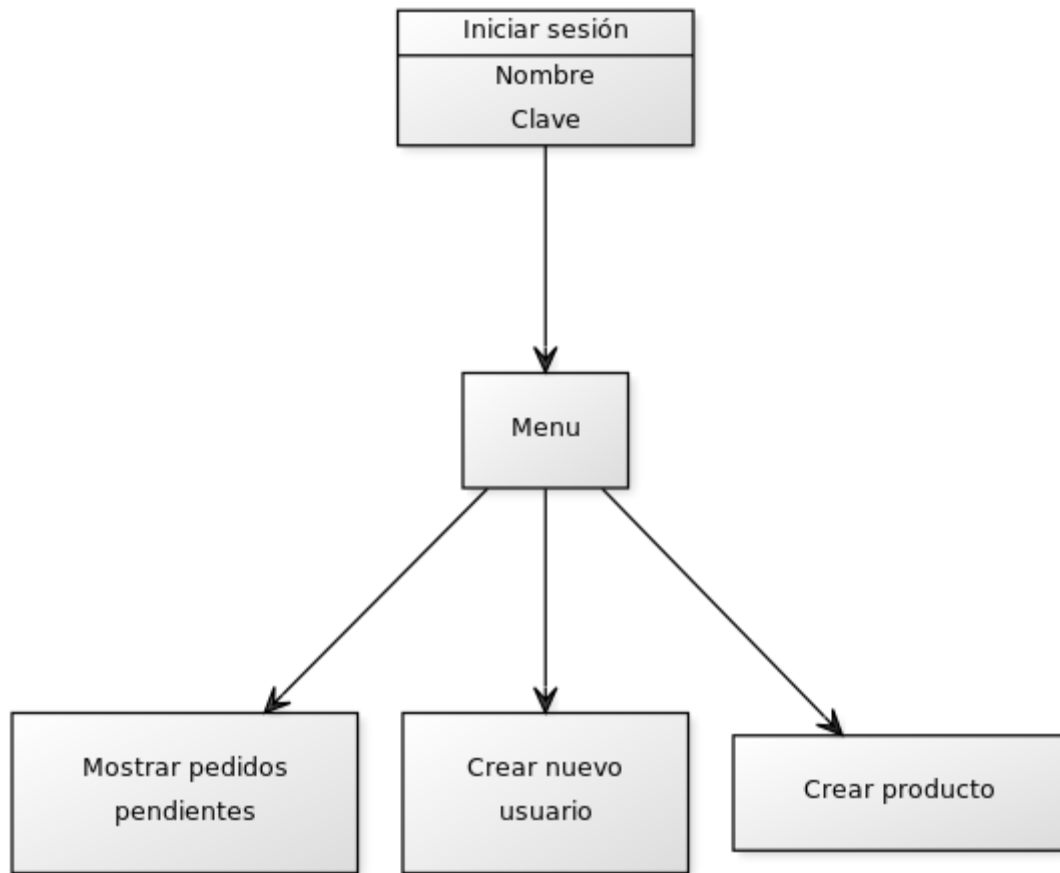


Figura 7. Diagrama de flujo del administrador.

8.1.4 Modelado de la aplicación

Conforme al análisis de requisitos, el modelado de nuestra base de datos queda de la siguiente manera:

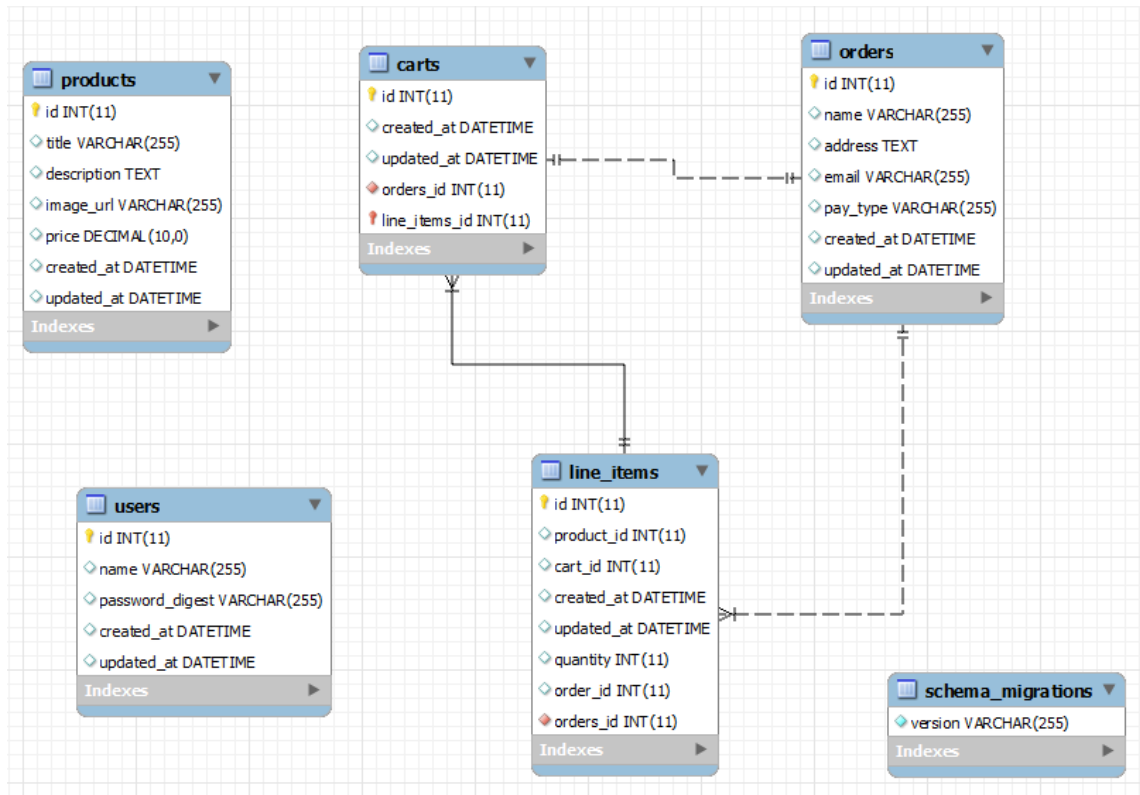


Figura 8. Modelado de nuestra base de datos.

8.2 Instalación y configuración de Rails

Para utilizar Rails en este proyecto, he utilizado el autoinstalable Railsinstaller, concretamente la versión Railsinstaller-windows 3.0.0-alpha2, el cuál instala tanto el lenguaje Ruby como el framework Rails de una manera automática.

En esta versión se instalan la versión Ruby 2.0.0 y la versión Rails 4.0.0 .

Este paquete tiene otros paquetes interesantes que se instalan, además de los dos mencionados anteriormente, entre los que cabe destacar Bundle, Git (control de versiones mencionado en puntos anteriores) o la base de datos Sqlite.



Figura 9. Railsinstaller.

8.3 Conector MySQL y gema mysql2

La base de datos por defecto en Rails es SQLite.

Primeramente, para usar MySQL, debemos instalar su entorno en nuestra máquina. En este proyecto se ha usado la versión MySQL 5.5. Es importante indicar una contraseña para el usuario root, y apuntarla bien, ya que será utilizada continuamente a lo largo del proyecto.

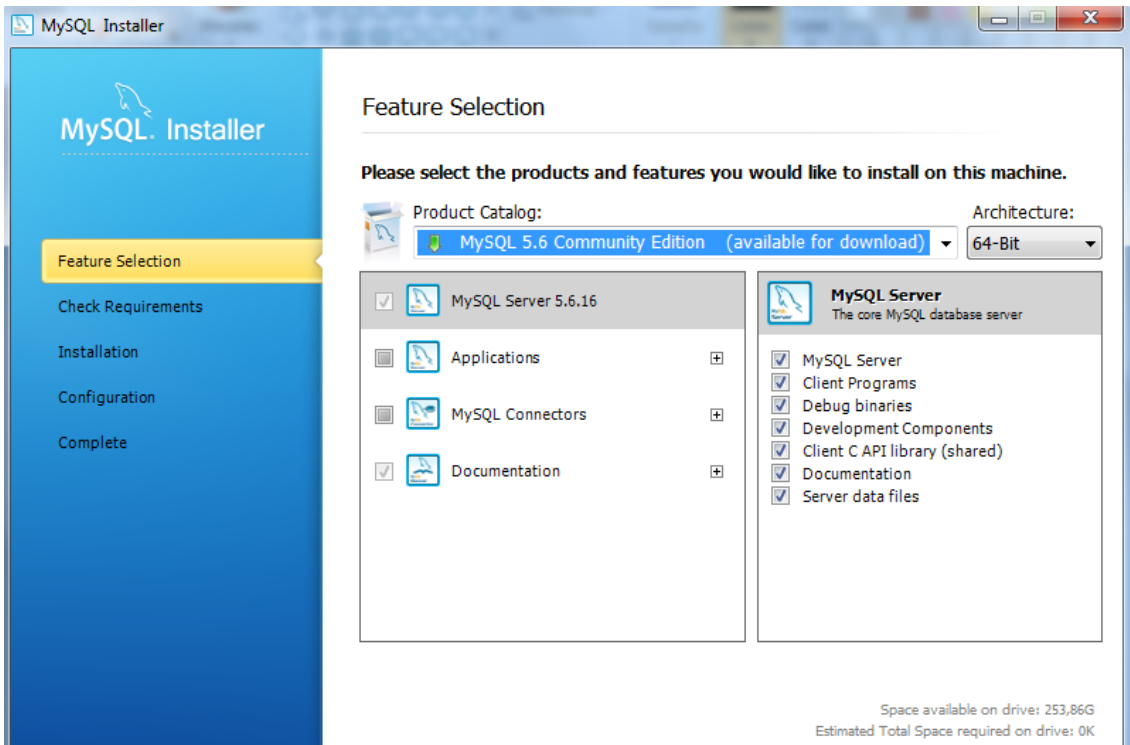


Figura 10. MySQL Installer.

Una vez instalado MySQL en nuestra máquina, debemos indicarle al PATH de sistema la ruta de los ejecutables de MySQL.

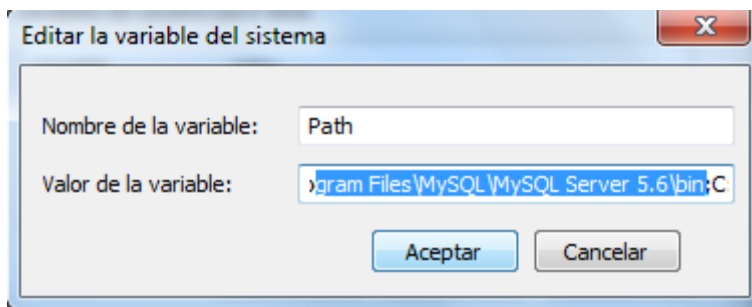


Figura 11. Editando el path MySQL.

Si queremos usar la gema `mysql2` de Rails, tenemos que instalar un conector de MySQL ligado a nuestro entorno Rails, y configurarlo.

El conector se descarga de la página oficial de MySQL, al igual que el instalador anterior. En este proyecto, hemos utilizado el conector más estable: `mysql-connector-c-noinstall-6.0.2-win32.zip`.

Una vez descargado, lo descomprimos en la ruta que nos sea más cómoda. En este caso, se ha dejado en el directorio C:\. Desde el prompt, ejecutamos la siguiente orden, que instala la gema necesaria para ejecutar MySQL en nuestro entorno Rails, mysql2:

```
gem install mysql2 --no-ri --no-rdoc -- --with-mysql-dir=c:\mysql-connector-c-noinstall-6.0.2-win32
```

Ya tenemos la gema instalada. Como último paso para acabar la instalación, debemos de poner en el PATH de sistema la ruta de libmysql.dll del conector:

```
c:\mysql-connector-c-noinstall-6.0.2-win32\lib
```

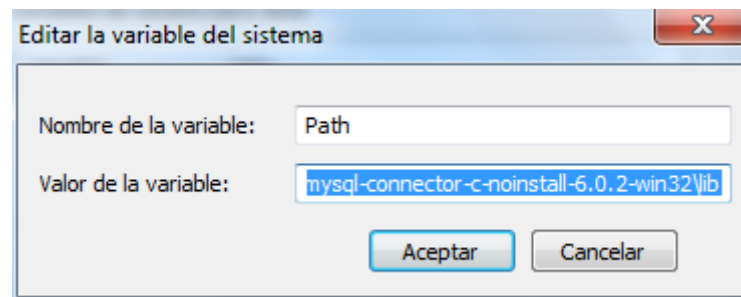


Figura 12. Editando el path del conector.

Ya tenemos preparado MySQL para ser utilizado en nuestra máquina con Rails.

8.4 Creación del proyecto

Una vez instalado todo el entorno, vamos a proceder a crear nuestra aplicación.

En primer lugar, vamos a crear la base de datos que vamos a utilizar (development).

```
mysqladmin -u root -p create pfcismaelbautista_development
```

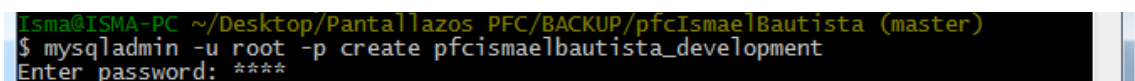
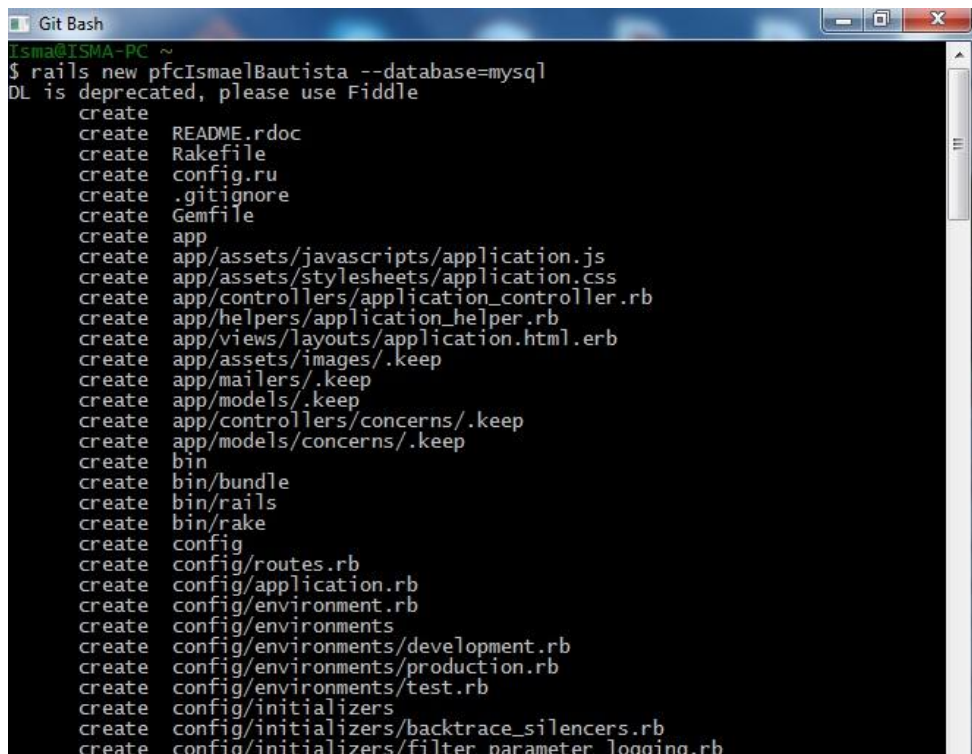


Figura 13. Creación de la base de datos.

En la misma línea de comandos de creación del proyecto, ya vamos a indicar que vamos a usar MySQL como base de datos:

```
rails new pfcIsmaelBautista --database=mysql
```

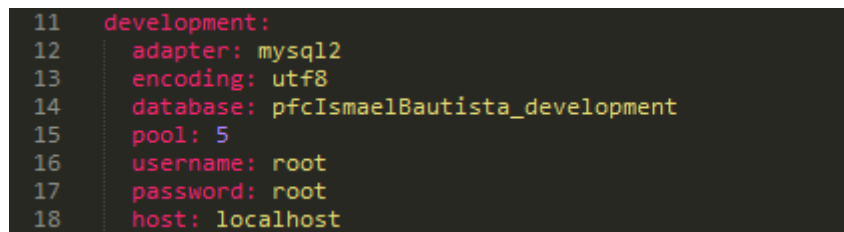


```
Git Bash
[isma@ISMA-PC ~]
$ rails new pfcIsmaelBautista --database=mysql
DL is deprecated, please use Fiddle
create
create  README.rdoc
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
create  app
create  app/assets/javascripts/application.js
create  app/assets/stylesheets/application.css
create  app/controllers/application_controller.rb
create  app/helpers/application_helper.rb
create  app/views/layouts/application.html.erb
create  app/assets/images/.keep
create  app/mailers/.keep
create  app/models/.keep
create  app/controllers/concerns/.keep
create  app/models/concerns/.keep
create  bin
create  bin/bundle
create  bin/rails
create  bin/rake
create  config
create  config/routes.rb
create  config/application.rb
create  config/environment.rb
create  config/environments
create  config/environments/development.rb
create  config/environments/production.rb
create  config/environments/test.rb
create  config/initializers
create  config/initializers/backtrace_silencers.rb
create  config/initializers/filter_parameter_logging.rb
```

Figura 14. Creación del proyecto.

Al ejecutar este comando, se crea el proyecto, tomando como base de datos la gema que hemos instalado antes, mysql2.

Acto seguido, debemos poner la contraseña que hemos asignado a nuestro usuario root de MySQL en el fichero database.yml .



```
11  development:
12    adapter: mysql2
13    encoding: utf8
14    database: pfcIsmaelBautista_development
15    pool: 5
16    username: root
17    password: root
18    host: localhost
```

Figura 15. Configuración database.yml.

Con estas acciones, la aplicación ya está preparada para trabajar con y en ella.



8.5 Scaffolding

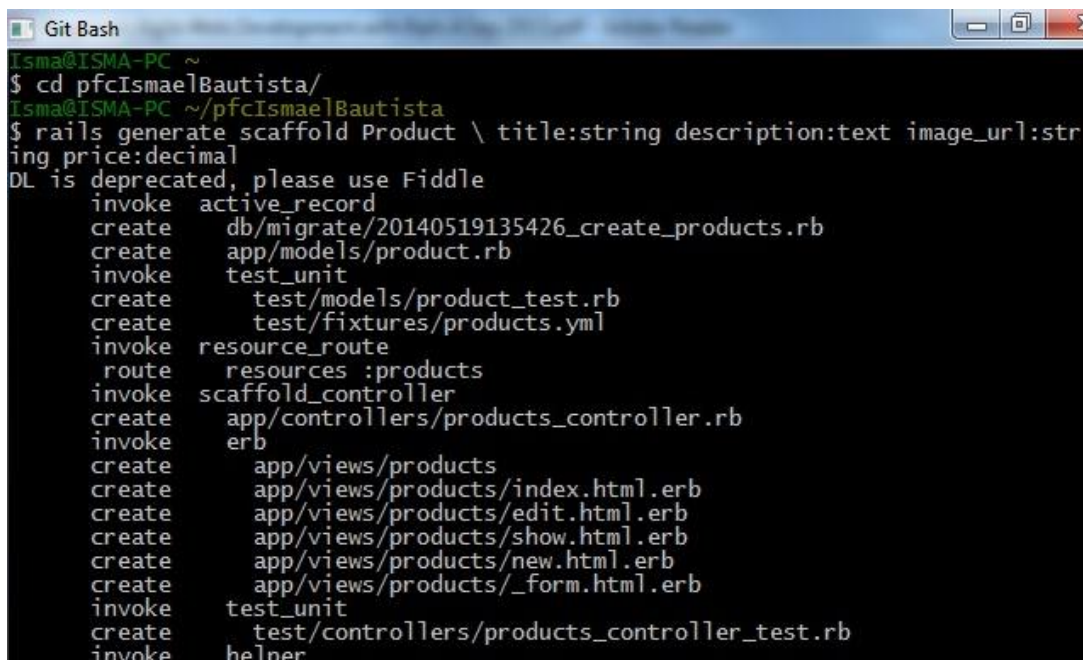
El scaffolding es una característica de Rails que nos permite tener las funcionalidades básicas de administración de un modelo en un controlador. Estas funcionalidades son las siguientes: CRUD (Create, Retrieve, Update, Delete).

El scaffolding como tal sirve para implantar de manera inmediata un entorno de administración temporal sobre el que trabajar, es un método para construir aplicaciones basadas en bases de datos, soportada generalmente por frameworks que siguen el MVC (como en este caso) en el cuál el programador escribe una especificación que describe cómo debe ser la base de datos.

Ejemplo de uso en esta aplicación es la creación del scaffold Product, tal y como se describe en la imagen inferior. En ella creamos la tabla products, con las tuplas title, description e image_url, indicando en cada una de ellas el tipo al que pertenecen.

También podemos usar la siguiente línea de comandos:

```
rails g scaffold Product title:string description:text image_url:string price:decimal
```



```

Git Bash
Isma@ISMA-PC ~
$ cd pfcIsmaelBautista/
Isma@ISMA-PC ~/pfcIsmaelBautista
$ rails generate scaffold Product \ title:string description:text image_url:string price:decimal
DL is deprecated, please use Fiddle
  invoke  active_record
  create  db/migrate/20140519135426_create_products.rb
  create  app/models/product.rb
  invoke  test_unit
  create  test/models/product_test.rb
  create  test/fixtures/products.yml
  invoke  resource_route
  route   resources :products
  invoke  scaffold_controller
  create  app/controllers/products_controller.rb
  invoke  erb
  create  app/views/products
  create  app/views/products/index.html.erb
  create  app/views/products/edit.html.erb
  create  app/views/products/show.html.erb
  create  app/views/products/new.html.erb
  create  app/views/products/_form.html.erb
  invoke  test_unit
  create  test/controllers/products_controller_test.rb
  invoke  helper

```

Figura 16. Scaffolding.

También es interesante comentar que modificando el archivo seeds.rb, situado en el directorio db, podemos hacer una carga de artículos nuevos en nuestra base de datos, y por tanto en nuestra aplicación. Simplemente hay que modificar el archivo a nuestro gusto, como por ejemplo, en la imagen inferior.

```
1 # This file should contain all the record creation needed to seed the database with its default values.
2 # The data can then be loaded with the rake db:seed (or created alongside the db with db:setup).
3 #
4 # Examples:
5 #
6 # cities = City.create([ { name: 'Chicago' }, { name: 'Copenhagen' } ])
7 # Mayor.create(name: 'Emanuel', city: cities.first)
8 Product.delete_all
9 Product.create!(title: 'Yamaha M1',
10 description:
11   %{\<p>
12     Prueba de Yamaha M1.
13   \</p>},
14 image_url: 'cs.jpg',
15 price: 12000.00)
16
17
18 Product.create!(title: 'Yamaha YZR',
19 description:
20   %{\<p>
21     Prueba de Yamaha YZR
22   \</p>},
23 image_url: 'ruby.jpg',
24 price: 16500.95)
25
26
27 Product.create!(title: 'Honda CBR 600',
28 description:
29   %{\<p>
30     <em>Prueba de Honda CBR</em>
31     Es de 600.
32   \</p>},
33 image_url: 'rtp.jpg',
34 price: 7000.95)
```

Figura 17. Seeds

Y luego ejecutar el comando :

```
rake db:seeds
```

8.6 Migraciones

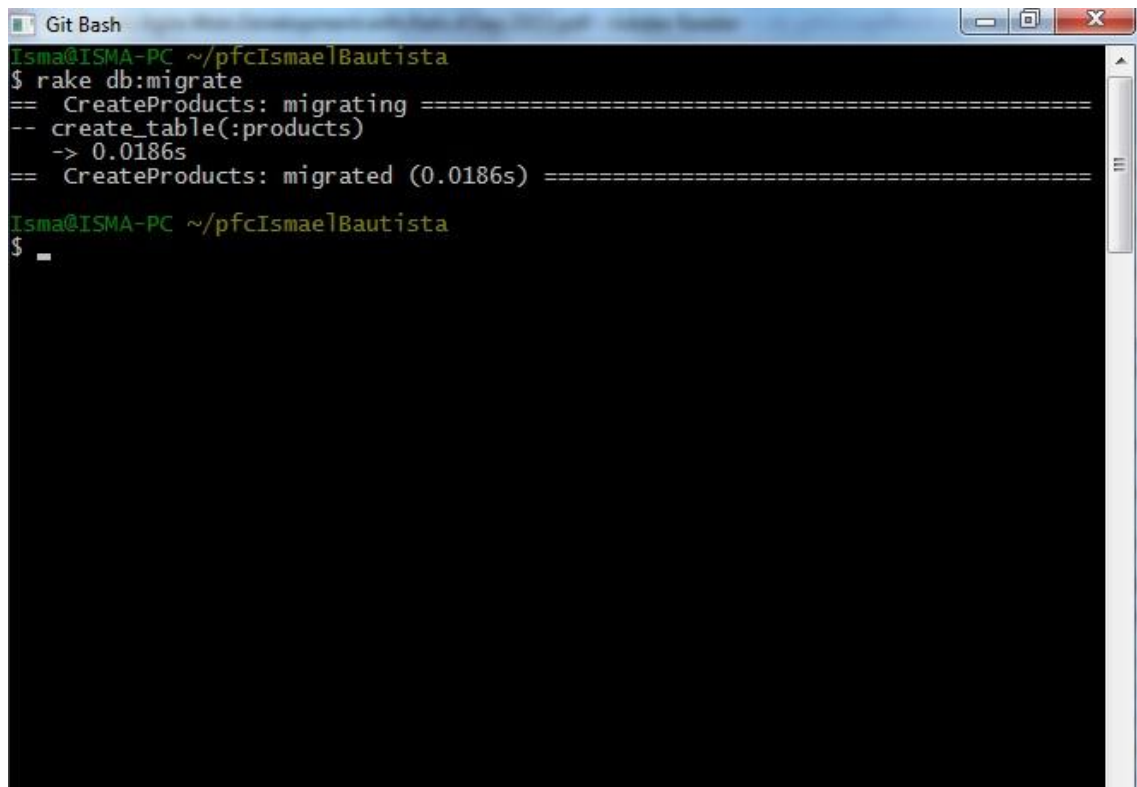
Una vez realizado el scaffold, y realizado los cambios si lo creemos oportuno en el fichero creado en la carpeta de migraciones, debe realizarse la migración del scaffold para que se acabe de crear totalmente la base de datos y se apliquen los cambios en ella.



Las migraciones nos permiten crear y modificar nuestra base de datos de una manera clara, organizada y estructurada. Además, reduce muchísimo el uso de SQL, lo cual facilita el trabajo.

En este ejemplo, realizamos la migración del scaffold creado en el apartado anterior, con el comando:

```
rake db:migrate
```

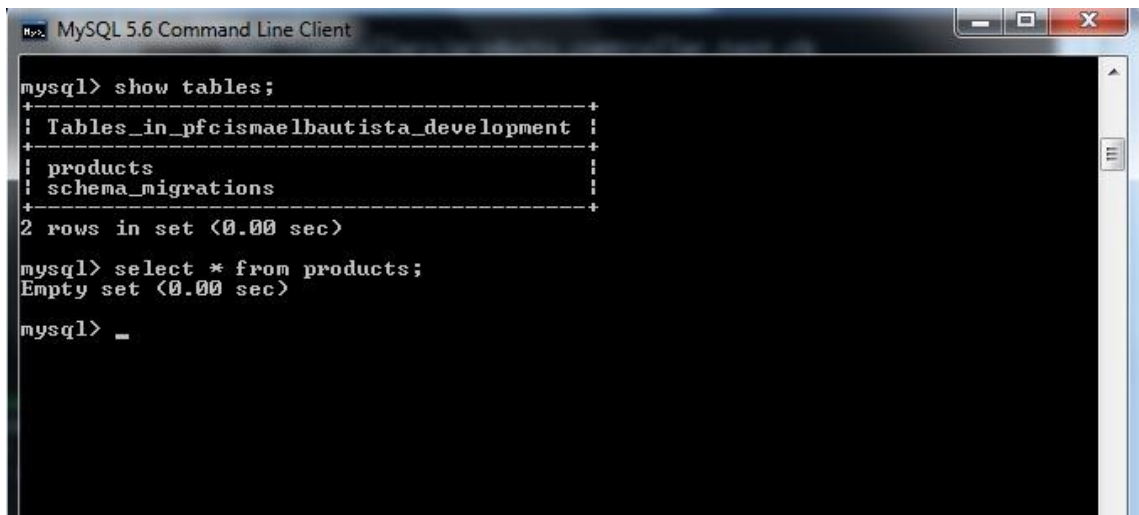


```
Git Bash
Isma@ISMA-PC ~/pfcIsmaelBautista
$ rake db:migrate
== CreateProducts: migrating =====
-- create_table(:products)
   -> 0.0186s
== CreateProducts: migrated (0.0186s) =====

Isma@ISMA-PC ~/pfcIsmaelBautista
$
```

Figura 18. Migración.

Una vez realizada la migración, nos conectamos a nuestra base de datos y observamos que la tabla se ha creado:



```
mysql> show tables;
+-----+
| Tables_in_pfcismaelbautista_development |
+-----+
| products                                 |
| schema_migrations                       |
+-----+
2 rows in set (0.00 sec)

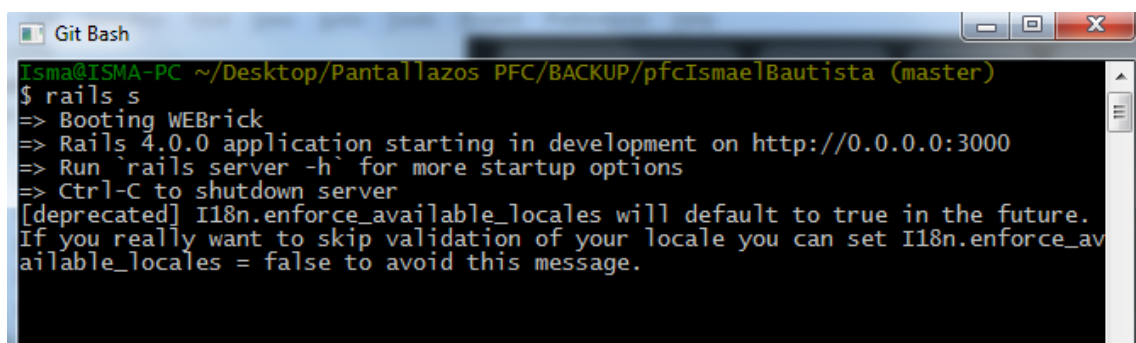
mysql> select * from products;
Empty set (0.00 sec)

mysql> _
```

Figura 19. MySQL Command Line Client.

8.7 Rails server (Webrick)

Rails tiene un servidor propio para nosotros, que nos facilita mucho la tarea. Se llama WEBrick, y es una librería de Ruby que nos proporciona los servicios básicos de un servidor HTTP. Es usado en Rails para testear las aplicaciones en un entorno de desarrollo, y también en el de producción.



```
Git Bash
isma@ISMA-PC ~/Desktop/Pantallazos PFC/BACKUP/pfcIsmaelBautista (master)
$ rails s
=> Booting WEBrick
=> Rails 4.0.0 application starting in development on http://0.0.0.0:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
[deprecated] I18n.enforce_available_locales will default to true in the future.
If you really want to skip validation of your locale you can set I18n.enforce_av
ailable_locales = false to avoid this message.
```

Figura 20. Rails Server.

Una vez arrancado, vamos en nuestro navegador a la ruta:

<http://localhost:3000/> y nos aparecerá la página de inicio de proyecto de Rails (en este caso es la que viene por defecto, antes de tocar nada).



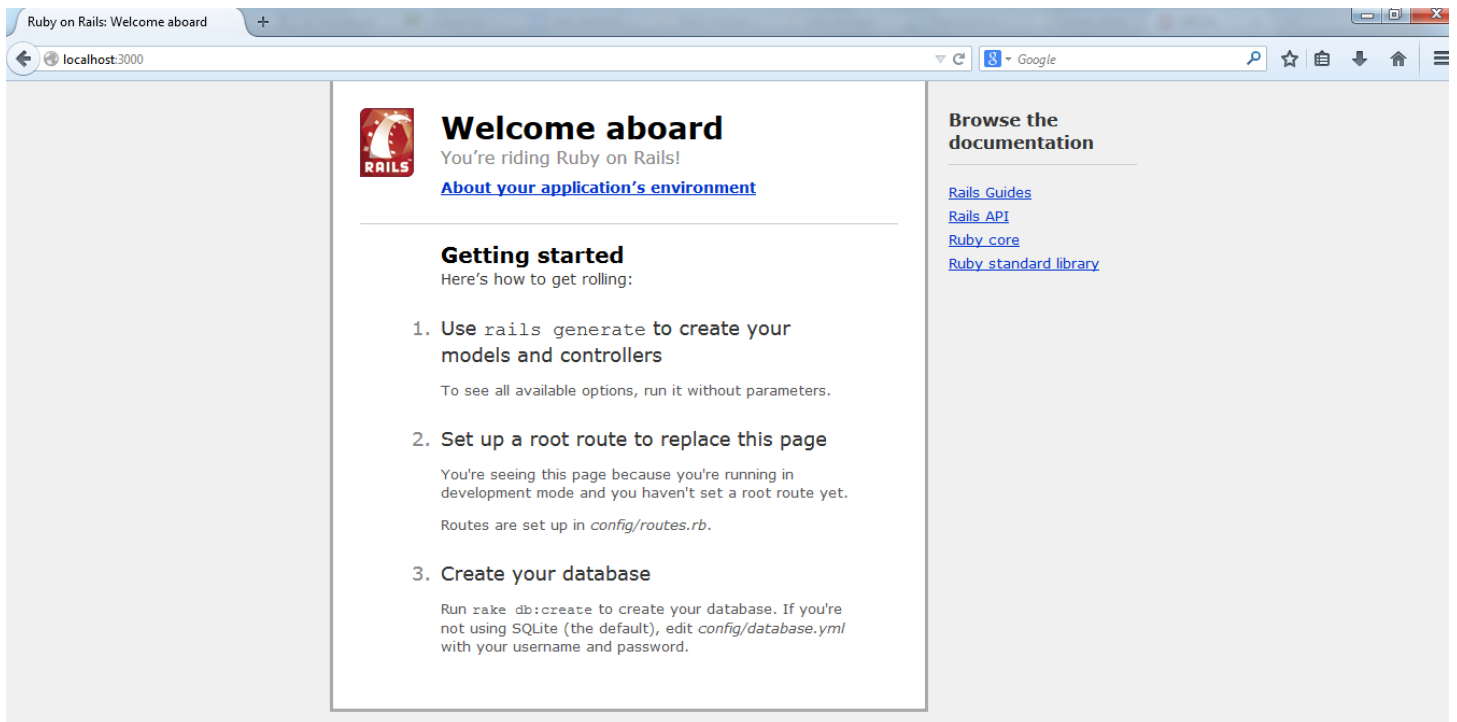


Figura 21. Página por defecto Rails

8.8 Tests

Diseñar nuestra aplicación en base a pruebas nos da muchísimos beneficios, debido a que evalúan los métodos que nosotros creamos oportunos, evaluando si funcionan correctamente o no.

Son fáciles de ejecutar, sólo con escribir una pequeña orden en nuestro bash se realizan.

En este proyecto se han realizado pruebas en el entorno de desarrollo. La estructura de estas pruebas está alojada en la carpeta Test, dentro de nuestra aplicación de Rails. La estructura es la siguiente:

- Models. Se hacen las pruebas de nuestros modelos de la aplicación.
- Controllers. Se hacen las pruebas de los controladores.
- Integration: Se hacen las pruebas que involucran la integración con nuestra aplicación.

- Fixtures: Aquí organizamos nuestros datos de prueba.
- Test_helper.rb: Archivo que contiene la configuración por defecto para los análisis en las pruebas.

Ejemplo de ejecución de los test:

Rake test

```

Git Bash
Isma@ISMA-PC ~/Desktop/Pantallazos PFC/BACKUP/pfcIsmaelBautista (master)
$ rake test
[deprecated] I18n.enforce_available_locales will default to true in the future.
If you really want to skip validation of your locale you can set I18n.enforce_ava
ailable_locales = false to avoid this message.
Run options: --seed 23442

# Running tests:

.....

Finished tests in 4.175781s, 11.4949 tests/s, 21.5529 assertions/s.
48 tests, 90 assertions, 0 failures, 0 errors, 0 skips
Isma@ISMA-PC ~/Desktop/Pantallazos PFC/BACKUP/pfcIsmaelBautista (master)
$

```

Figura 22. Rake test.

Esta aplicación ha sido testada con un gran número de pruebas, siendo todas satisfactorias, garantizando su robustez y su tolerancia a fallos.

```

25 test "should create product" do
26   assert_difference('Product.count') do
27     #post :create, product: { description: @product.description, image_url: @product.image_url, price: @product.price, tit
28     post :create, product: @update
29   end
30
31   assert_redirected_to product_path(assigns(:product))
32 end
33
34 test "should show product" do
35   get :show, id: @product
36   assert_response :success
37 end
38
39 test "should get edit" do
40   get :edit, id: @product
41   assert_response :success
42 end
43
44 test "should update product" do
45   #patch :update, id: @product, product: { description: @product.description, image_url: @product.image_url, price: @produ
46   patch :update, id: @product, product: @update
47   assert_redirected_to product_path(assigns(:product))
48 end

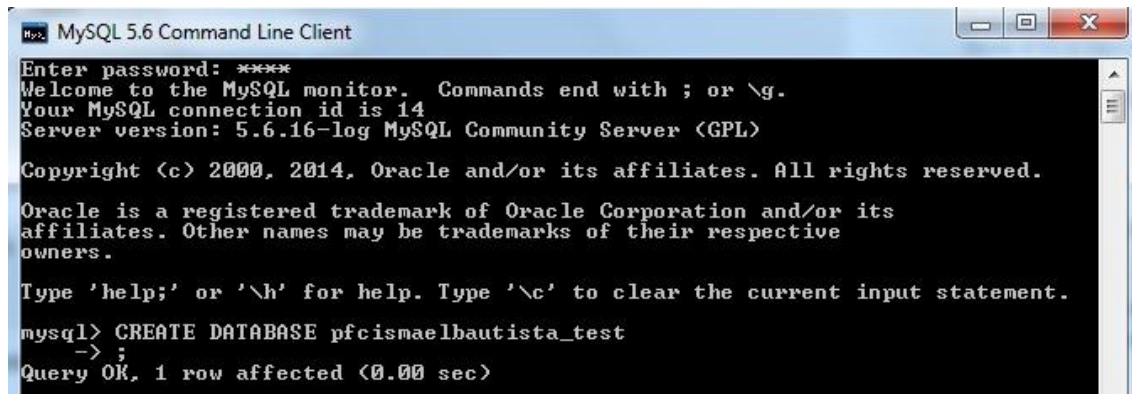
```

Figura 23. Tests en el código.



Para integrar el entorno test a nuestra base de datos pfcismaelbautista_test, se han tenido que hacer una serie de modificaciones en SQL, creando primeramente la base de datos y luego otorgándole permisos. Se detalla a continuación:

1 – Debemos loguearnos como root en nuestro shell de MySQL (o en el que queramos) y crear la base de datos pfcismaelbautista_test .



```
MySQL 5.6 Command Line Client
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.6.16-log MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.


Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE pfcismaelbautista_test
-> ;
Query OK, 1 row affected (0.00 sec)
```

Figura 24. MySQL1

2 – Debemos crear el usuario elegido para la base de datos y asignarle una contraseña. Acto seguido, usamos la base de datos.




```
mysql> CREATE USER 'test_pfc'@'localhost' IDENTIFIED BY 'mypass123';
Query OK, 0 rows affected (0.00 sec)

mysql> USE pfcismaelbautista_test
Database changed
```

Figura 25. MySQL2

3 – Damos todos los privilegios a la base de datos, cambiando la contraseña a la que contraseña que vamos a pondremos en el archivo database.yml.



```
mysql> GRANT ALL privileges ON pfcismaelbautista_test.* to test_pfc@localhost IDENTIFIED BY 'mypass1234';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH privileges;
Query OK, 0 rows affected (0.00 sec)
```

Figura 26. MySQL3

4 – Actualizamos el fichero database.yml, poniendo el usuario y contraseña asignados a la base de datos en el paso anterior.

```
20 # Warning: The database defined as "test" will be erased and
21 # re-generated from your development database when you run "rake".
22 # Do not set this db to the same as development or production.
23 test:
24   adapter: mysql2
25   encoding: utf8
26   database: pfcIsmaelBautista_test
27   pool: 5
28   username: test_pfc
29   password: mypass1234
30   host: localhost
```

Figura 27. Configuración database.yml.

8.9 Mailer

Esta aplicación ha sido programada con la capacidad de enviar un e-mail al comprador, cada vez que realiza un pedido, indicándole en él el contenido del pedido solicitado.

La primera configuración se realiza modificando los archivos development.rb y ordernotifier.rb.

```
29
30 config.action_mailer.delivery_method = :smtp
31 config.action_mailer.smtp_settings = {
32   address: "smtp.gmail.com",
33   port: 587,
34   #domain: "domain.of.sender.net",
35   authentication: "plain",
36   user_name: "cuentapfcismaelbautista@gmail.com",
37   password: "pfcismaelbautista",
38   enable_starttls_auto: true
39 }
```

Figura 28. Development.rb

En development.rb configuramos los datos de servidor smtp, cuenta, puerto, contraseña y demás.

```
1 class OrderNotifier < ActionMailer::Base
2   default from: "cuentapfcismaelbautista@gmail.com"
3
4   # Subject can be set in your I18n file at config/locales/en.yml
5   # with the following lookup:
6   #
7   #   en.order_notifier.received.subject
8   #
9   def received(order)
10    @order = order
11
12    mail to: order.email, subject: 'Confirmación de su compra'
13  end
```

Figura 29. Ordernotifier.rb

En ordernotifier.rb definimos la dirección de envío por defecto, y definimos los métodos de envío.

Una vez configurados, pasaremos a crear el código para realizar los envíos. Rails tiene una clase llamada mailer, ubicada en la ruta app/mailers. Utilizando el siguiente comando en nuestro bash, creamos todo el esqueleto de nuestro mailer: en este caso hemos creado uno para cuándo recibamos el pedido y otro para cuándo sea enviado, aunque se ha dejado en funcionamiento solo el primero de ellos.

rails generate mailer OrderNotifier received shipped

```
isma@ISMA-PC ~/PFC ISMA FINAL/pfcIsmaelBautista (master)
$ rails g mailer OrderNotifier received shipped
DL is deprecated, please use Fiddle
create  app/mailers/order_notifier.rb
invoke  erb
create  app/views/order_notifier
create  app/views/order_notifier/received.text.erb
create  app/views/order_notifier/shipped.text.erb
invoke  test_unit
create  test/mailers/order_notifier_test.rb
```

Figura 29. Mailer

8.10 Ajax y jQuery

En nuestra aplicación, se ha utilizado Ajax para ir actualizando el carrito de la barra lateral sin tener que cargar y visualizar toda la página de nuevo. Es decir, queremos que los botones Añadir al carrito (Add to cart) llamen a la acción `add_to_cart` en segundo plano (`background`). Es bastante importante que cuando se vaya a trabajar con Ajax, primeramente se trabaje con la aplicación sin nada de Ajax, e ir introduciéndolo poco a poco de una manera gradual.

```
<%= button_to 'Add to Cart', line_items_path(product_id: product),  
remote: true %>
```

Finalmente, añadiendo un poco de JavaScript, ya tenemos nuestro pequeño aporte de Ajax funcionando.

```
$('#cart').html("<%= escape_javascript render(@cart) %>");
```

En esta aplicación se ha usado también un poco de jQuery, librería que ya viene por defecto en Rails. Por ejemplo, se ha utilizado para no mostrar que hay un carrito en la parte izquierda del layout cuando este está vacío.

```
if ($('#cart tr').length == 1) { $('#cart').show('blind', 1000); }
```

8.11 Logging in y Bcrypt

Para crear usuarios admin, lo primero que tenemos que hacer es crear la tabla correspondiente en la base de datos:

```
rails generate scaffold User name:string password:digest  
rake db:migrate
```



Después de realizar varias comprobaciones en el modelo, como nombres únicos y demás, debemos utilizar una gema que viene por defecto en nuestro Gemfile: Bcrypt-ruby e instalarla:

```
bundle install
```

Una vez realizado esto, debemos generar un controlador para las sesiones (Sessions) y otro para crear un index para el usuario admin.

```
rails generate controller Sessions new create destroy
```

```
rails generate controller Admin index
```

Acabado esto, y jugando con las sesiones para cuándo hagamos log in y log out en el controlador de sesiones (sessions_controller.rb), deberemos restringir el acceso a usuarios que no sean administrador a las vistas del mismo. Esto lo conseguimos definiendo un método en el controlador de la aplicación, y utilizándolo, tal y como se muestra en la imagen inferior:

```
6   before_action :authorize
7
8   protected
9
10  def authorize
11    unless User.find_by(id: session[:user_id])
12      redirect_to login_url, notice: "Please log in"
13    end
14  end
```

Figura 30. Sessions_controller.rb . Restricciones.

Con esto lo que hemos realizado es una prohibición de acceso a toda la aplicación a todos los usuarios que no sean administrador. Para evitar esto, y permitir que nuestros usuarios puedan comprar, o navegar por las vistas que nosotros creamos convenientes, debemos autorizar en el resto de controladores los permisos de acceso para que se pueda acceder sin ser administrador (en los que se crea conveniente que se debe hacer). Esto se realiza de la siguiente manera.

```
1 class MainController < ApplicationController
2   skip_before_action :authorize
3 end
```

Figura 31. Ejemplo de autorización en controlador.

Una vez realizado todo esto, podemos crear nuestro menú de administrador usando la sesión del mismo, es decir, comprobando si se ha logueado un admin y en ese caso, mostrando dicho menú,

```
49 <!-- Si estamos logueados, opciones de administrador y deslogueo -->
50 <% if session[:user_id] %>
51   <ul>
52     <li><%= link_to 'Pedidos', orders_path %></li>
53     <li><%= link_to 'Productos', products_path %></li>
54     <li><%= link_to 'Usuarios Admin', users_path %></li>
55   </ul>
56   <%= button_to 'Logout', logout_path, method: :delete %>
57 <% end %>
```

Figura 32. Menu administrador.

En este menú se han definido las siguientes opciones:

- Crear, modificar o eliminar productos nuevos para la venta
- Ver los pedidos pendientes y eliminarlos en caso de ya estar terminados.
- Crear nuevos usuarios admin.

8.12 Internacionalización (i18N)

La internacionalización (i18n) consiste en dotar a una aplicación web de las características necesarias para que pueda mostrarse en diferentes idiomas. En este caso, la aplicación realizada ha sido programada para estar en dos idiomas y poder elegir entre ellos según lo necesitemos: español e inglés.

En primer lugar, debemos crear un nuevo fichero de configuración en el que declararemos qué idiomas vamos a utilizar, y cuál vamos a usar como idioma por defecto. Lo situaremos en la ruta config/initializers/i18n.rb.

```
i18n.rb
1 I18n.default_locale = :en
2
3 LANGUAGES = [
4   ['English', 'en'],
5   ["Español", 'es']
6 ]
```

Figura 33. I18n.rb

Acto seguido, modificaremos el fichero routes.rb; y el controlador de la aplicación de la siguiente manera, programando un manejador para que se ponga el idioma adecuado según el parámetro que se le pase. Así tendremos configurado nuestro entorno:

```

18   scope '(:locale)' do
19     resources :orders
20
21     resources :line_items
22
23     resources :carts
24
25     post "contact_us/index"
26
27     root 'main#index', as: 'main', via: :all
28
29   end

```

Figura 34. Routes.rb.

```

4   before_action :set_internacionalization_from_params

16  def set_internacionalization_from_params
17    if params[:locale]
18      if I18n.available_locales.map(&:to_s).include?(params[:locale])
19        I18n.locale = params[:locale]
20      else
21        flash.now[:notice] =
22          "#{params[:locale]} translation not available"
23        logger.error flash.now[:notice]
24      end
25    end
26  end
27
28  def default_url_options
29    { locale: I18n.locale }
30  end

```

Figuras 35 y 36. Application_controller.rb.

Una vez realizado esto, lo que debemos hacer es, en las vistas dónde queramos usar la internacionalización, sustituir el texto escrito por el siguiente código: t('blabla'), dónde “t” indica que vamos a usar la internacionalización y “blabla” es el nombre puesto al encapsulamiento del texto, el cual estará situado en sus consecuentes ficheros de idiomas (es.yml y en.yml, situados en la ruta config/locales) con la traducción para cada uno de ellos. Veamos un ejemplo.

```
22 <%= @page_title || t('.title') %>
```

Figura 37. Layout. Internacionalización.

En la imagen superior, correspondiente a la vista del layout de esta aplicación, hemos hecho referencia a la internacionalización con el campo 'title'. Una vez hecho esto, en los ficheros de idiomas, colocamos el texto que queremos que salga cuándo se referencia a este campo.

```
23 en:
24   layouts:
25     application:
26       title: "Your books, at your university"
```

```
2 es:
3   layouts:
4     application:
5       title: "Tus libros, en tu universidad"
```

Figuras 38 y 39. Ficheros de idiomas.

Finalmente, para poder elegir entre idiomas sin tener que cambiar la URL en la barra del explorador, se ha programado un switcher que nos permita elegir entre ambos idiomas. En primer lugar, en el layout de la página situaremos el switcher y su código para poder elegirlo. Añadimos un poco de javascript para que no haga falta hacer Submit para realizar el cambio de idioma.

```
14 <!-- I18N. SWITCHER PARA ELEGIR IDIOMA -->
15 <%= form_tag main_url, class: 'locale' do %>
16   <%= select_tag 'set_locale', options_for_select(LANGUAGES, I18n.locale.to_s), onchange: 'this.form.submit()' %>
17   <%= submit_tag 'submit' %>
18   <%= javascript_tag "$('.locale input').hide()" %>
19 <% end %>
```

Figura 40. Switcher para cambiar de idioma la aplicación.

Después, en nuestro controlador de main, que es la página la que nos redirigirá cuándo se realice el cambio de idioma, se incluye el siguiente código, el cuál, como he dicho antes, nos redirigirá a la vista main si se ha utilizado el cambio de idioma, realizando el susodicho cambio.

```
4   def index
5     if params[:set_locale]
6       redirect_to main_url(locale: params[:set_locale])
7     end
8   end
```

Figura 41. Main_controller.rb

9. Referencias

- Ruby on Rails: Desarrollo práctico de aplicaciones web. Autor: Santiago Ponce Moreno.
- Página Stackoverflow: <http://stackoverflow.com>
- Página oficial RoR: <http://rubyonrails.org>
- Github: <http://github.com>
- W3Schools : <http://w3schools.com>

