UNIVERSIDAD DE OVIEDO

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA EN INFORMÁTICA DE OVIEDO

TRABAJO FIN DE MÁSTER INGENIERÍA WEB

"Portal Web para promoción y gestión de hotel rural"

DIRECTOR: Lourdes Tajes Martínez

AUTOR: Luis Amor Álvarez

Vº Bº del Director del Proyecto

Agradecimientos

Me gustaría aprovechar esta sección para agradecer a mis padres y a Mireia Ruiz su apoyo durante el desarrollo del máster y la ayuda para compatibilizar trabajo y Trabajo Fin de Máster.

Palabras Clave

Casa rural, hotel, Ruby on Rails, Diseño adaptativo, Gestión de reservas.

Keywords

Rural turism house, hotel, Ruby on Rails, Response design, booking management.

Índice General

Índice de Figuras

Capítulo 1. Memoria del Proyecto

1.1 Resumen

El proyecto consiste en un desarrollo web a medida para la promoción y gestión de un hotel pequeño o casa rural de alquiler por habitaciones. Existe un **cliente real** al cual está dirigido el proyecto: *"Casa rural Sobrefuentes".*

La aplicación está concebida como el principal punto de contacto entre el cliente y el propietario del establecimiento. Debe satisfacer toda la información que pueda interesar al cliente: tipo de negocio, ubicación, disponibilidad, precio, contacto así como ofrecer una vía rápida y sencilla para la **conversión** de visitantes en clientes. En nuestro caso el objetivo es la reserva de habitaciones. Como punto de encuentro entre cliente y propietario, se pone especial atención a todos los aspectos que puedan influir en el **SEO** de la web.

Además de **herramienta informativa**, el proyecto sirve para la **gestión** del alquiler de las habitaciones mediante la administración de: precios, número y tipo de habitaciones, clientes, reservas, disponibilidad, facturas. También es posible administrar la galería fotográfica de la web.

La web consta de una **parte pública** para los clientes y una **parte privada** para la administración a la cual se accede tras un proceso de autenticación. Tanto el administrador de la aplicación como los clientes pueden acceder a la parte privada con su cuenta, pero tendrán acceso a opciones diferentes en función de su rol (administrador o cliente).

El lenguaje de programación escogido para el proyecto ha sido Ruby, utilizando el framework para desarrollo de aplicaciones web **Ruby on Rails**. Para el interfaz o *frontend* se ha utilizado HTML5, CSS3, JavaScript (librerías jQuery, AJAX y JSON) y **response design** en la parte pública con el objetivo de ofrecer la mejor experiencia de usuario posible al cliente.

1.2 Motivación, Objetivos y Alcance del Proyecto

La motivación por la cual se decidió proponer y realizar este proyecto, es buscar una solución software para la gestión de reservas en una Casa Rural de alquiler en régimen de habitaciones llamada Sobrefuentes. Dado que es un negocio familiar los problemas y necesidades en la gestión de un establecimiento de estas características son bien conocidos por lo tanto se pretende hacer un proyecto al cual pueda darle un uso real en un negocio existente.

También se pretende aplicar los conocimientos en desarrollo de aplicaciones web adquiridos por el alumno en el Máster de Postgrado en Ingeniería Web de la Universidad de Oviedo.

Se escogió el *framework* de desarrollo Ruby on Rails ya que es un framework ágil, para desarrollo de aplicaciones web, de código abierto y trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis muy legible.

Los **objetivos** son: por una parte renovar la obsoleta web del establecimiento (tanto en contenidos como tecnología) y por otra proporcionar un sistema de reservas que permitan al cliente solicitar una reserva a través de la web sin necesidad de utilizar las vías clásicas de contacto como el teléfono o el correo electrónico. Con esto se pretende mejorar la conversión de visitantes en clientes ofreciendo más alternativas para formalizar la reserva.

Además el tener un sistema automatizado donde poder administrar las reservas facilita la tarea de gestión de estas, así como la gestión de facturas asociadas a reservas y usuarios, integrados en el sistema.

Lógicamente para poder ofrecer una información detallada de la disponibilidad del establecimiento al cliente, el administrador debe disponer de un panel de administración donde administrar los precios, el número y tipo de habitaciones, los usuarios y las reservas.

También es posible modificar la **galería fotográfica** del sitio web a gusto del administrador. Se descartó introducir un sistema de gestión de contenidos más profundo debido a que tanto las secciones como los textos son muy poco dinámicos. Las descripciones del establecimiento apenas varían con el paso del tiempo por lo tanto se decidió que no merece la pena invertir tiempo en un desarrollo más profundo para tener un gestor de contenidos.

Una vez que el cliente formaliza una reserva, **no se solicita ninguna transacción monetaria**. El 100% del pago se realiza en el día de salida o *checkout* en el mismo establecimiento. En un principio se sopesó la posibilidad de solicitar un porcentaje del precio total de la estancia, ya sea por transferencia bancaria o a través de una pasarela de pago aceptando el pago con tarjeta de crédito o PayPal. Está opción fue descartada por el cliente por dos motivos principales.

Por una parte los **costes** de habilitar una pasarela de pago eran altos para un negocio con una baja facturación anual. A esto se sumarían las comisiones por transacción en muchos casos superiores al 3%. Por otra parte, cuando se produce una cancelación, la política de devoluciones dicta que se devuelve el 100% del importe, lo cual provoca trabajo extra al dueño del establecimiento, y costes económicos si hay comisiones de por medio.

Una posible modificación de la política de devoluciones no está descartada, por lo tanto la implantación de una pasarela de pago, partiendo del sistema de reservas actual está recogida en la sección de ampliaciones, pero queda fuera del alcance del actual proyecto.

1.3 Resumen de todos los aspectos

En primer lugar se realizará un resumen general de todos los aspectos técnicos destacados del proyecto, en el cual se mencionará, a grandes rasgos y sin profundizar demasiado en estos, los aspectos técnicos y las tecnologías utilizadas para el desarrollo del proyecto. Después se realizará un resumen de cada uno de los aspectos más importantes del proyecto: planificación y presupuesto, análisis, diseño y pruebas.

1.3.1 Introducción

Se ha desarrollado un proyecto web a medida orientado a un cliente real.

Para la parte pública de la web se ha llevado a cabo un desarrollo utilizando HTML5, CSS3, JavaScript. Esta parte cuenta con diseño adaptativo de las páginas, es decir, el contenido se adapta al tamaño de la ventana del navegador. Como base para el desarrollo adaptativo se ha utilizado la librería CSS *Skeleton*.

Los usuarios pueden registrarse y autenticarse en la aplicación. Una vez logueados pueden acceder a un panel de control cuyas opciones varían en función del rol del usuario (cliente o administrador).

El administrador puede gestionar las reservas, precios, disponibilidad, clientes, facturas y fotografías de la web desde el panel de administración. El cliente puede acceder a su cuenta, modificar sus datos personales y visualizar las reservas pendientes o finalizadas.

El desarrollo de la lógica de la aplicación tiene asociado pruebas unitarias para cada módulo y funcionalidad, haciendo un desarrollo orientado a test, conocido como TDD *(test driven development)*.

Se han llevado a cabo técnicas para mejorar el SEO de la web.

1.3.2 Planificación y presupuesto

La duración del proyecto se ha estimado inicialmente en unos 169 días en total (excluyendo días no laborables), tomando como base una jornada laboral de 2 horas diarias. El presupuesto final ha quedado estimado en unos 15.104,43 € para el desarrollo de esta aplicación. Se adjunta diagrama GANTT.

1.3.3 Análisis

En este apartado comenzaremos por la determinación del alcance del sistema, a partir de la cual obtendremos la especificación de requisitos. La aplicación debe:

- Mostrar información del hotel a posibles visitantes (descripción, fotografías, disponibilidad, localización) así como ofrecer formas de contacto con el establecimiento (formulario de contacto, teléfono).
- Permitir al administrador la administración del tipo y número de habitaciones.
- Permitir al administrador realizar reservas en el hotel en nombre de clientes.
- Permitir editar tanto los precios que se muestran en la web como los intervalos de tiempo en los que aplican estos.
- Permitir guardar, editar y visualizar información sobre las reservas de la casa rural.
- Generar facturas asociadas a reservas.
- Permitir al administrador modificar las fotografías de la galería fotográfica.
- Permitir visualizar la ocupación de la casa a los clientes, aunque sea de forma orientativa.

A partir de estos requisitos se elaboran los casos de uso y los escenarios, que nos servirán para definir una primera aproximación del sistema que vamos desarrollar, a partir de la cual realizaremos el diseño.

1.3.4 Diseño

En este apartado definiremos cómo vamos a desarrollar el sistema, cómo va a ser su arquitectura y qué componentes formarán el sistema. La información de este apartado la utilizaremos a la hora de implementar la aplicación.

1.3.5 Implementación

Se utilizara el framework Ruby on Rails, la última versión de Rails 3 ya que incluye mejoras respecto a la versión 2 de Rails. Además el abanico de gemas (librerías para Ruby) es más amplio para la versión 3.

1.3.6 Pruebas

En este apartado realizamos las pruebas funcionales al sistema para comprobar el correcto funcionamiento del mismo teniendo en cuenta los requisitos y escenarios que definimos inicialmente. Utilizaremos la librería **Rspec** para los test unitarios. Además se realizarán algunas pruebas de usabilidad. El objetivo es intentar, en la medida de lo posible, comprobar y mejorar la facilidad de uso de nuestro proyecto.

1.3.7 Ampliaciones

Hay más funcionalidades que serían útiles para la gestión del establecimiento como: implementar una pasarela de pago, exportar las reservas a un archivo, importar reservas desde un buzón de correo, un módulo social con opiniones de los clientes. Debido a la necesidad de acotar el tiempo de desarrollo del proyecto se han quedado fuera de este desarrollo, no descartando su futura implementación y agregación al actual proyecto. Las ampliaciones están detalladas en el apartado 10.2.

Capítulo 2. Introducción

2.1 Justificación del Proyecto

El turismo rural es un sector en auge en la actualidad, para este tipo de negocios pequeños es difícil hacerse visibles al potencial cliente.

Este proyecto está dirigido a de la casa rural Sobrefuentes, es una casa rural con 6 habitaciones que se alquila por noches en régimen de habitaciones independientes.

Este negocio disponía ya de una página web estática en PHP. La web fue creada en el año 2005, coincidiendo con la inauguración del establecimiento. Lo que se plantea es una **renovación total y mejora** sobre lo que ofrecía el sitio web obsoleto. Los propietarios querían poder ofrecer a los visitantes la reserva de habitaciones directamente a través del sitio web. Esto implicaba lógicamente la posibilidad de administrar las habitaciones, la ocupación y los precios.

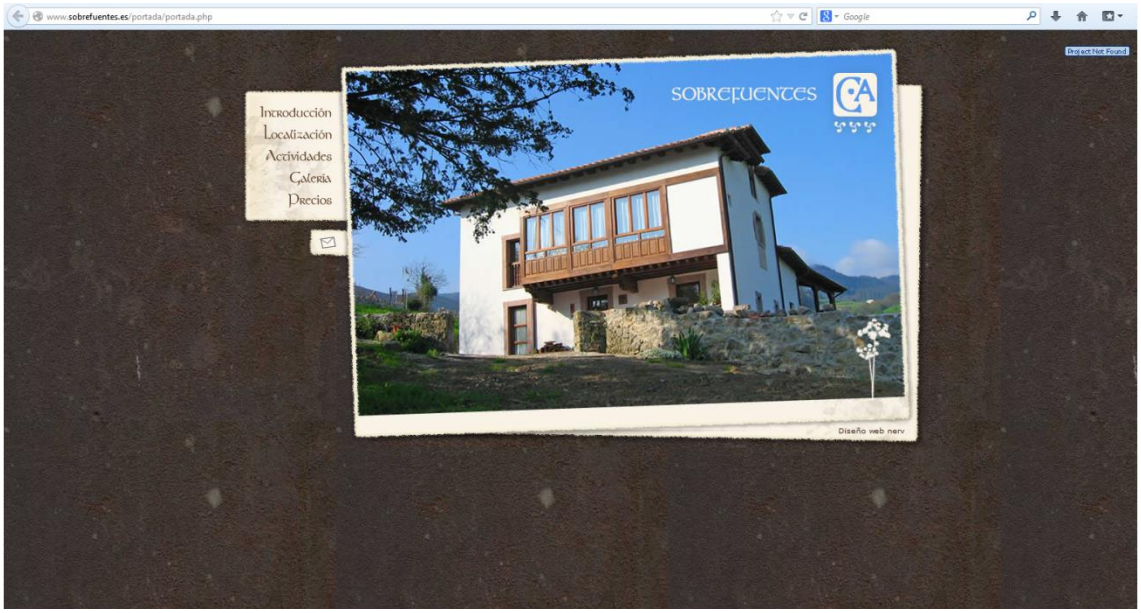


Figura 1.1. Página Web antigua a sustituir

La página Web de nuestra empresa es un tema de crucial importancia estratégica para nuestro negocio. Considerando que la mayoría de los potenciales clientes utilizan Internet como herramienta fundamental de su proceso de búsqueda de alojamiento, nuestra presencia y visibilidad en Internet se vuelve esencial. Necesitamos que la primera toma de contacto entre el cliente y el establecimiento transmita una imagen profesional, con información clara y actualizada de nuestro establecimiento y servicios. Que nos permita convertir un visitante en cliente y comunicarnos con nuestros clientes actuales y potenciales

La página web y los portales de turismo a los cuales esté suscrito el negocio (los cuales muchas veces hacen referencia a la web del establecimiento) son las principales fuentes de

captación de clientes, por ello debe ponerse especial cuidado al contenido y formato de la web.

En este caso concreto, los propietarios se encontraron con que las herramientas software para gestionar hoteles eran complejas de utilizar, demasiado amplias, más aun cuando algunas características de un hotel no son aplicables a un negocio pequeño. Por tanto se optó por crear una herramienta personalizada para este negocio.

También demandaban la compatibilidad con diferentes dispositivos: móviles, tabletas y PC, por lo que se optó por aplicar un diseño adaptativo.

2.2 Objetivos del Proyecto

Como se ha comentado anteriormente este proyecto pretende una renovación total del sitio web, tanto en contenido como forma. Además pretende integrar en el sitio web un mecanismo para gestionar las habitaciones, precios, disponibilidad, clientes, facturas y galería fotográfica.

Además de las funcionalidades mencionadas, se busca optimizar la visualización de los contenidos en diferentes dispositivos mediante *response design*. El *responsive design* o diseño web adaptativo, consiste en la adaptación de la web al medio en el que se muestra. De esta manera, el mismo sitio web podrá visualizarse de manera óptima desde cualquier dispositivo, independientemente de que sea móvil, tableta u ordenador de sobremesa.

A continuación se enumeran los objetivos relacionados con la experiencia de usuario para el cliente a cumplir por el proyecto:

- Renovar **diseño** y **contenidos** de la web para ofrecer al visitante un aspecto más moderno y profesional.
- Optimizar la navegabilidad por el sitio web para **diferentes dispositivos**.
- Ofrecer al visitante información visual amigable de la **disponibilidad** de habitaciones del establecimiento.
- Ofrecer al visitante un mecanismo para **formalizar la reserva** de habitaciones sin necesidad de entablar una comunicación directa con el propietario del establecimiento.
- Ofrecer al cliente (visitante que ya ha reservado habitaciones o que se ha alojado previamente en el hotel) **información de su reserva**, así como la posibilidad de modificación de sus datos personales.
- El sitio web debe soportar varios **idiomas**, al menos para castellano e inglés, con la posibilidad de agregar más idiomas posteriormente.

Otros objetivos que debe satisfacer el proyecto, estos destinados a la gestión del establecimiento por parte del administrador son:

- Gestión de **reservas**: el administrador debe poder crear, modificar y visualizar reservas. Las reservas están asociadas a un cliente, y en algunos casos a un *partner* o socio encargado de enviar clientes.
- Modificar el número de **habitaciones**.
- Las habitaciones se agrupan por el **tipo** de estas, debe ser posible modificar el tipo de las habitaciones y crear nuevos tipos.
- Modificar las **tarifas**. Los precios se gestionan mediante intervalos de tiempo. Es posible crear nuevos intervalos, y modificar el precio de estos.
- Modificar **galería fotográfica** desde el panel de administración.
- Administración de empresas asociadas o cadenas hoteleras que proporcionen clientes a cambio de comisiones o pago de cuota anual.
- Creación automática de **facturas** asociadas a reservas.

Además la web **debe comunicar vía correo electrónico** tanto al cliente como al administrador las acciones relevantes que se lleven a cabo en el establecimiento: alta nueva, reserva nueva, etc.

Otro objetivo a tener muy en cuenta es el posicionamiento **SEO** *(Search Engine Optimization)* de la web. De poco nos servirá una buena web si nadie accede a ella.

2.3 Estudio de la Situación Actual

El turismo rural es un sector en auge, existe multitud de software de escritorio para gestionar reservas de hoteles, así como posibles gestores de contenido para la gestión del sitio web, pero lo que se propone en este proyecto es integrar en el sitio web la gestión de reservas del establecimiento de una manera personalizada a las necesidades del negocio.

Algunas alternativas podrían ser:

- Desarrollar el sitio web con un gestor de contenidos que permita:
  - Modificación de galería fotográfica.
  - Incluir un módulo para la gestión de reservas hoteleras.
- Desarrollar la web, sin módulo de reservas y delegar estas a una herramienta gestionada por terceros a través de un *iframe* o redirección web.

2.3.1 Evaluación de Alternativas

Otras opciones podrían haber sido implantar algún gestor de contenidos en PHP, Java o .net que tenga algún módulo para gestión de reservas o delegar estas a un tercero. En esta sección mostrare una lista de posibles alternativas tanto módulos de reservas completos como reservas en nuestras web gestionadas por terceros. . Existen muchos módulos de este tipo para gestores de contenidos de código abierto populares como pueden ser WordPress o *Joomla*, sólo mencionare los más relevantes.

2.3.1.1 WordPress Plugin for making reservations

Plugin para el framework PHP que permite la gestión de reservas integrándose con un sitio web desarrollado en *WordPress*:

WordPress es un sistema de gestión de contenido enfocado a la creación de blogs (sitios web periódicamente actualizados). Desarrollado en PHP y MySQL, bajo licencia GPL y código modificable.

Este *plugin* de WordPress permite hacer reservas online desde su sitio web. Los visitantes del sitio podrán comprobar la disponibilidad de apartamentos, casas, habitaciones de hotel o servicios que la web ofrece. También pueden hacer reservas con la posibilidad de elegir entre varios días, de un solo día o por la reserva de hora. Sus clientes pueden incluso ver y registrar las próximas reservas.

Las licencias son diferentes en función de las necesidades que se necesiten en el establecimiento:

- Como **ventajas** se puede destacar la facilidad de integración y la potencia de la herramienta en las licencias más avanzadas.

- Como **desventajas** destacaría que es de pago, donde la cuota anual en las licencias interesantes empieza a ser un poco cara. Además la difícil personalización de la parte visual de la herramienta e integración con nuestra web si queremos salirnos del diseño predefinido.

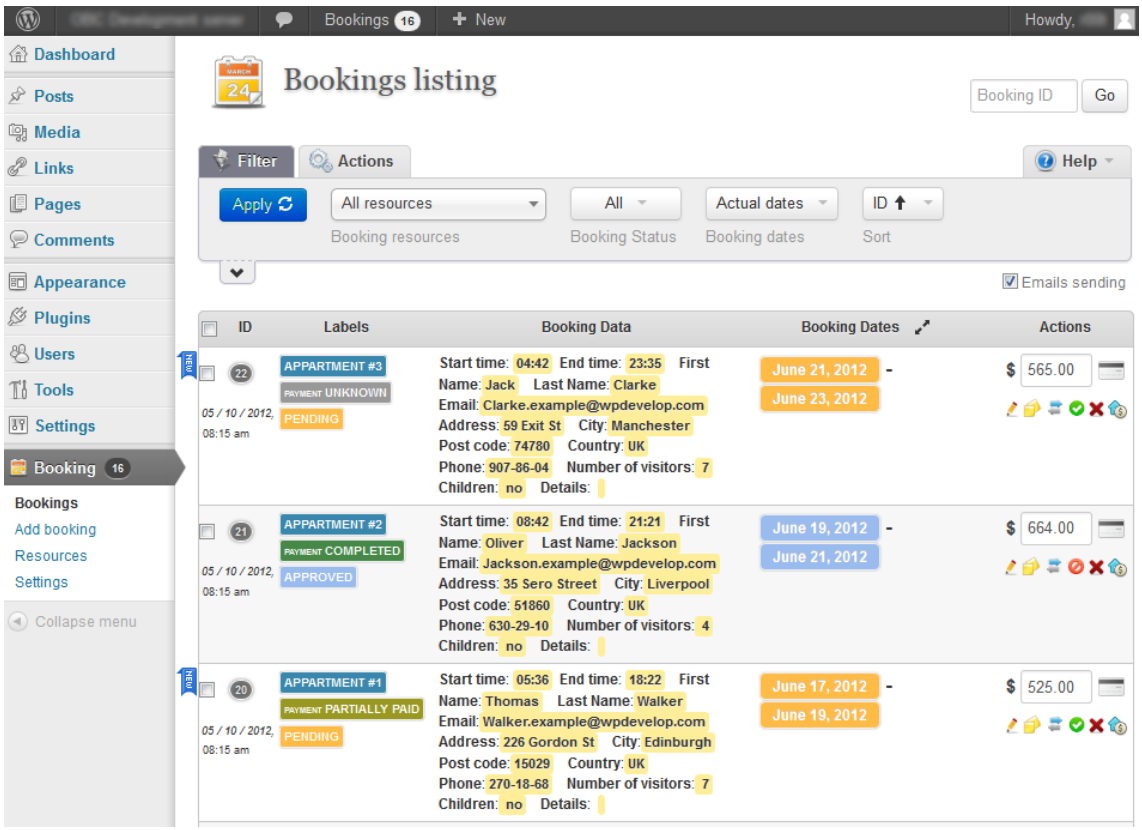


Figura2.2 Captura de pantalla de WordPress Plugin for making reservations

2.3.1.2 Jomres

Jomres es un potente componente para Joomla! que permite la administración de hoteles desde una página web implementada con el gestor de contenidos Joomla.

Joomla! es un sistema de gestión de contenidos, y entre sus principales virtudes está la de permitir editar el contenido de un sitio web de manera sencilla. Es una aplicación de código abierto programada mayoritariamente en PHP bajo una licencia GPL. Este administrador de contenidos puede trabajar en Internet o intranets y requiere de una base de datos MySQL, así como, preferiblemente, de un servidor HTTP Apache

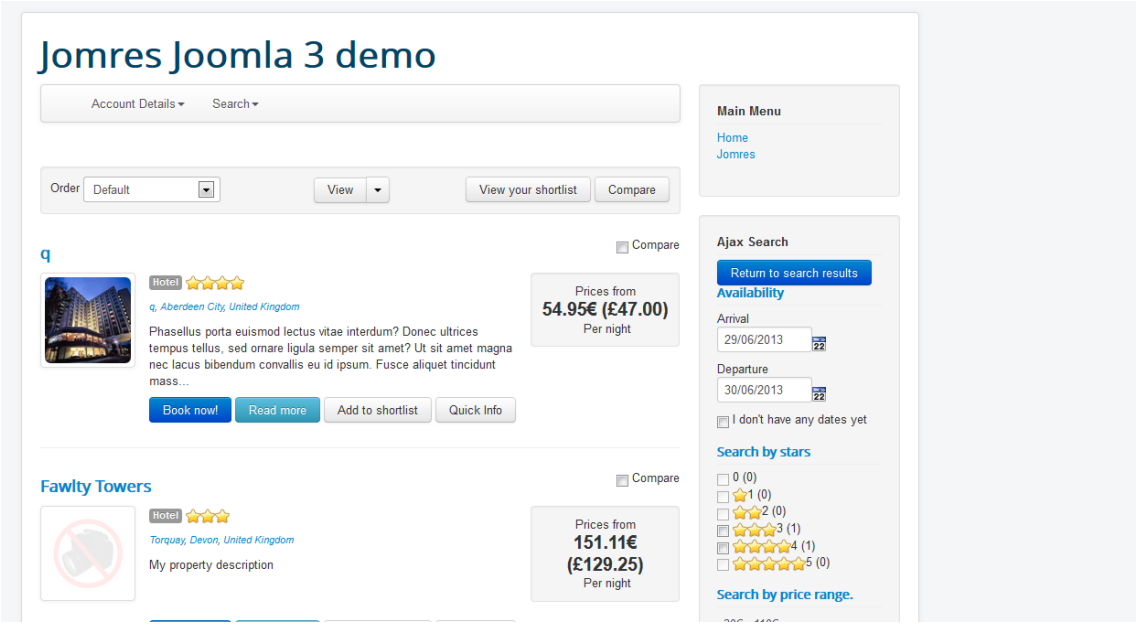


Figura2.2 Captura de pantalla de Jomres

- Como **ventajas** destacaría la posibilidad de gestionar diferentes hoteles, y la cantidad de opciones de configuración que ofrece. Es de pago, pero la licencia es relativamente barata.
- Como **desventajas**, complejidad de instalación y configuración, exceso de posibilidades para un negocio pequeño.

2.3.1.3 Reservas a través de Booking.com

El motor de reservas Booking.com permite a sus hoteles afiliados, integrar en su página web un formulario a través del cual, el visitante reserva directamente contra el gestor de Booking.com. Ofrece distintos tipos de integraciones:

- El **link** de Booking.com en HTML: es la manera más sencilla y rápida de integrar Booking.com. Nuestro sitio web se abrirá en una ventana nueva con el mismo aspecto de siempre. Hacemos un seguimiento del número de reservas a través del link a Booking.com.
- **Integración** de marca blanca: la integración de marca blanca muestra el nombre de tu establecimiento. Booking.com opera en tu sitio web de forma invisible. El motor de reservas de marca blanca se puede cargar en un marco (*iframe*) de tu sitio web o abrirse en una nueva ventana. Ofrece la posibilidad de editar la hoja de estilo para que se integre a la perfección con el diseño de tu sitio web. Además, podemos introducir códigos HTML en el encabezado y el pie para completar la integración. La integración de marca blanca es fácil de implementar y mantener. Como las páginas se cargan directamente desde los servidores de Booking.com, las tarifas y disponibilidad se gestiona desde el panel de administración de Booking.com

  - Como **ventajas** destacaría la facilidad de integración, y la posibilidad de sincronizar la gestión de habitaciones con el motor de reservas de Booking.com.
  - Como **desventajas**: la baja personalización pese a que permita editar una hoja de estilos (el código no podrías tocarlo) y la más importante, Booking se lleva una **comisión** del 10% sobre el precio de la reserva, cuando la conversión de visitante en cliente es mérito de la web del establecimiento, no del portal de Booking.

Capítulo 3. Aspectos Teóricos

3.1 Conceptos

3.1.1 Framework

Un *framework* es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Es una estructura conceptual y tecnológica de soporte definido, normalmente con módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

3.1.2 MVC (Modelo Vista Controlador)

El proyecto sigue el paradigma de arquitectura Modelo Vista Controlador (**MVC**), es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

El patrón de llamada y retorno MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

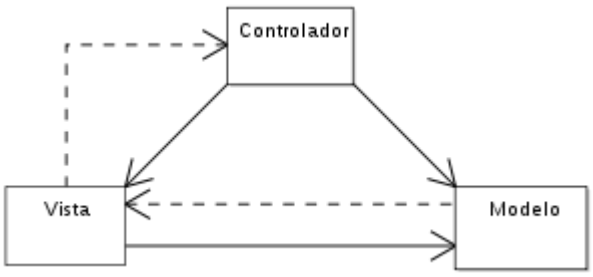


Figura 3.1. Diagrama MVC

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador**: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

3.1.3 DRY (Don't Repeat Yourself)

El principio No te repitas (en inglés *Don't Repeat Yourself* o DRY, también conocido como Una vez y sólo una) es una filosofía de definición de procesos que promueve la reducción de la duplicación especialmente en computación.

Según este principio toda pieza de información nunca debería ser duplicada debido a que la duplicación incrementa la dificultad en los cambios y evolución posterior, puede perjudicar la claridad y crear un espacio para posibles inconsistencias. Por "pieza de información" podemos entender, en un sentido amplio, desde datos almacenados en una base de datos pasando por el código fuente de un programa de software hasta llegar a información textual o documentación.

Cuando el principio DRY se aplica de forma eficiente los cambios en cualquier parte del proceso requieren cambios en un único lugar. Por el contrario, si algunas partes del proceso están repetidas por varios sitios, los cambios pueden provocar fallos con mayor facilidad si todos los sitios en los que aparece no se encuentran sincronizados.

3.1.4 TDD (Test Driven Development)

Los modelos del proyecto se han desarrollado siguiendo la práctica TDD. Desarrollo guiado por pruebas de software, o Test driven development (TDD) es una práctica de programación que involucra otras dos prácticas: Escribir las pruebas primero y Refactorización. Para escribir las pruebas generalmente se utilizan las pruebas unitarias (librería Rspec en este proyecto).

En primer lugar, se escribe una prueba y se verifica que las pruebas fallan. A continuación, se implementa el código que hace que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione. La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido.

3.1.5 SEO (Search Engine Optimization)

El posicionamiento en buscadores u optimización de motores de búsqueda es el proceso de mejorar la visibilidad de un sitio web en los resultados orgánicos de los diferentes buscadores.

Es frecuente nombrarlo por su título inglés, SEO, también es común llamarlo posicionamiento web, aunque este término no resulta tan preciso, ya que engloba otras fuentes de tráfico fuera de los motores de búsqueda. En los últimos años, la generalización de las estrategias de posicionamiento en buscadores y su implementación en un mayor número

de sitios web, han logrado generar la consciencia que ocupar los primeros puestos en las páginas de resultados puede ser crucial para un sitio.

3.1.6 Response Design

El **responsive design o diseño web adaptativo,** consiste en la adaptación de la web al medio en el que se muestra. De esta manera, el mismo sitio web podrá visualizarse de manera óptima desde cualquier dispositivo, independientemente de que sea móvil, *tablet* u ordenador de sobremesa.

Según el libro blanco de minube.com "Más del 50% de los usuarios de móviles en España dispone de un *smartphone*, lo que nos convierte en el **primer país por penetración en Europa**, y esa es una ventaja que debemos aprovechar".

Los establecimientos turísticos tienen que evolucionar a la vez que lo hacen sus clientes y la tecnología que estos utilizan. El *responsive design* de la web permitirá asegurar que el establecimiento sea accesible a los clientes a través de cualquier tipo de dispositivos.

Si tenemos en cuenta que en este año se espera que las búsquedas relacionadas con el turismo desde smartphones y tablets continúen en aumento, al establecimiento no le queda de otra que optimizar su web sí o sí. Los hábitos del turista están cambiando y toca adaptarse o morir.

3.1.7 Otros conceptos teóricos

Otros conceptos teóricos más sencillos serían:

- **Autenticación**: es el proceso de autenticación de algo. En nuestro caso nos referimos a autenticación de usuarios en el sistema. Cuando un usuario introduce sus credenciales en el sistema de forma satisfactoria, se confirma que forman parte de él.
- **Autorización**: La autorización protege los recursos del sistema permitiendo que sólo sean usados por aquellos usuarios a los que se les ha concedido autorización para ello. En nuestro una vez autenticado el usuario tendrá autorización a unos recursos u otros en función de su rol.
- ***Datepicker****:* un *datepicker* es un selector de fechas mediante un interfaz gráfico. Generalmente se genera mediante JavaScript y CSS.
- Pernoctación: pasar la noche en determinado lugar, especialmente si es fuera del propio domicilio. Nos referimos a pernoctaciones como noches que el cliente pasa hospedado en el establecimiento.
- **Checkin**: Fecha de entrada al establecimiento.
- **Checkout:** Fecha de salida del establecimiento.
- **Plugin:** es un módulo de código fuente que se incluye opcionalmente en una aplicación y proporciona nuevas funcionalidades.
- **Gema:** con este nombre se refiere a los *plugins* o librerías de *Ruby on Rails* que vienen empaquetadas en un fichero .gem.

Capítulo 4. Planificación del Proyecto y Resumen de Presupuestos

4.1 Planificación

Para la planificación del proyecto se ha seguido el modelo evolutivo incremental. El proceso se divide en 4 partes: análisis, diseño, código y pruebas, sin embargo, para la producción del Software, se usa el principio de trabajo en cadena o "Pipeline", utilizado en muchas otras formas de programación. Con esto se mantiene al cliente en constante contacto con los resultados obtenidos en cada incremento. Es el mismo cliente el que incluye o desecha elementos al final de cada incremento a fin de que el software se adapte mejor a sus necesidades reales. El proceso se repite hasta que se elabore el producto completo.

La necesidad de que el cliente este en continuo contacto con el desarrollo del proyecto habitualmente supone un problema ya que hay clientes que no están dispuestos a invertir el tiempo necesario, en nuestro caso al ser un proyecto dirigido a un negocio familiar eso no representa ningún problema ya que se puede mantener el contacto con el cliente de forma continua y sencilla.

4.1.1 Modelo Incremental

Los evolutivos son modelos iterativos, permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final deseado.

Características:

- Los requisitos iniciales están bien definidos al principio pero el esfuerzo necesario para un desarrollo completo excluye un proceso puramente lineal. Se puede tener la necesidad de proporcionar al usuario un conjunto limitado de funcionalidad lo antes posible, adelantarla en el tiempo, y después expandirla en entregas posteriores.
- Combina elementos del modelo en cascada aplicado de forma iterativa. Se realizan una serie de iteraciones sobre el propio modelo cascada.
- Cada iteración produce un incremento, es decir, una versión más refinada del sistema hasta alcanzar una que satisfaga los requisitos.
- Cada incremento se construye sobre aquel que ya ha sido cotejado con el cliente.

Ventajas: Obtener un producto ejecutable lo antes posible que se pone en explotación, disminuir el periodo de congelación de requisitos, los incrementos se pueden realizar para manejar determinados riesgos por ejemplo el del tamaño.

Inconvenientes: Los sistemas están a menudo pobremente estructurados, al introducir cambios el software queda muy "parcheado". En este caso evitamos esto al tener unos requisitos iniciales bien definidos.

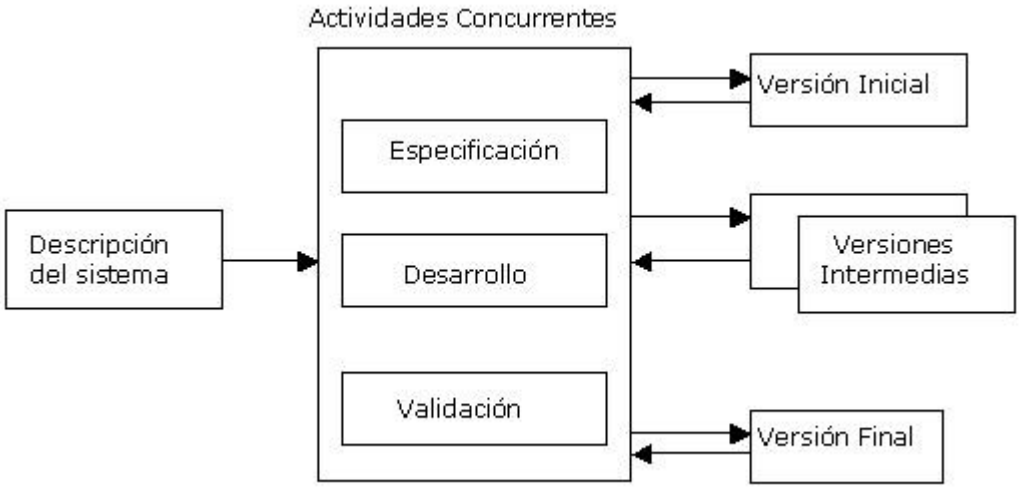


Figura 4.1. Modelo evolutivo incremental

4.1.2 Planificación temporal

A continuación se muestra una tabla con las tareas que forman el desarrollo del proyecto.

Nº	Tarea	Nivel	Avance (%)	Descripción
1	TFM	1	94	Trabajo completo
2	Trámites	2	73	Documentación oficial para el proceso de presentación del TFM
3	Estudio de proyecto	3	100	Estudio de la situación de mercado, alternativas software, mejores soluciones.
4	Propuesta de proyecto al director	3	100	Trasladar propuesta al director del TFM
5	Trámite formalización	3	100	Presentar formulario anteproyecto en secretaría previo consenso con el director

	anteproyecto			del TFM.
6	Solicitud de defensa de TFM	3	100	Presentar formulario de solicitud de defensa del TFM
7	Formación	2	100	Formación en diferentes áreas técnicas necesarias para afrontar el desarrollo del proyecto.
8	Instalación entorno de trabajo	3	100	Preparación de entorno de desarrollo: Ruby, Rails, gemas, mysql, netbeans, etc.
9	Rails 3	3	100	Formación en Rails 3
10	Plugins jQuery	3	100	Formación en JavaScript (framework jQuery)
11	Tecnologías CSS3 HTML5	3	100	Formación en maquetación de vista (html5 y css3).
12	Desarrollo	2	100	Desarrollo técnico de la aplicación
13	Análisis	3	100	En este apartado vamos a constatar hasta donde vamos a llegar el proyecto, es decir qué límites vamos a poner en el desarrollo estableciendo qué se va a hacer y qué se va a omitir.
14	Definición del sistema	4	100	Alcance del sistema
15	Extracción de requisitos del sistema	4	100	Obtención de requisitos funcionales y no funcionales.
16	Diagrama de clases preliminar	4	100	Definiremos la estructura de clases (UML) y jerarquías.
17	Reunión con cliente	4	100	Validación de los apartados anteriores con el cliente.
18	Requisitos de interfaz cliente	4	100	Se añaden o modifican los requisitos prepuestos por el cliente.
19	Diseño	3	100	Implementación del desarrollo de la versión inicial
20	Diseño de interfaz de usuario	4	100	Diseño web. Diseño del interfaz gráfico tanto de *frontend* como de *backend*
21	Elección de arquitectura del sistema	4	100	El sistema creado puede estar compuesto por varios procesos software y máquinas

				que colaboran para llevar a cabo la tarea, aquí se define.
22	Diseño de BBDD	4	100	Relaciones entre tablas de base de datos, columnas, índices. Diseño del modelo de datos (*schema*) que va a utilizar la aplicación.
23	Diseño de modelos	4	100	Diseño de los modelos. En los modelos se representa la información con la cual el sistema opera, así como el comportamiento de los objetos.
24	Reunión con cliente	4	100	Se añade o modifica la arquitectura en función de las necesidades del cliente.
25	Implementación	3	100	Procesos que comprenden el desarrollo técnico de la aplicación.
26	Crear proyecto	4	100	Punto de partida en el desarrollo. Creación del proyecto en el IDE.
27	Estructura páginas públicas	4	100	*Layout* común a todas las páginas de la parte pública.
28	Estructura páginas privadas	4	100	*Layout* común a todas las páginas de la parte privada (administración).
29	Módulo sesión	4	100	Conjunto de operaciones para permitir la autenticación de diferentes usuarios.
30	Autenticación de usuarios	5	100	Integración y configuración de la gema Devise para autenticar usuarios.
31	Roles de usuarios	5	100	Integración y configuración de la gema Cancan para gestionar roles de usuarios.
32	Vistas autenticación	5	100	Formularios de login.
33	Módulo habitaciones	4	100	Conjunto de operaciones para permitir la gestión de habitaciones.
34	Lógica gestión de habitaciones	5	100	Modelo y controlador de habitación.
35	Vistas habitaciones	5	100	Vistas públicas y administrativa para las habitaciones.
36	Módulo intervalo de precio	4	100	Conjunto de operaciones para permitir la gestión de precios.

37	Lógica gestión precios	5	100	Modelo y controlador de precios. Incluye una tabla dinámica para ofrecer información visual rápida a los visitantes.
38	Vistas intervalos precios	5	100	Vistas públicas y para la gestión de los intervalos de precios.
39	Módulo galería fotográfica	4	100	Conjunto de operaciones para permitir la gestión de fotografías.
40	Gestión de fotografías	5	100	Modelo y controlador de listas fotográficas.
41	Vistas galería y administración de esta	5	100	Vistas públicas y para la gestión de las fotografías.
42	Módulo partners	4	100	Conjunto de operaciones para permitir la gestión de partners.
43	Lógica gestión partners	5	100	Modelo y controlador de *partners*.
44	Vistas administración *partners*	5	100	Vistas públicas y para la gestión de los partners.
45	Módulo reservas	4	100	Conjunto de operaciones para permitir la gestión de reservas.
46	Lógica reservas	5	100	Modelo y controlador de reservas.
47	Calendario disponibilidad público	5	100	Visualización en un calendario JavaScript de la ocupación actual y futura del establecimiento.
48	Formulario reserva público	5	100	Implementación del formulario de reserva por parte del cliente, dividido en 4 pasos. Con validación en servidor en cada paso y generación de correos y reservas en el último paso.
49	Administración privada reservas	5	100	Gestión por parte del administrador de las reservas.
50	Facturas asociadas	5	100	Generación de factura en PDF asociada a usuario y reserva.
51	Vistas	4	100	Capa de vista de la aplicación.
52	Maquetación en detalle de parte pública	5	100	HTML y CSS de la parte pública de la web.

53	Maquetación response design	5	100	Implementación de diseño adaptativo para diferentes resoluciones. El contenido es el mismo pero la forma de visualizarlo cambia adaptándose al tamaño de la pantalla. Uso de mediaqueries.
54	Maquetación en detalle de panel administración	5	100	HTML y CSS de la parte privada de la web.
55	Contenidos parte pública	5	100	Elección de fotografías y redacción de textos para la web.
56	Estudio y optimización SEO	4	100	Estudio y aplicación de técnicas SEO para la web.
57	Pruebas	2	100	Conjunto de pruebas de la aplicación.
58	Sistema	3	100	Pruebas que permiten probar la funcionalidad de todo el sistema en conjunto.
59	Integración	3	100	Pruebas que permiten probar la integración con otros sistemas.
60	Unitarias	3	98	Pruebas para la lógica de los modelos.
61	Usabilidad	3	100	Pruebas de usabilidad en diferentes usuarios al utilizar la aplicación.
62	Documentación	2	82	Conjunto de documentación del proyecto.
63	Tomo documentación	3	97	Documentación maquetada en formato texto y encuadernada.
64	Memoria	4	100	Resumen de la motivación, objetivos y Alcance del proyecto
65	Introducción	4	100	Introducción a la funcionalidad del proyecto.
66	Aspectos teóricos	4	100	Documentación donde se relatan los aspectos teóricos relevantes del proyecto.
67	Planificación	4	100	Documentación relativa a la planificación temporal de trabajo y presupuesto, incluye diagrama Gantt.
68	Análisis	4	100	Documentación relativa al análisis teórico de la aplicación.

69	Diseño	4	100	Documentación relativa al diseño de la solución software de la aplicación.
70	Implementación	4	88	Documentación del desarrollo técnico de la solución software.
71	Pruebas	4	100	Documentación relativa a las pruebas implementadas.
72	Manuales	4	100	Manuales de usuario y de instalación.
73	Conclusiones	4	100	Conclusiones del proyecto.
74	Presupuesto	4	100	Presupuesto del proyecto detallado.
75	Apéndices y referencias	4	100	Conjunto de apéndices y referencias recopilados a lo largo del tomo de documentación.
76	Presentación	3	0	Preparación de la exposición con el apoyo de diapositivas y demostración práctica de la aplicación.
77	Exposición	3	0	Exposición del proyecto con explicación teórica y práctica delante del tribunal.

A continuación se muestra el diagrama de Gantt, se adjunta en el proyecto.

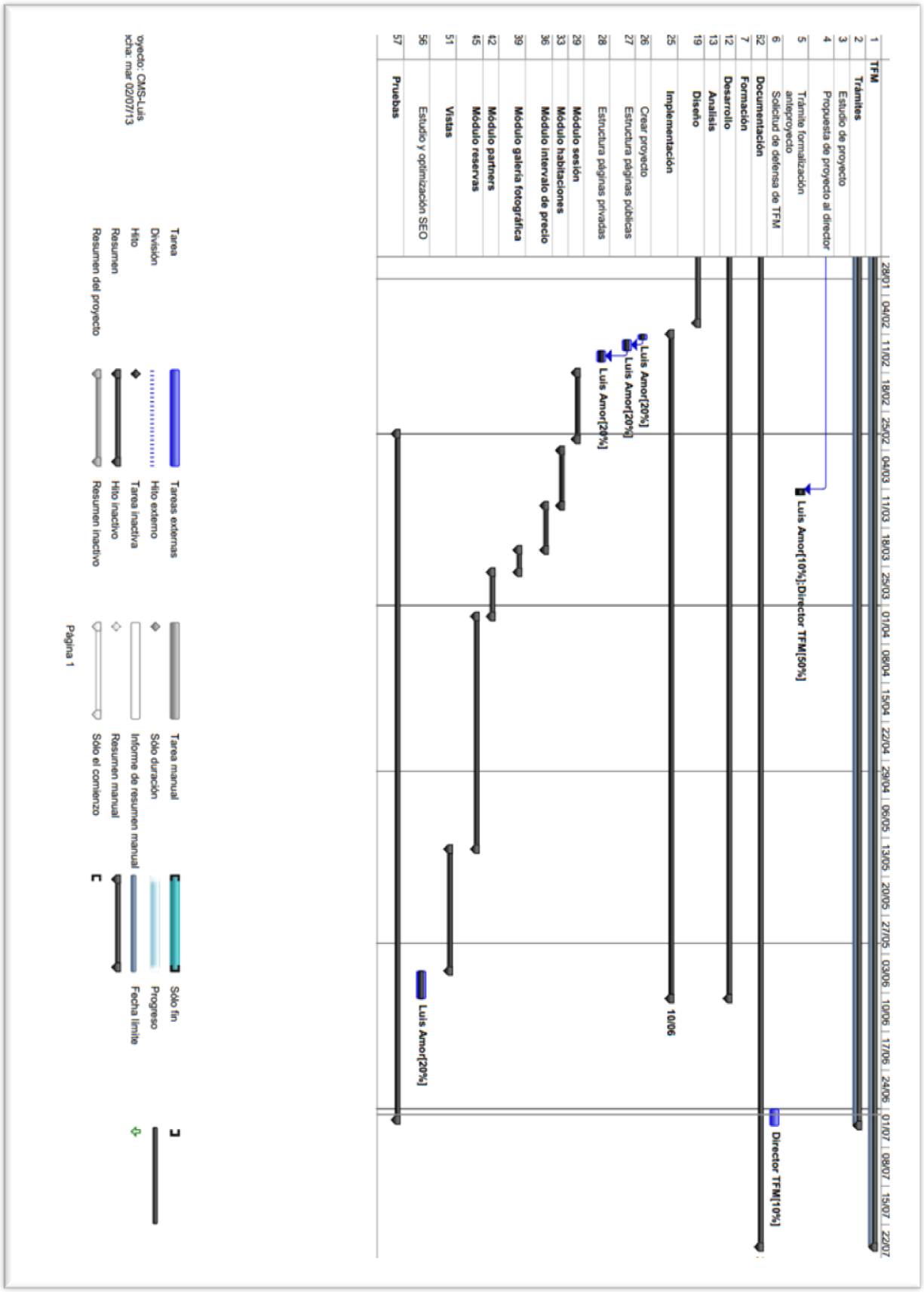


Figura 4.2. Diagrama Gantt

4.2 Resumen del Presupuesto

A continuación se muestra un resumen del presupuesto para la realización de nuestro proyecto en la tabla siguiente:

Item	Concepto	Total
1	**Desarrollo de la aplicación**	10.476,00 €
2	**Formación**	900,00 €
3	**Licencias software**	0,00 €
4	**Otros conceptos**	1.107,00 €
	Subtotal	**12.483,00 €**
	I.V.A. (21%)	**2.621,43 €**
	TOTAL	**15.104,43 €**

El presupuesto desglosado con más detalle se encuentra en la sección 11.

Capítulo 5. Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

5.1 Definición del Sistema

5.1.1 Determinación del Alcance del Sistema

Los objetivos eran por una parte la renovación de la página web del establecimiento. La web debe describir el negocio, su ubicación, las instalaciones en detalle, galería fotográfica, mostrar información de contacto así como ofrecer al visitante la posibilidad de solicitar información extra al propietario mediante un formulario de contacto.

Por otra parte también se pretende gestionar las reservas y la disponibilidad del establecimiento de una manera sencilla e intuitiva. El proyecto permite la edición de número y tipo de habitaciones, precios, alta, baja y modificación de fotográficas para la galería fotográfica, gestión de reservas y posibilidad para los clientes de consultar la disponibilidad de manera orientativa. De esta manera quedaría integrado en el sitio web el sistema de gestión de reservas. Cada cliente tendrá un usuario en la web, y podrá acceder al panel de control de su usuario dentro de la aplicación para consultar sus datos y reservas.

En un principio se contempló la posibilidad de implementar una pasarela de pago que permita al cliente realizar el pago (total o un adelanto) de la reserva en la web. Debido a la política de reservas que implica la devolución íntegra en caso de cancelación, se desestimó ya que ocasionaría un gasto en comisiones y un trabajo extra al administrador del hotel al tener que encargarse de hacer devoluciones.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

Tras las reuniones pertinentes con el cliente se han dado a conocer los siguientes requisitos del sistema:

5.2.1.1 Requisitos funcionales

Código	Nombre Requisito	Descripción del Requisito
R1.1	Panel administración privado	El administrador debe poder acceder a una parte privada mediante un formulario de login.
R1.2	Sesión usuario	Deben pedirse usuario y contraseña para iniciar sesión y acceder a la parte privada.
R1.3	Registrar usuario	Debe permitir al visitante registrarse en la web.
R2.1	Crear reservas	El sistema debe permitir al administrador crear reservas nuevas en función de la disponibilidad existente. Las reservas además deben tener un usuario asociado.
R2.2	Modificar reservas	El sistema debe permitir modificar las fechas de las reservas cumpliendo las validaciones de disponibilidad.
R2.3	Validación de reservas	Las reservas deben tener unas fechas coherentes, que la fecha de entrada no sea anterior a la actual o posterior a la salida. Validar que no se hace overbooking.
R2.4	Anular reservas	El sistema debe estar capacitado para eliminar las reservas que el administrador quiera (anulaciones).
R2.5	Crear reservas publicas	El sistema debe permitir crear reservas al visitante, para ello deberá tener un usuario registrado en el sitio web.
R2.6	Validar reservas	El sistema debe validar que la reserva es correcta, que la fecha de entrada no sea anterior a la actual o posterior a la salida.
R2.7	Modificar reservas	En cualquier momento el sistema debe ser capaz de modificar los campos de una reserva previamente almacenada.
R2.8	Eliminar reserva	El sistema debe ser capaz de eliminar reservas por si se sufre alguna cancelación.
R2.9	Visualizar reservas de forma intuitiva	El sistema debe proporcionar al administrador algún mecanismo para ver las reservas de forma gráfica.
R2.10	Facturas asociadas a reservas	El sistema debe ser capaz de crear una factura asociada a la reserva en un formato que se pueda guardar en el disco duro local.
R2.11	Mostrar disponibilidad	El sistema debe informar a los potenciales clientes de la ocupación del establecimiento de una manera

		orientativa en función de las reservas que tenga registradas en su base de datos.
R3.1	Administrar usuarios	El sistema debe permitir al administrador crear y modificar usuarios.
R4.1	Gestionar habitaciones	El sistema debe permitir al administrador modificar el número de habitaciones disponibles en el hotel, así como sus características. Cada habitación estará asociada a un tipo de habitación.
R4.2	Tipo de habitación	Las habitaciones se agruparán por tipos. Cada tipo de habitación diferente tiene un precio asociado.
R5.1	Mostrar precios	El sistema debe mostrar al visitante una tabla de precios, siempre y cuando estén definidos.
R5.2	Administración de precios	El sistema debe permitir crear y modificar intervalos de tiempo con un precio concreto para cada tipo de habitación.
R5.3	Validar precios	El sistema debe validar que hay precios establecidos para todos los días de la temporada y que los intervalos de tiempo en que se aplican los precios no se pisan entre ellos.
R6.1	Administrar fotográficas	El sistema debe poder subir archivos de imagen al servidor para mostrarlos en la web en varios tamaños. En la galería de fotos, deben poder crearse nuevas fotografías, editarlas y borrarlas.
R7.1	Información de la web	La web debe contener al menos la siguiente información: descripción del negocio, descripción de las habitaciones, galería de fotos, información de contacto, mapa y dirección.
R7.2	Soporte varios idiomas	La web debe estar internacionalizada.
R7.3	Formulario de reserva	El sistema debe permitir al cliente generar una reserva desde la web.
R7.4	Formulario de contacto	El sistema debe enviar un email al correo del administrador al rellenar el formulario de contacto
R7.5	Comunicación por email	El sistema debe enviar un email de confirmación al usuario que relleno el formulario de contacto, informándole de que la el formulario ha funcionado correctamente y que la empresa se pondrá en contacto con él.

5.2.1.2 Requisitos no funcionales

- **Requisitos Tecnológicos**: Al ser un proyecto web es multiplataforma, debe subirse a un servidor que tenga instalado un servidor Ruby on Rails y soporte MySQL. Para su utilización es necesario un navegador web moderno que soporte Javascript y conexión a Internet. Debe ser compatible con los principales navegadores del mercado.
- **Requisitos de seguridad**: se debe controlar el acceso a todas las funciones de la parte privada con un formulario, en caso de intentar acceder directamente a una funcionalidad privada debe redirigirse al formulario de login. Es necesario realizar copias de seguridad de la base de datos ya que es donde se guarda toda la información de los usuarios y las reservas.

- **Requisitos de rendimiento**: El sistema debe estar operativo 365 días al año, 24 horas al día. El peso de la web debe ser ligero, así como la velocidad de acceso a la base de datos.

5.2.1.3 Atributos del software

A continuación, se especifican los atributos del sistema software que el proyecto debe cumplir:

- **Escalabilidad**: el proyecto ha de ser sensible a cambios de tal forma que se puedan introducir nuevas funcionalidades o modificar las existentes de manera sencilla.
- **Usabilidad**: consiste en que debido a él fácil uso y aprendizaje del sistema no se requiere de amplios conocimientos de informática
- **Seguridad**: capacidad del sistema de operar sin fallos, tanto a nivel operacional como de privacidad.
- **Confiabilidad**: grado de confianza que depositan los usuarios en el sistema. Interesa que sea el mayor posible. El nivel de confiabilidad se mide en función de:
    - **Disponibilidad**: capacidad del sistema para entregar los servicios cuando son requeridos.
    - **Fiabilidad**: capacidad del sistema para entregar los servicios como han sido especificados.
- **Eficiencia**: el software creado debe evitar el malgasto de recursos. Hay que intentar obtener la mayor eficiencia utilizando el menor número de recursos posible.
- **Accesibilidad**: software utilizables por cualquier persona con cierta discapacidad.
- **Portabilidad**: software diseñado para poder ser ejecutado y trabajar con él desde diferentes plataformas. El proyecto debería funcionar en cualquier sistema operativo y navegador.
-

5.2.2 Identificación de Actores del Sistema

Existen tres actores principales:

- El usuario anónimo o **visitante** que visita las secciones de la web como potencial cliente. Pueden ver los contenidos de la parte pública pero carecen de credenciales para acceder a la parte privada.
- El **cliente** es aquel visitante que está registrado en la base de datos de la aplicación. Puede acceder a la parte privada, pero sólo a su panel de control, donde puede ver las reservas que ha hecho y modificar sus datos personales.
- El **administrador** del sitio web. Tiene acceso al panel de administración de la parte privada y por lo tanto administrar usuarios, fotografías, precios y gestión de reservas. El administrador tiene control total en todas las secciones.

5.2.3 Especificación de Casos de Uso

A partir de la tabla de los requisitos funcionales, se puede organizar los casos de uso en 7 grupos:

- Gestión cuenta cliente
- Crear/administrar reservas
- Gestión usuarios
- Administración habitaciones
- Gestionar precios
- Gestionar fotografías
- Navegación visitante

A continuación se muestran diagramas de algunos de los casos de uso más relevantes junto a una breve descripción:

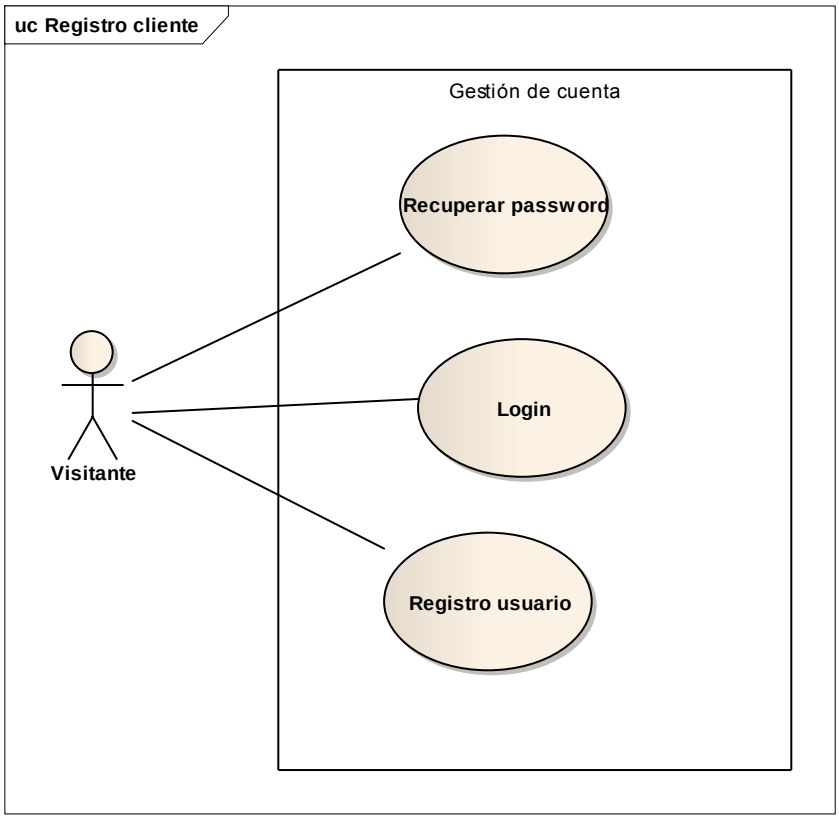


Figura 5.2. Caso de uso gestión cuenta cliente

Nombre del Caso de Uso
Login
Descripción
Proceso de autenticación del usuario a través de un formulario de login para acceder al panel de control. El usuario puede ser un cliente o un administrador.

Nombre del Caso de Uso
Recuperar password
Descripción
El sistema envía al correo electrónico un correo para recuperar su password.a través de un formulario. El usuario puede ser un cliente o un administrador.

Nombre del Caso de Uso
Registrar usuario
Descripción
El sistema permite al visitante registrarse en el sistema como cliente a través de un formulario. El sistema enviará un email de confirmación al cliente y al administrador.

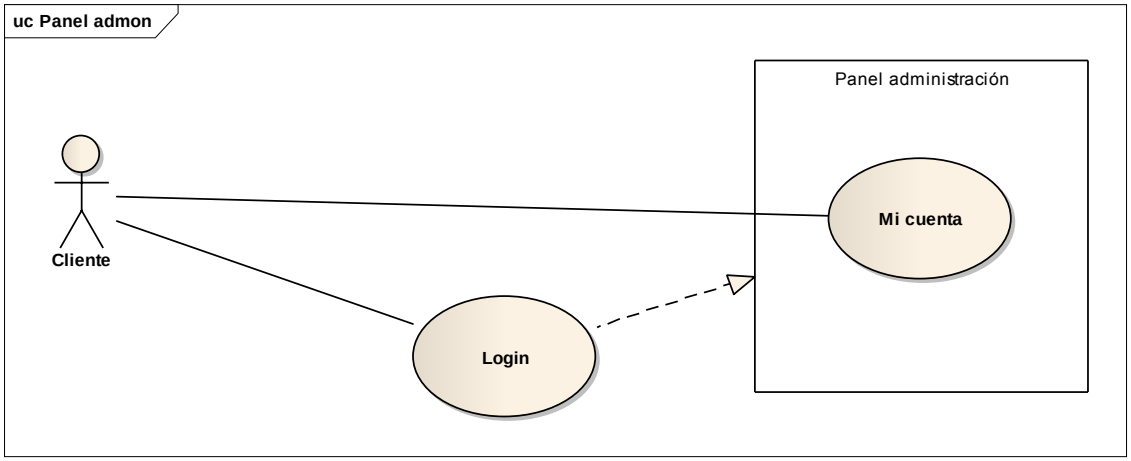


Figura 5.1. Caso de uso **Gestión cuenta cliente 2**

Nombre del Caso de Uso
Mi cuenta
Descripción
Cuando un cliente inicia sesión en el sistema puede acceder a su panel de administración, donde puede editar su información personal y visualizar las reservas asociadas a su usuario.

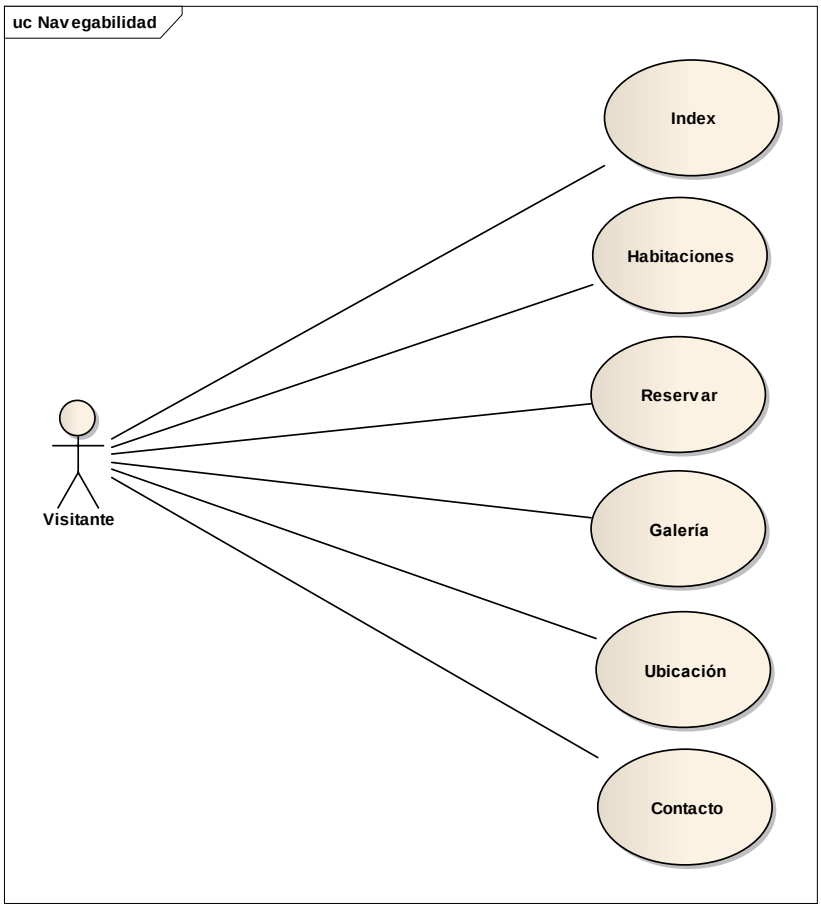


Figura 5.1. Caso de uso **navegación por la web**

Nombre del Caso de Uso
Index
Descripción
Página principal de la web, cuenta con una galería de fotos y la descripción del negocio.

Nombre del Caso de Uso
Habitaciones
Descripción
Describe las habitaciones del establecimiento.

Nombre del Caso de Uso
Reservar
Descripción
Muestra el calendario de disponibilidad y el formulario de selección de fechas para la reserva. El formulario consta de 4 pasos que se detallan más adelante.

Nombre del Caso de Uso
Galería
Descripción
Página que contiene una galería fotográfica interactiva del establecimiento.

Nombre del Caso de Uso
Ubicación
Descripción
Página que contiene la dirección detallada del establecimiento, incluyendo un mapa interactivo.

Nombre del Caso de Uso
Contacto
Descripción
Página con dirección de contacto, tanto física, como email, como teléfono además de un formulario de contacto directo con el administrador.

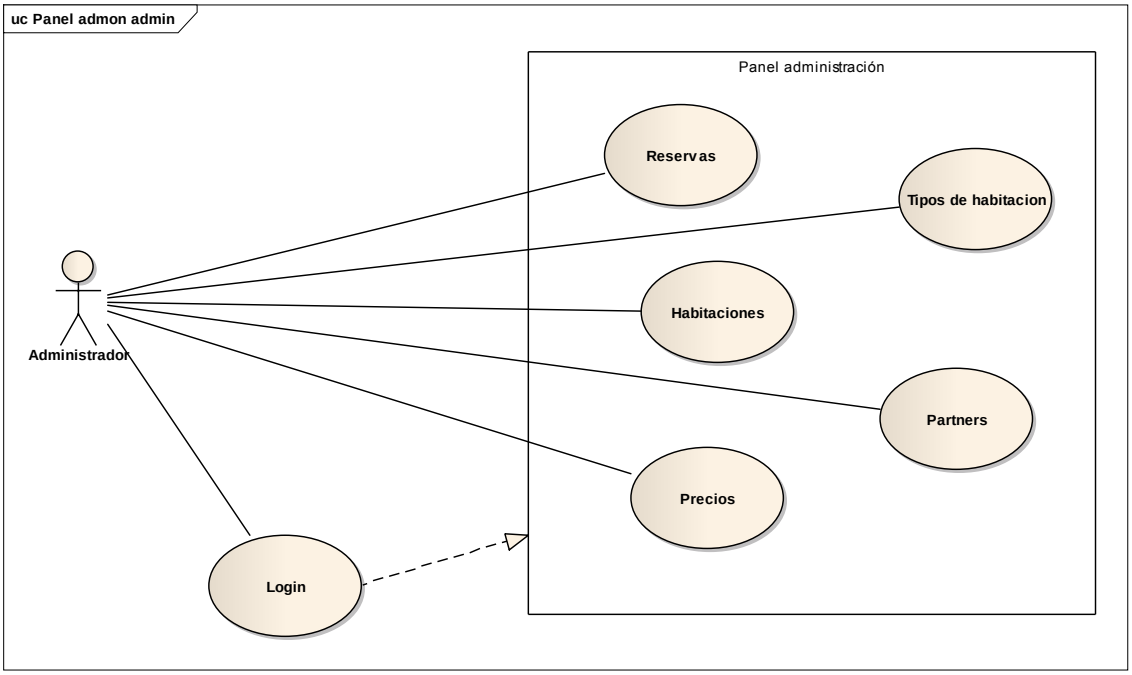


Figura 5.2. Caso de uso **Panel de administración privado**

Nombre del Caso de Uso
Reservas
Descripción
Cuando el administrador esta autenticado en el sistema puede acceder a la gestión de reservas. Tiene varios modos para visualizarlas

Nombre del Caso de Uso
Habitaciones
Descripción
Cuando el administrador esta autenticado en el sistema puede gestionar (altas, bajas, modificaciones) las habitaciones asociadas al establecimiento.

Nombre del Caso de Uso
Tipos de habitación
Descripción
Cuando el administrador esta autenticado en el sistema puede gestionar (altas, bajas, modificaciones) los tipos de habitación existentes en el establecimiento. El sistema mostrará un formulario con los campos necesarios para cada operación.

Nombre del Caso de Uso
Partners
Descripción
Cuando el administrador esta autenticado en el sistema puede gestionar (altas, bajas, modificaciones) los tipos de partners (socios turísticos) asociados al establecimiento. El sistema mostrará un formulario con los campos necesarios para cada operación.

Nombre del Caso de Uso
Precios
Descripción
Cuando el administrador esta autenticado en el sistema puede gestionar (altas, bajas, modificaciones) los intervalos de precios asociados a cada tipo de habitación. El sistema mostrará un formulario con los campos necesarios para cada operación.

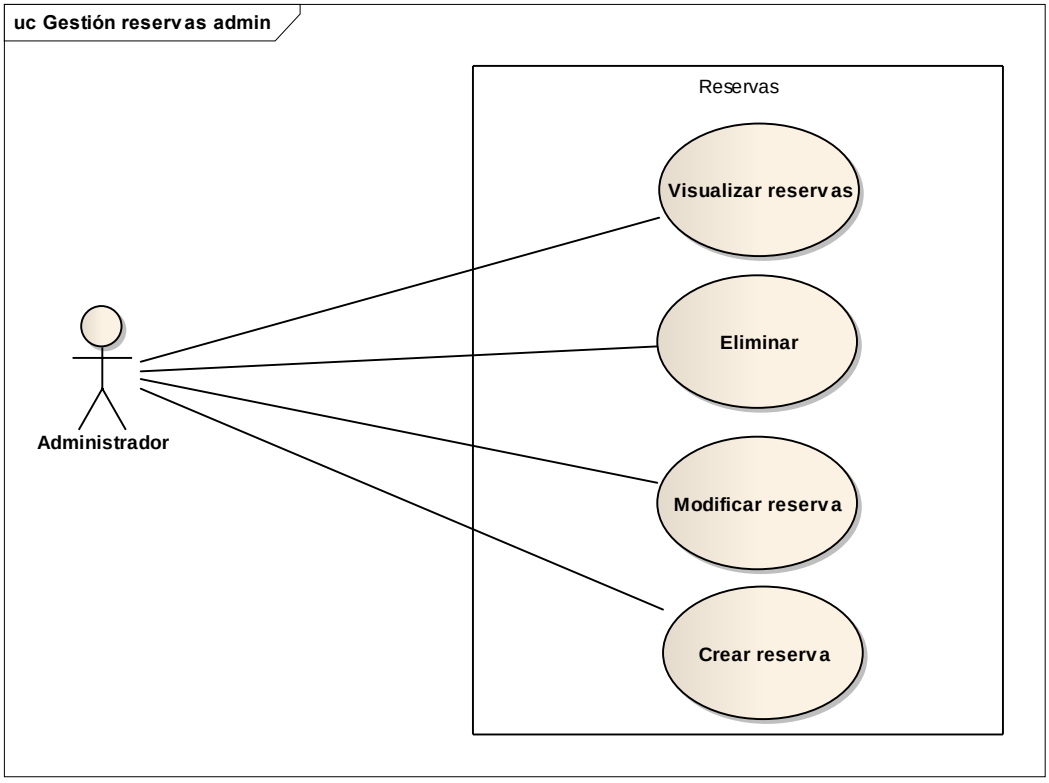


Figura 5.2. Caso de uso **gestionar reservas**

Nombre del Caso de Uso
Visualizar reservas
Descripción

Cuando el administrador esta autenticado en el sistema puede ver las reservas en modo tabla, con los datos relevantes organizados por columnas o en modo calendario, de manera más intuitiva.

Nombre del Caso de Uso
Eliminar reserva
Descripción
Cuando el administrador esta autenticado en el sistema puede eliminar una reserva que haya sido cancelada por el cliente. El sistema mostrará una alerta de confirmación antes de eliminar la reserva.

Nombre del Caso de Uso
Modificar reservas
Descripción
Cuando el administrador esta autenticado en el sistema puede cambiar el número o tipo de habitaciones, así como las fechas de la reserva. El sistema mostrará un formulario con los campos necesarios.

Nombre del Caso de Uso
Crear reserva
Descripción
Cuando el administrador esta autenticado en el sistema puede crear nuevas reservas, especificando directamente las habitaciones y el cliente implicados. El sistema mostrará un formulario con los campos necesarios.

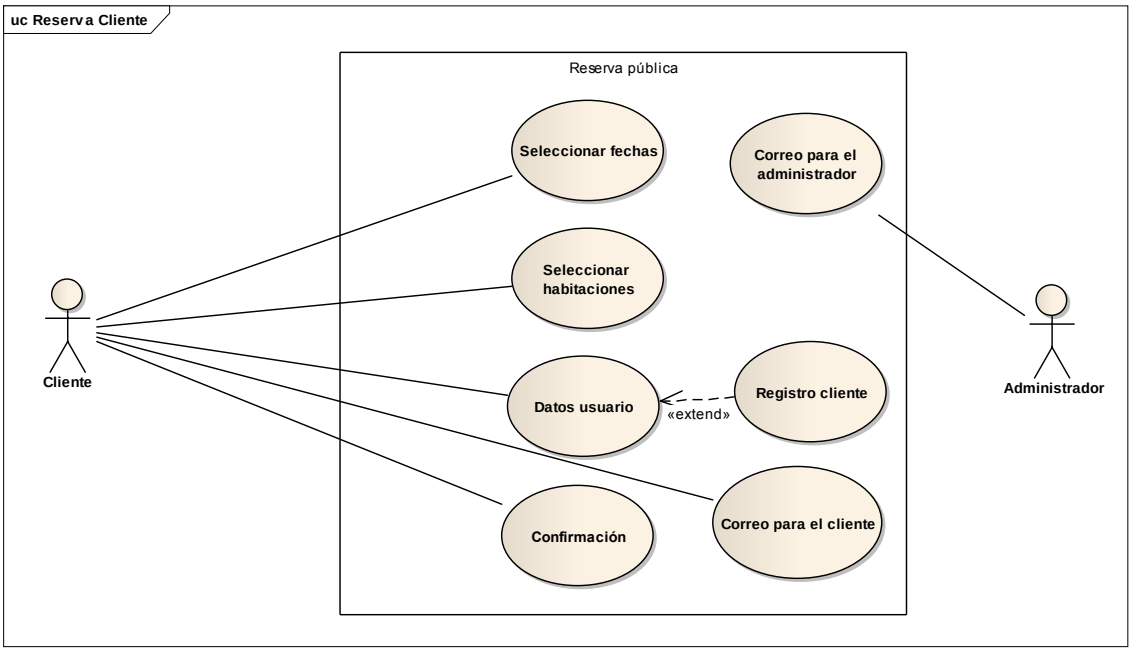


Figura 5.2. Caso crear reserva: pública

Nombre del Caso de Uso
Seleccionar fechas (paso 1)
Descripción
El visitante puede seleccionar una fecha de entrada y una de salida a partir de dos *date pickers*. El sistema recogerá estas fechas, validará que sean consistentes y mostrara la disponibilidad de habitaciones (en casa de existir) para esas fechas. En la respuesta se mostrará el precio para cada habitación, tanto doble como individual en función del tipo.

Nombre del Caso de Uso
Seleccionar habitaciones (paso 2)
Descripción
Tras el primer paso (selección de fechas) el visitante llegaría a la selección de habitaciones. Se muestran el precio de cada tipo de habitación para 1 habitación durante el intervalo de tiempo señalado en el paso anterior. Se ofrece la posibilidad de seleccionar habitaciones de diferentes tipos, y todas las disponibles. Al pasar de 1 a varias habitaciones también se mostraría el precio asociado.

Nombre del Caso de Uso
Datos de usuario (paso 3)
Descripción
Tras el segundo paso (selección de habitaciones) se solicita al visitante sus datos personales. • Si es un usuario autenticado, en vez de solicitar sus datos se mostrarían en pantalla, y se le permitiría pasar al siguiente paso. • Si no está autenticado, al pedir sus datos se crearía automáticamente el usuario en la base de datos con los datos recogidos. Se pasaría al siguiente paso con ese usuario autenticado y asociado a la reserva.

Nombre del Caso de Uso
Confirmación (paso 4)
Descripción
Tras el tercer paso ya tenemos todos los datos necesarios. Se muestran por pantalla, si el usuario está de acuerdo puede pulsar en finalizar con lo cual (tras la validación en el servidor de toda la información) se registraría la reserva en la base de datos y se enviaría correos electrónicos con los datos de la reserva tanto al administrador como al cliente.

5.3 Identificación de los Subsistemas en la Fase de Análisis

5.3.1 Descripción de los Subsistemas

En el sistema se pueden definir los siguientes subsistemas:

1. Subsistema de sesión
2. Subsistema de gestión de reservas.
3. Subsistema de gestión de usuarios.
4. Subsistema de gestión de habitaciones.
5. Subsistema de gestión de precios.
6. Subsistema de gestión de fotografías.
7. Subsistema de vistas

5.4 Diagrama de Clases Preliminar del Análisis

5.4.1 Diagrama de Modelos

El modelo es la representación específica de la información con la cual el sistema opera.

A continuación muestro la estructura de modelos de la aplicación, este diagrama ha sido generado automáticamente a partir del código fuente del proyecto mediante la herramienta **RailRoady**. Es un generador de diagramas de clases para aplicaciones desarrolladas con Ruby on Rails. Se trata de un script Ruby que carga las clases de la aplicación y analiza sus propiedades y relaciones. Su salida es la descripción de un grafo en el lenguaje DOT.

Dado que Ruby on Rails es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en ActiveRecord, las definiciones de las clases no necesitan especificar los nombres de las columnas, Ruby puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

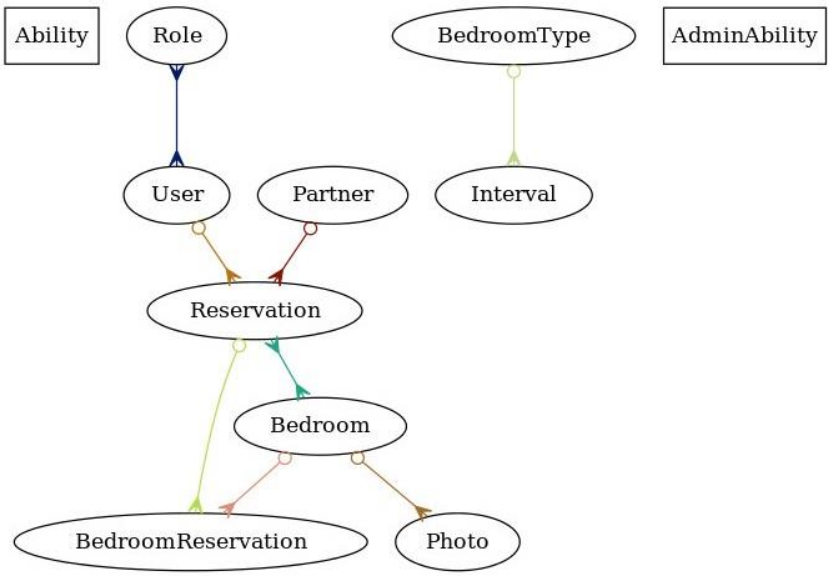


Figura 5.3. Diagrama de modelos generado

5.4.2 Descripción de los Modelos

A continuación se hace una breve descripción de los modelos de la aplicación organizados por subsistema.

5.4.2.1 Subsistema de sesión

Nombre del modelo
Session
Descripción
Representa una sesión de un usuario en el sistema. Guarda la ip desde donde se crea
Responsabilidades
Cuando un usuario crea sesión (login) esta se crea en la BD, cuando se hace el logout la sesión se destruye. Su misión es autenticar a un usuario al encontrar el mismo nombre y contraseña en la tabla de personas
Atributos Propuestos
ip_address: dirección IP desde donde se crea la sesión. **path:** guarda la última vista visitada
Métodos Propuestos
authenticate_person: comprueba si las credenciales introducidas son correctas para la autenticación. **asociate_person_to_session**: asocia una persona con una sesión **sesion_has_been_asociated:** comprueba si la sesión tiene asociada un usuario

5.4.2.2 Subsistema de gestión de usuarios

Nombre de la Clase
User
Descripción
Modelo que representa posibles usuarios, administradores o clientes. Permite la administración de estos.
Responsabilidades
Valida que los usuarios sean únicos, el formato del nombre y la contraseña, permite validar la contraseña y usuario para autenticarse, modifica el rol de los usuarios.
Atributos Propuestos
dni: DNI o equivalente del cliente **role_ids**: roles asociados con el cliente **username**: nombre de usuario **name**: nombre **surname**: apellido **city**: ciudad **post_code**: código postal **country**: país **email**: email del usuario **password**: contraseña del usuario **password_confirmation**: campo adicional para asegurarnos de que el usuario introduce un campo correcto en la contraseña **telephone**: teléfono
Métodos Propuestos
setup_role: Cuando se crea un usuario se le asigna un rol por defecto **check_for_reservations**: comprueba si el usuario tiene reservas asociadas **role?:** devuelve el tipo de rol del usuario

.

5.4.2.3 Subsistema de gestión de habitaciones.

Nombre del modelo
Bedroom
Descripción
Representa las habitaciones en el sistema.
Responsabilidades
Representa a la entidad habitación. Debe controlar que la construcción de una habitación sea consistente.
Atributos Propuestos
name: nombre de la habitación **bedroom_type_id**: es el tipo de habitación **get_price**: devuelve el precio en función del tipo de habitación **photos:** fotografías asociadas **supports_extra_bed**: atributo booleano que determina si en la habitación cabe una cama extra. **tv**: disponibilidad de televisión **parking**: disponibilidad de parking **bath**: disponibilidad de baño propio en la habitación

breakfast: desayuno incluido en la habitación **handicapped**: determina si la habitación está preparada para usuarios minusválidos
Métodos Propuestos
set_main_photo: subir una foto al servidor como presentación de la habitación.

Nombre del modelo
Bedroom_type
Descripción
Representa el tipo de habitaciones que se pueden crear y reservar en el sistema.
Responsabilidades
Representa el tipo de habitación. Debe proporcionar un precio correcto para un intervalo de fechas.
Atributos Propuestos
intervals: lista de intervalos de precios **name**: nombre del tipo de habitación **individual_difference:** diferencia de precio entre habitación individual y doble **breakfast**: desayuno incluido en la habitación **handicapped**: determina si la habitación está preparada para usuarios minusválidos
Métodos Propuestos
get_range_price: devuelve un precio para el tipo de habitación, recibiendo las fechas y el número de personas que se hospedaran en la habitación.

5.4.2.4 Subsistema de gestión de precios.

Nombre del modelo
Interval
Descripción
Representa el precio de un tipo de habitación, en un intervalo de tiempo.
Responsabilidades
Representa el precio de un tipo de habitación. Debe controlar que todos los días de la temporada tengan un precio asignado, y que solo haya un intervalo aplicable a cada día del año. El intervalo debe ser único, y validar las fechas.
Atributos Propuestos
bedroom_type: tipo de habitación asociada al intervalo **name**: nombre del intervalo **price**: precio del intervalo **start**: día de comienzo del intervalo **end**: día de fin
Métodos Propuestos
contain_day: devuelve un booleano, si la fecha que recibe como parámetro está contenida en el intervalo.

5.4.2.5 Subsistema de gestión de fotografías.

Nombre del modelo
Photo
Descripción
Representa un fichero de imagen
Responsabilidades
Tratar la imagen y redimensionarla a varios tamaños o formatos.
Atributos Propuestos
Name: nombre del objeto **image:** representa el objeto imagen con el que trabajaremos **image_file_name**: nombre del archivo imagen **image_content_type:** formato de la imagen **image_file_size:** tamaño de la imagen **image_updated**: fecha edición de la imagen
Métodos Propuestos

5.4.2.6 Subsistema de gestión de reservas.

Nombre del modelo
Reservation
Descripción
Representa al objeto reserva
Responsabilidades
Validar la disponibilidad de habitaciones para el hotel, ofrecer información de disponibilidad al cliente o el administrador, crea y gestiona las reservas.
Atributos Propuestos
user: cliente asociado a la reserva **bedrooms:** habitaciones asociadas a la reserva **partner**: empresa relacionada con la reserva **check_in:** día de entrada **check_out:** día de salida
Métodos Propuestos
right_dates: consistencia de fechas **nights**: número de pernoctaciones **price_for_type**: precio para cada tipo de habitación y las fechas de la reserva **price**: calcula el precio total de la reserva **available_bedrooms_for_new_reservation**: devuelve las habitaciones disponibles para el intervalo de fechas de la reserva **available_rooms:** devuelve las habitaciones disponibles para un día concreto. **available_bed_types_for_new_reservation:** habitaciones disponibles para el tipo de habitación solicitado. **has_bedroom_type**: devuelve un booleano si la reserva contiene habitaciones del tipo señalado.

asign_bedrooms_for_reservations: transforma una petición de N habitaciones cualquiera de un tipo, en N habitaciones concretas de ese tipo.

Nombre del modelo
Partner
Descripción
Representa una empresa externa asociada, que envía clientes al establecimiento a cambio de una comisión.
Responsabilidades
Simplemente informar al administrador de que clientes vienen a través de un socio. Se asocia a cada reserva, si procede.
Atributos Propuestos
name: nombre del socio **telephone:** teléfono **web:** dirección web del socio turístico **comments:** comentarios al respecto para este socio **logo:** logotipo del partner **logo_file_name**: nombre del archivo imagen **logo_content_type:** formato de la imagen **logo_file_size:** tamaño de la imagen

5.6 Análisis de Interfaces de Usuario

En el proyecto hay dos interfaces de usuario diferentes y bien definidas, por un lado lo que en conjunto representa el sitio web, sería la interfaz pública.

Por otra parte el panel de administración representaría la interfaz privada, desde la cual se pueden acceder a los diferentes subsistemas. El cliente, una vez registrado tiene acceso a la interfaz privada, pero sólo a su panel de usuario.

5.6.1 Descripción de la Interfaz

Diseño base para las páginas públicas:

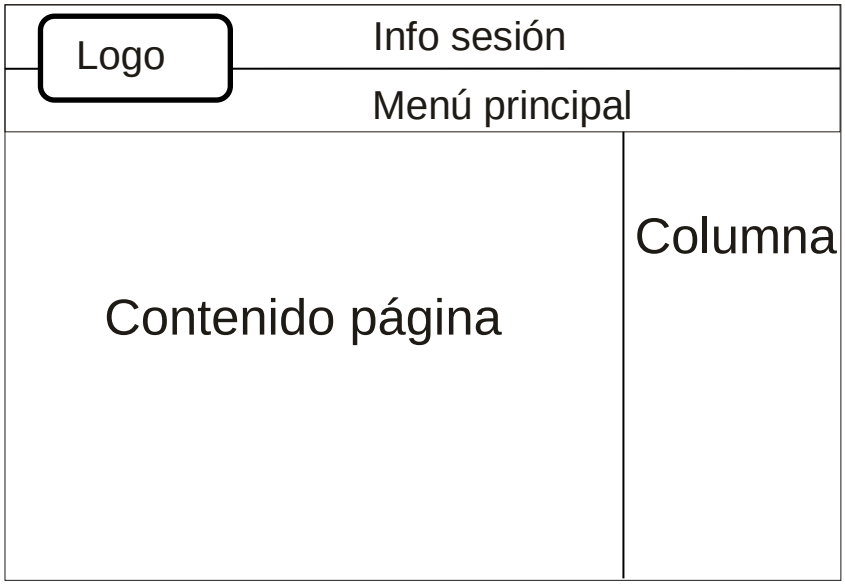


Figura 5.6. Boceto de una interfaz pública

Esta parte representaría el sitio web público, al que tienen acceso los potenciales clientes del establecimiento. Constaría de las secciones: introducción, localización, actividades, galería, precios, contacto y disponibilidad.

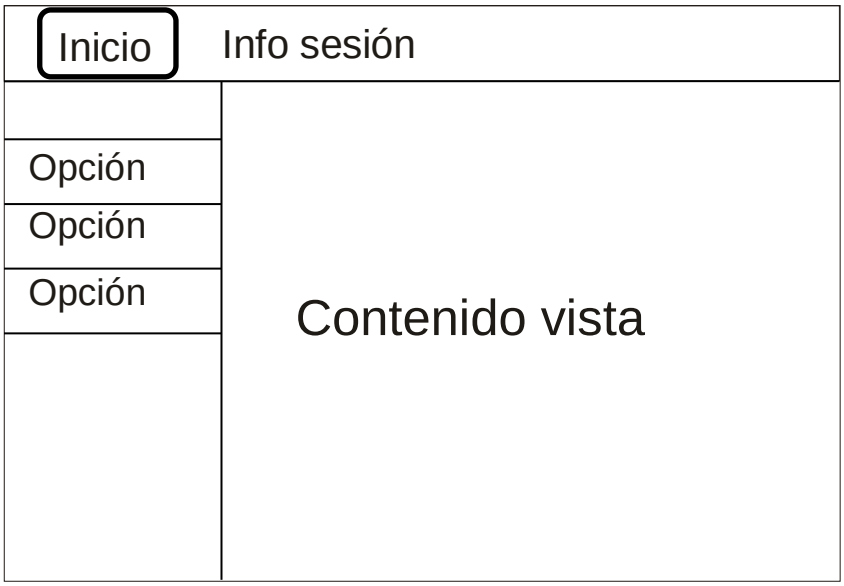


Figura 5.6. Boceto de una interfaz privada

Esta parte representaría el sitio web privado, al que tienen acceso el administrador y el cliente registrado. El administrador accedería a las vistas para gestionar reservas, habitaciones, precios, fotografías, partners, y usuarios. El cliente sólo podría acceder a su panel de control de su cuenta.

5.6.2 Descripción del Comportamiento de la Interfaz

La interfaz pública debe tener un diseño adaptable al tamaño de la pantalla del dispositivo con el que se visualice la web.

Se optimizará el contenido para 4 resoluciones diferentes:

- **Base**: para resoluciones mayores o iguales a 960px de ancho
- **Tabletas**: para resoluciones entre 768px y 959px de ancho
- **Móviles (horizontal)**: para resoluciones entre 480px y 767px de ancho
- **Móviles (vertical)**: para resoluciones entre 320px y 480px de ancho

5.6.3 Diagrama de navegabilidad

En esta sección se incluye un diagrama que muestre la navegación que habrá entre las pantallas del programa y su relación con las computaciones que tienen lugar en las mismas. Se muestran las transiciones entre pantallas y no el contenido de cada pantalla en sí.

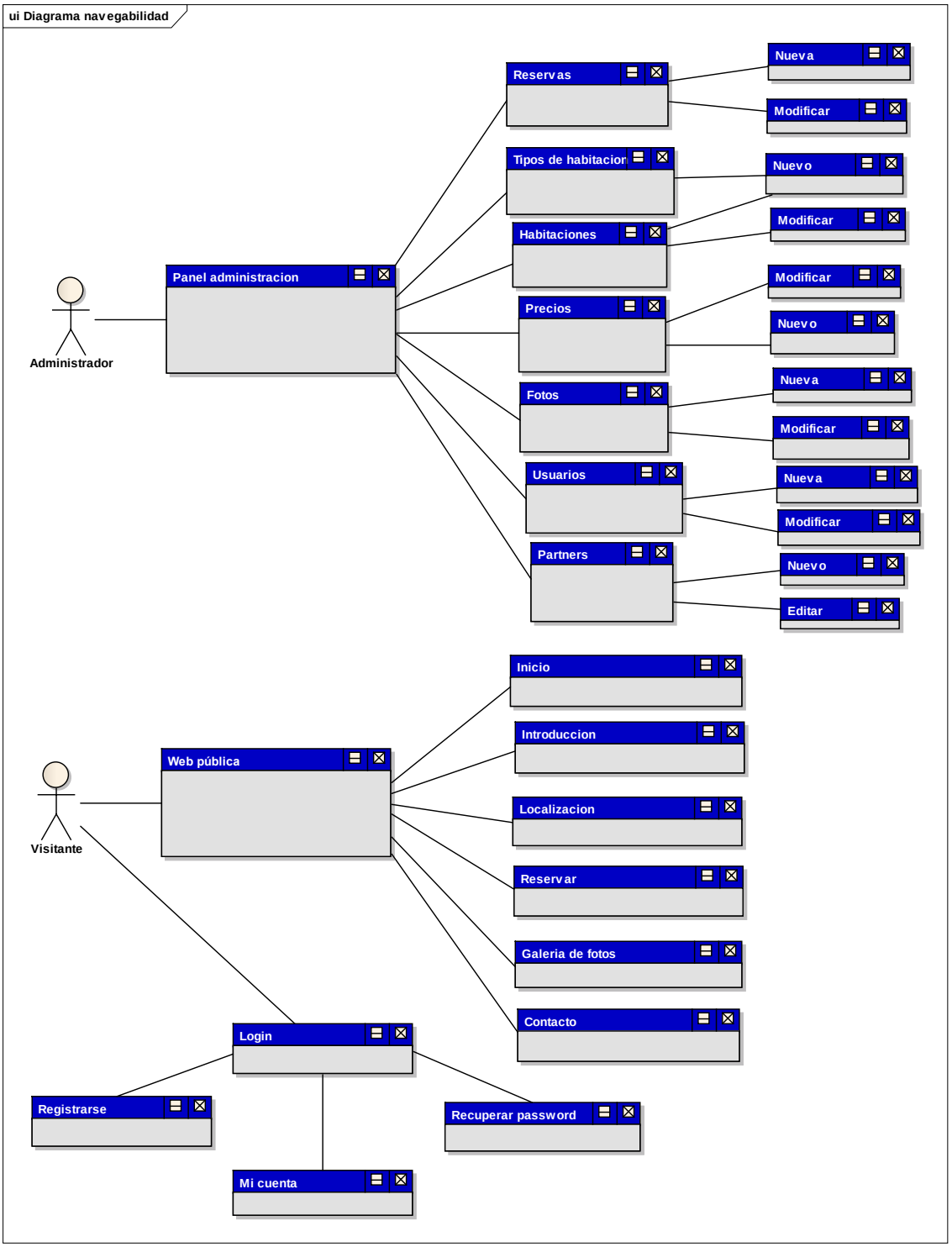


Figura 5.6. Diagrama de navegabilidad

5.7 Especificación del Plan de Pruebas

Las pruebas realizadas durante el desarrollo del proyecto y al finalizar su implementación consistían principalmente en probar todas las posibles situaciones que podían darse para una funcionalidad dada. Para cada una de estas situaciones, se comprobaba que el sistema respondía tal y como se esperaba; resolviendo los posibles errores que iban surgiendo.

Se contemplarán hasta cuatro tipos de pruebas:

- **Pruebas Unitarias**: A partir de los casos de uso, los escenarios y clases vistos anteriormente, desarrollaremos pruebas unitarias que consideremos necesarias y especificaremos los resultados que se espera encontrar una vez ejecutada la operación sobre cada una de ellas. Para su creación utilizaremos una gema llamada *Rspec*, se creará una batería de pruebas la cual nos permitirá saber si al introducir un cambio en un modelo este puede afectar a la funcionalidad del mismo. De este modo podemos tener un sistema más robusto frente a posibles cambios.

- **Pruebas de Integración**: Las pruebas de integración comprenden verificaciones asociadas a grupos de componentes, verificando que éstos funcionan correctamente cuando estos son ensamblados para cumplir con una función concreta. Para ello, cada escenario debe probarse con el mayor número de entradas posibles (y relevantes) que sea posible, incluyéndose entradas con datos correctos y con datos incorrectos para probar que el sistema reacciona correctamente ante errores de los usuarios. Para elaborar estas pruebas debemos tener en cuenta las características de la aplicación.

- **Pruebas del sistema**: Las pruebas de sistema se llevarán a cabo durante el desarrollo y al final de este. Se dará una versión del software al cliente, el cual podrá probar el software e indicarnos posibles fallos o mejoras.

- **Pruebas de Usabilidad y Accesibilidad**: Las pruebas de usabilidad y accesibilidad se han dejado para el final del proyecto cuando se tenga una versión estable de la interfaz de este.

Capítulo 6. Diseño del Sistema

6.1 Arquitectura del Sistema

6.1.1 Diagramas de Paquetes

Nuestra aplicación sigue el patrón Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón de llamada y retorno MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

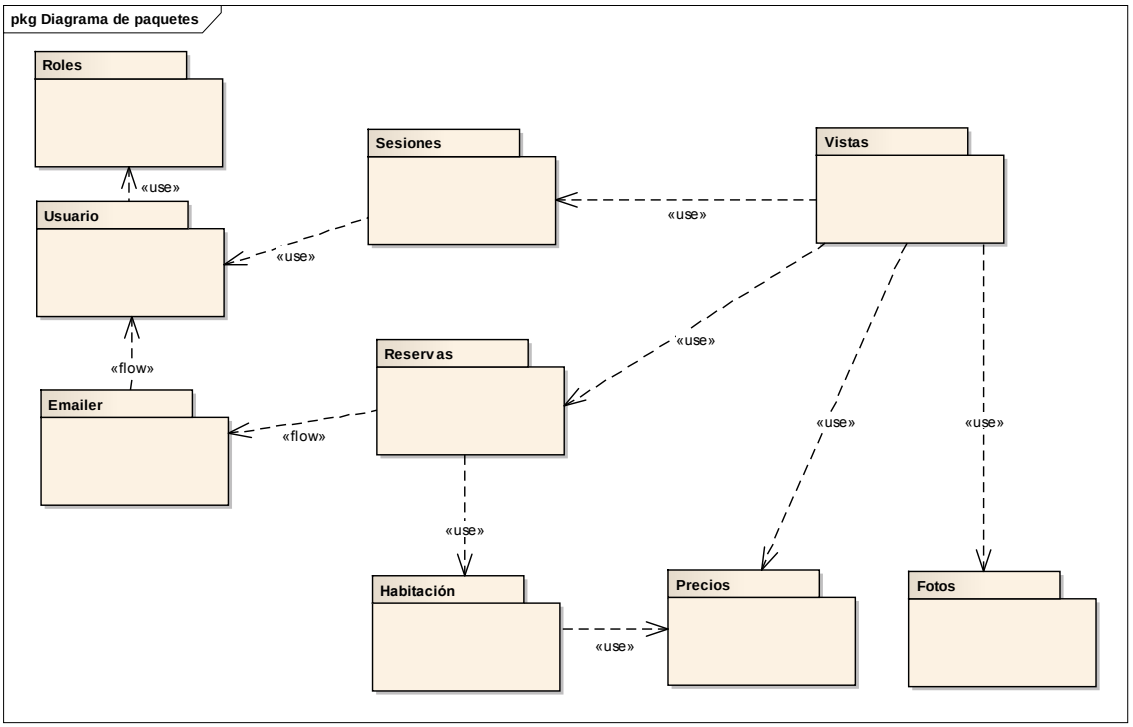


Figura 6.1 Diagrama de paquetes

6.1.1.1 Sesiones

Es el paquete con la lógica encargada de validar si una sesión es correcta, guarda las sesiones que se inician en la aplicación en su correspondiente tabla de la base de datos. Utiliza el modelo de roles para validar los permisos de un usuario que ha iniciado sesión.

6.1.1.2 Usuarios

Contiene el modelo Usuario, su controlador y sus vistas, que es necesario para crear y manejar clientes registrados. Las reservas tienen asociadas un cliente.

6.1.1.3 Emailer

Es el encargado de generar correos electrónicos. Genera correos tanto para el cliente como para el administrador en función del caso de uso. Los correos para el cliente tienen estructura corporativa en formato HTML.

6.1.1.4 Vistas

Representa la capa de presentación de la web, es el conjunto de fuentes, JavaScript, CSS, HTML e imágenes. En la vista se representan todos los datos de la aplicación.

6.1.1.5 Reservas

Contiene el modelo, vistas y controlador para gestionar las reservas del establecimiento, además de mostrar la disponibilidad de este. Una reserva tiene como mínimo una habitación pero puede tener varias. Las reservas están asociadas a un usuario.

6.1.1.6 Habitación

Contiene el modelo de Habitación y de Tipo de habitación. Las habitaciones tienen un tipo asociado. Las reservas de los clientes se hacen sobre el tipo de habitación, no sobre la propia habitación. Se tiene en cuenta el número de habitaciones disponibles para informar al cliente del número disponible para cada tipo. El precio de cada reserva se obtiene a partir del precio asignado a cada tipo de habitación.

6.1.1.7 Precios

Los precios de las habitaciones se fijan mediante intervalos de tiempo. Cada tipo de habitación tiene sus intervalos, es recomendable que sean los mismos pero no obligatorio. Una fecha no puede estar afectada por dos intervalos, y obligatoriamente debe estar incluida en un intervalo (sólo uno).

6.1.1.8 Fotos

El usuario puede gestionar una lista de fotografías, agregar nuevas, modificar títulos y posición y eliminarlas. Estas fotos se subirán al servidor en varios tamaños y se mostrarán en la página pública *Galería fotográfica*.

6.1.2 Diagramas de Despliegue

El diagrama de despliegue que define nuestra aplicación es el siguiente.

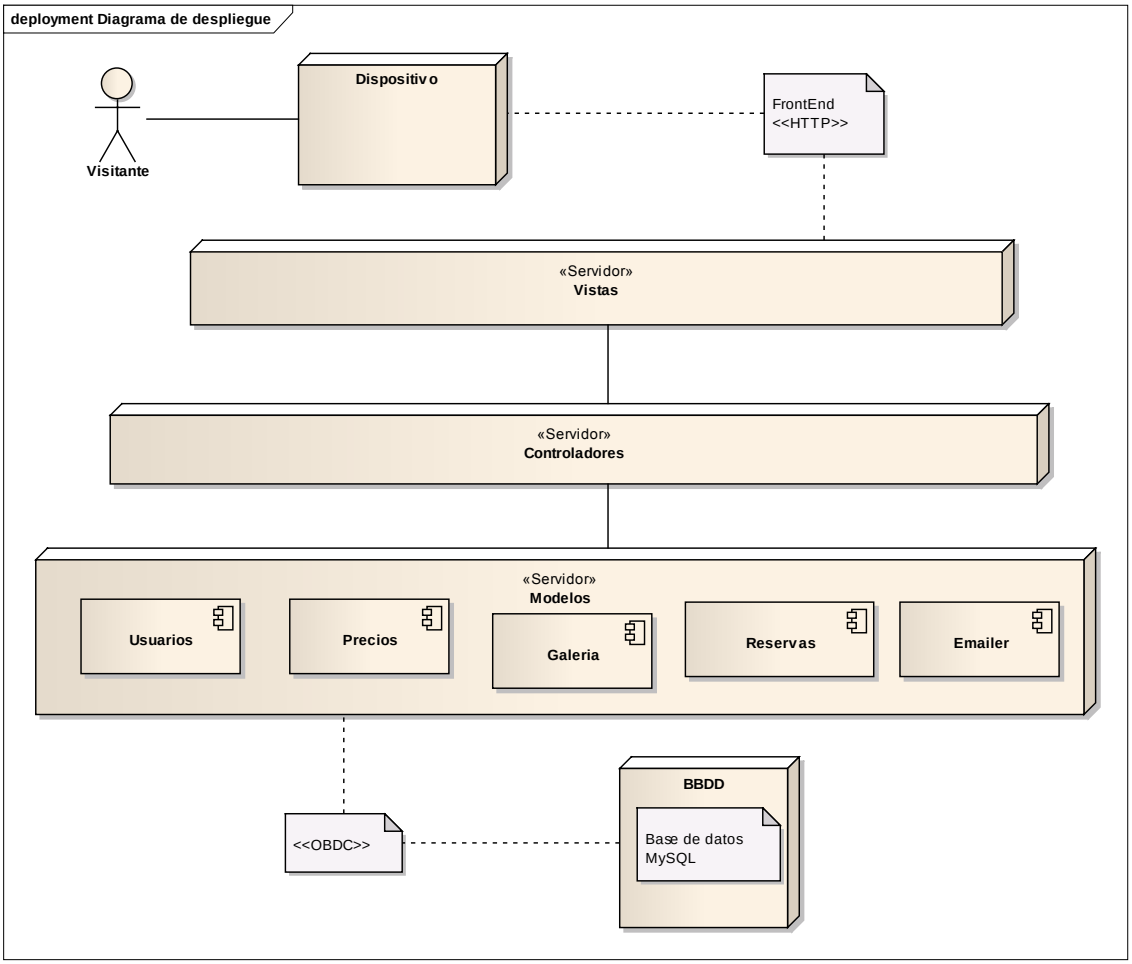


Figura 6.1. Diagrama de despliegue del sistema

El proyecto sigue la arquitectura **Modelo Vista Controlador**. La arquitectura MVC pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

El usuario interactúa con la **interfaz de usuario** a través de un dispositivo con acceso a internet.

El controlador recibe (por parte de los objetos de la interfaz-vista) **la notificación de la acción** solicitada por el usuario. El controlador gestiona el evento que llega.

El **controlador accede al modelo**, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza la fecha de una reserva).

El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado de las reservas). El modelo no tiene conocimiento directo sobre la vista.

La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Corresponde al **modelo gestionar los datos** que debe utilizar la aplicación

6.1.2.1 Vistas

Esta será la parte que verá el usuario de la aplicación. Aquí se representa la parte visual del programa, o lo que es lo mismo, la parte que usa el usuario. En este caso al tratarse de una aplicación web la vista está formada por HTML, CSS, JavaScript y contenido estático, como textos e imágenes. En la vista se representan todos los datos de la aplicación.

6.1.2.2 Controladores

Los controladores responden a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y a la vista. El enrutador nos examina nuestra petición HTTP y nos ayuda a determinar que se debe de hacer. La acción del controlador carga el modelo, librerías, *helpers*, *plugins* y todos los demás recursos necesarios para satisfacer nuestra petición.

6.1.2.3 Modelos

Representa la información del sistema. Trabaja junto a la vista para mostrar la información al usuario y es accedido por el controlador para añadir, eliminar, consultar o actualizar datos. También representa la forma de comportarse de los objetos.

6.1.2.4 BBDD

El acceso a la base de datos es totalmente abstracto desde el punto de vista del programador, es decir que es agnóstico a la base de datos, y Rails gestiona los accesos a la base de datos automáticamente (aunque, si se necesita, se pueden hacer consultas directas en SQL) Rails intenta mantener la neutralidad con respecto a la base de datos, la portatibilidad de la aplicación a diferentes sistemas de base de datos y la reutilización de bases de datos preexistentes.

6.2 Diseño de Modelos

6.2.1 Diagrama de Modelos

En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos.

A continuación muestro la estructura de modelos de la aplicación, este diagrama ha sido generado automáticamente a partir del código fuente del proyecto mediante la herramienta **RailRoady**. Es un generador de diagramas de clases para aplicaciones desarrolladas con Ruby on Rails. Se trata de un script Ruby que carga las clases de la aplicación y analiza sus propiedades y relaciones. Su salida es la descripción de un grafo en el lenguaje DOT.

Dado que *Ruby on Rails* es un *framework* de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en *ActiveRecord*, las definiciones de las clases no necesitan especificar los nombres de las columnas, Ruby puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante. Por lo general, lo único que tiene que hacer el programador es heredar de la clase *ActiveRecord::Base*, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

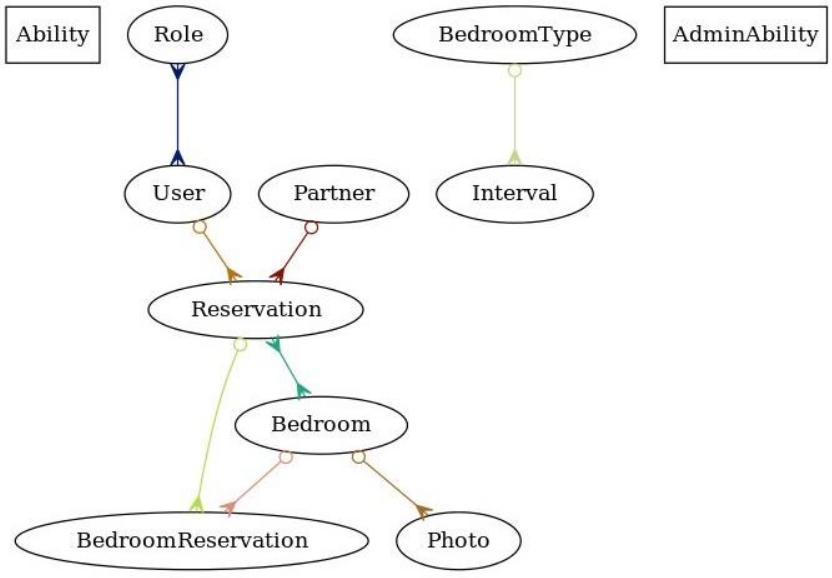


Figura 6.2. Diagrama de modelos

6.3 Diseño de la Base de Datos

6.3.1 Descripción del SGBD Usado

El sistema para la gestión de la base de datos utilizado en el proyecto ha sido *MySQL*, concretamente la versión 5.

Utilizo la herramienta **MySQL Workbench** para administrar la base de datos de la aplicación.

MySQL, es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. *MySQL AB* desarrolla *MySQL* como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la *GNU GPL* para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

Al contrario de proyectos como *Apache*, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet.

Existen varias APIs que permiten, a aplicaciones escritas en diversos lenguajes de programación, acceder a las bases de datos MySQL, incluyendo C, C++, C#, Pascal, Eiffel, Smalltalk, Java (con una implementación nativa del driver de Java), Lisp, Perl, PHP, Python, **Ruby**, Gambas, REALbasic (Mac y Linux), (x)Harbour (Eagle1), FreeBASIC, y Tcl; cada uno de estos utiliza una API específica. También existe una interfaz ODBC, llamado MyODBC que permite a cualquier lenguaje de programación que soporte ODBC comunicarse con las bases de datos MySQL. También se puede acceder desde el sistema SAP, lenguaje ABAP.

MySQL es muy utilizado en aplicaciones web, como Drupal o phpBB. Su popularidad como aplicación web está muy ligada a PHP, que a menudo aparece en combinación con MySQL.

MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional *MyISAM*, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

6.3.2 Integración del SGBD en Nuestro Sistema

El sistema consta de una base de datos MySQL llamada (al menos por el momento) "sobrefuentes". Esta base de datos está integrada por varias tablas correspondientes a los modelos de la aplicación descritos en apartados anteriores.

A continuación mostraré el fichero **schema.rb** que representa en código fuente la estructura de la base de datos. Los atributos de las tablas se corresponden con los atributos de los modelos, explicados en apartados anteriores:

```
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on another
# system, you should be using db:schema:load, not running all the migrations
# from scratch. The latter is a flawed and unsustainable approach (the more migrations
# you'll amass, the slower it'll run and the greater likelihood for issues).
#
# It's strongly recommended to check this file into your version control system.

ActiveRecord::Schema.define(:version => 20130622124006) do

  create_table "bedroom_reservations", :force => true do |t|
    t.integer  "reservation_id"
    t.integer  "bedroom_id"
    t.date     "assigned_at"
    t.datetime "created_at",                    :null => false
    t.datetime "updated_at",                    :null => false
    t.integer  "num_guest",      :default => 2
  end

  add_index "bedroom_reservations", ["bedroom_id"], :name =>
"index_bedroom_reservations_on_bedroom_id"
  add_index "bedroom_reservations", ["reservation_id"], :name =>
"index_bedroom_reservations_on_reservation_id"

  create_table "bedroom_types", :force => true do |t|
    t.string   "name",                  :default => ""
    t.integer  "price",                 :default => 60
    t.integer  "bedroom_id"
    t.datetime "created_at",                              :null => false
    t.datetime "updated_at",                              :null => false
    t.integer  "individual_difference", :default => 0
  end

  create_table "bedrooms", :force => true do |t|
    t.string   "name",                    :default => ""
    t.boolean  "supports_extra_bed",      :default => false
    t.boolean  "tv",                      :default => true
    t.boolean  "parking",                 :default => true
    t.boolean  "wifi",                    :default => true
    t.boolean  "bath",                    :default => true
    t.boolean  "breakfast",               :default => true
    t.boolean  "handicapped",             :default => false
    t.datetime "created_at",                                  :null => false
    t.datetime "updated_at",                                  :null => false
    t.integer  "bedroom_type_id"
    t.string   "main_photo_file_name"
    t.string   "main_photo_content_type"
    t.integer  "main_photo_file_size"
    t.datetime "main_photo_updated_at"
  end

  create_table "intervals", :force => true do |t|
    t.string   "name",            :default => ""
    t.integer  "price"
    t.integer  "bedroom_type_id"
    t.datetime "created_at",                     :null => false
```

```
    t.datetime "updated_at",                      :null => false
    t.date     "start"
    t.date     "end"
  end

  create_table "partners", :force => true do |t|
    t.string   "name"
    t.string   "website"
    t.integer  "telephone"
    t.string   "logo_file_name"
    t.string   "logo_content_type"
    t.integer  "logo_file_size"
    t.datetime "logo_updated_at"
    t.datetime "created_at",         :null => false
    t.datetime "updated_at",         :null => false
  end

  create_table "photos", :force => true do |t|
    t.string   "name"
    t.integer  "bedroom_id"
    t.string   "image_file_name"
    t.string   "image_content_type"
    t.integer  "image_file_size"
    t.datetime "image_updated_at"
    t.datetime "created_at",          :null => false
    t.datetime "updated_at",          :null => false
  end

  create_table "reservations", :force => true do |t|
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
    t.integer  "user_id"
    t.integer  "partner_id"
    t.date     "check_in"
    t.date     "check_out"
  end

  create_table "roles", :force => true do |t|
    t.string   "name"
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
  end

  create_table "roles_users", :id => false, :force => true do |t|
    t.integer "role_id"
    t.integer "user_id"
  end

  create_table "users", :force => true do |t|
    t.string   "email",                  :default => "", :null => false
    t.string   "encrypted_password",     :default => "", :null => false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          :default => 0
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
    t.datetime "created_at",                           :null => false
    t.datetime "updated_at",                           :null => false
    t.string   "confirm_email"
    t.string   "name"
    t.string   "telephone"
    t.string   "surname"
    t.string   "address"
    t.string   "city"
    t.integer  "post_code"
    t.string   "country"
    t.string   "username"
    t.string   "dni"
  end

  add_index "users", ["email"], :name => "index_users_on_email", :unique => true
  add_index "users", ["reset_password_token"], :name =>
"index_users_on_reset_password_token", :unique => true
```

```
end
```

6.4 Diseño de la Interfaz

Como se comentó en apartados anteriores, en el proyecto hay dos interfaces de usuario diferentes y bien definidas, por un lado lo que en conjunto representa el sitio web, sería la interfaz pública.

Por otra parte el panel de administración representaría la interfaz privada, desde la cual se pueden acceder a los diferentes subsistemas. El cliente, una vez registrado tiene acceso a la interfaz privada, pero sólo a su panel de usuario. A continuación se mostrará el diseño de algunas de las partes más representativas de la aplicación.

6.4.1 Parte pública

La página principal o índice, tendría una galería de fotografías con un texto descriptivo cada una. La galería es dinámica.

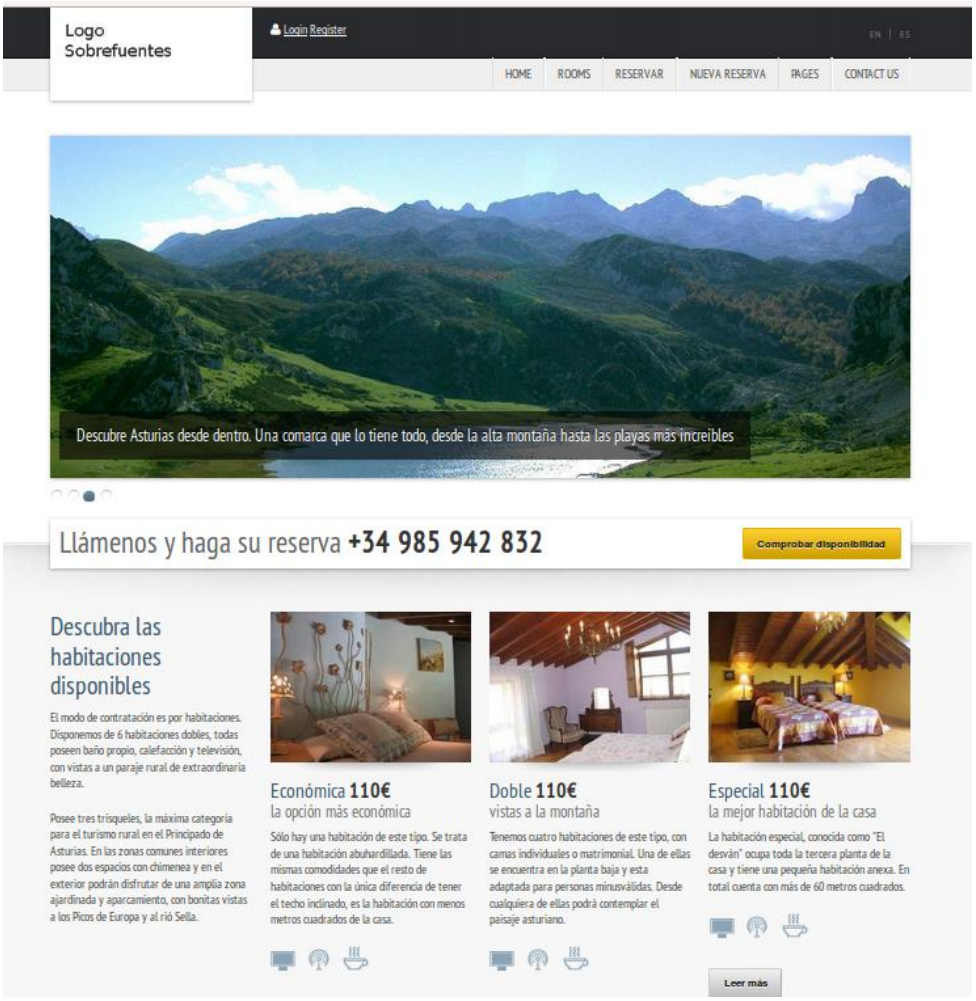


Figura 6.1 Diseño index

Arriba del todo se podrá seleccionar el idioma y acceder al *login* o registro en la web. Si el usuario está autenticado su cuenta aparecerá en la parte de arriba.

En función del ancho, los contenidos se adaptan a este. Por ejemplo, para uno ancho de 320px los contenidos se dispondrían de la siguiente manera:

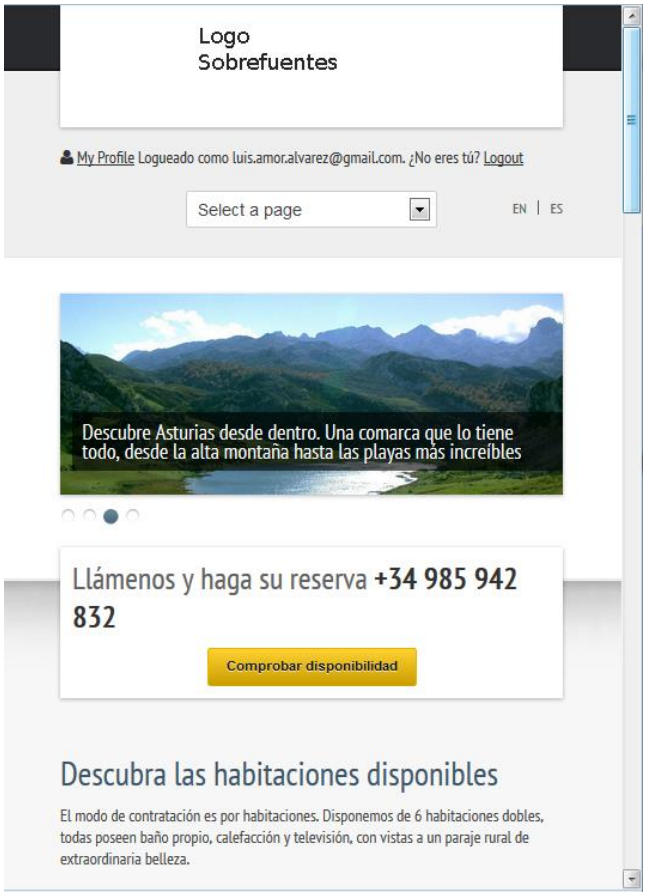


Figura 6.2 Diseño index versión móvil

El menú desplegable se sustituiría por un selector de página, la galería se adaptaría al ancho y las columnas se colocarían en vertical, en lugar de horizontal.

Cuando un visitante quiere formalizar una reserva, debe rellenar un formulario de 4 pasos, los campos introducidos en cada paso se validan tanto en el navegador como en el servidor:

1. En primer lugar, selecciona las fechas de la reserva con la ayuda de *datepickers*.
2. Una vez que el sistema conoce las fechas deseadas, muestra las habitaciones disponibles, así como sus precios. El cliente puede seleccionar el tipo y número que desee.

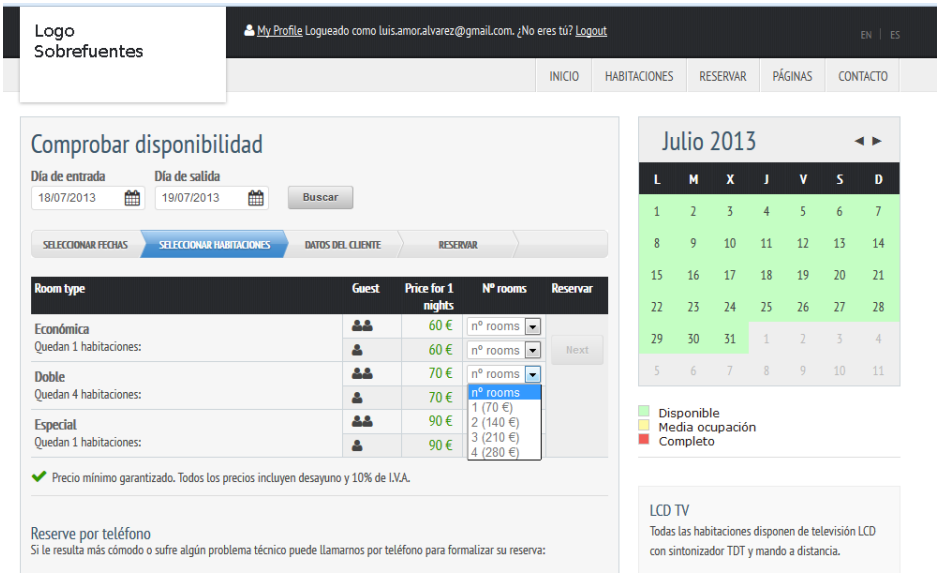


Figura 6.3 Diseño paso 2 reservas (habitaciones)

3. Una vez seleccionadas las habitaciones, se solicitan los datos al cliente en caso de que este no este logueado en el sistema:

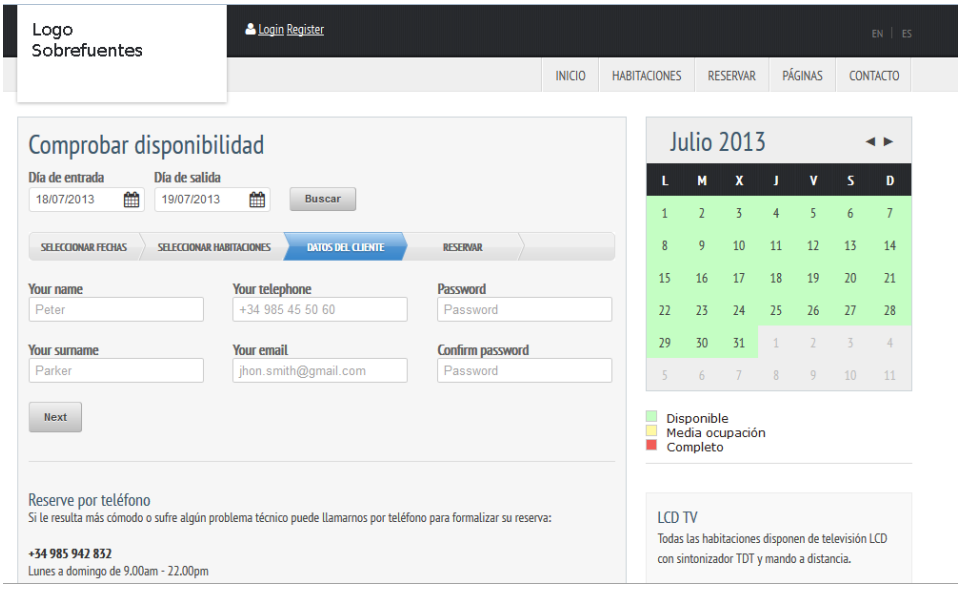


Figura 6.4 Diseño paso 3 reservas (datos usuario no logueado)

Si el usuario está autenticado en el sistema, sus datos aparecerán automáticamente en el formulario, junto a un hipervínculo a su panel de usuario, por si quisiera realizar alguna modificación de estos.

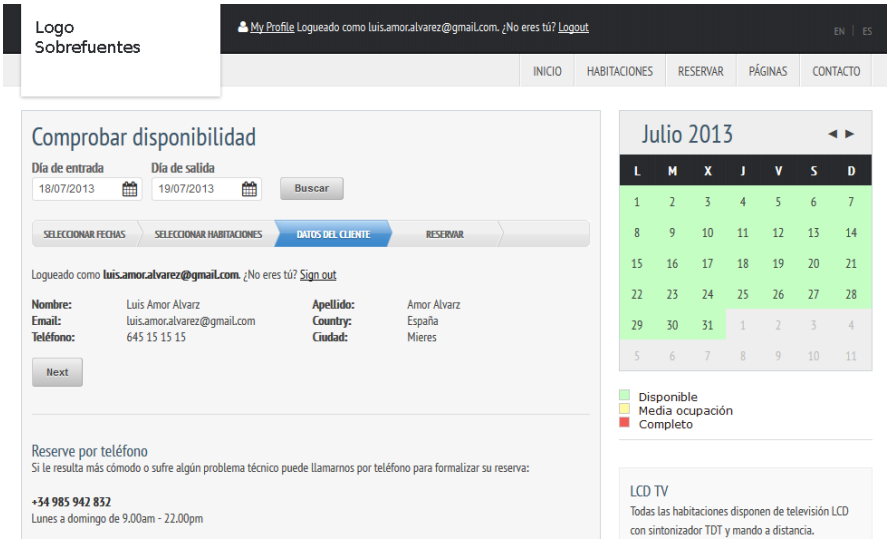


Figura 6.5 Diseño paso 3 reservas (datos usuario logueado)

4. En el último paso se muestra todos los datos introducidos anteriormente, para que el usuario de su confirmación final:

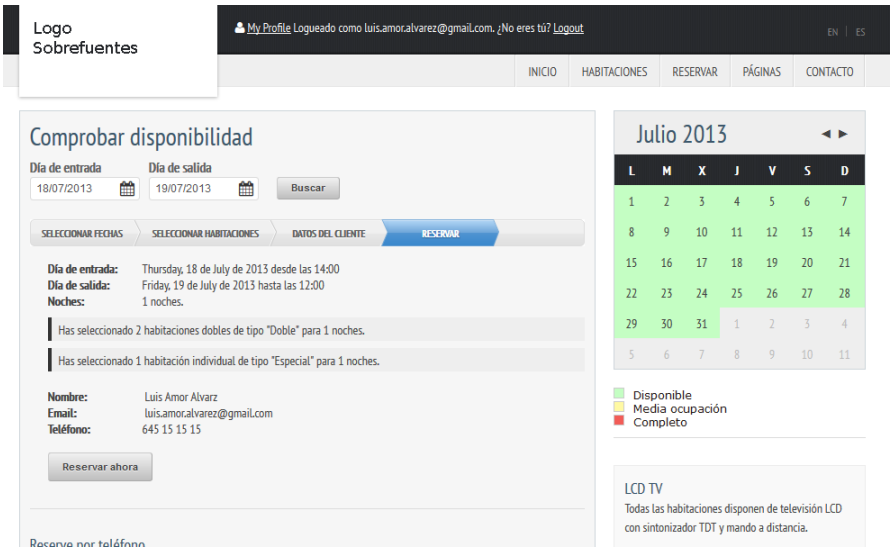


Figura 6.6 Diseño paso 4 reservas (confirmar reserva)

5. Una vez confirmada la reserva, se muestran todos los datos y se envía un correo electrónico tanto al administrador del establecimiento como al cliente, con los datos relativos a la reserva.

Figura 6.7 Diseño paso confirmar reserva

El formulario de inicio de sesión ha sido desarrollado con CSS3 puro y la fuente Awesome Font, sin uso de imágenes:

Figura 6.8 Diseño formulario inicio sesión

6.4.2 Panel de administración privado

A continuación se describe el panel de administración para las tareas de gestión por parte del administrador, y el panel "Mi cuenta" para los clientes registrados.

El *layout* consiste consta de: en la esquina superior izquierda hay un vínculo a la página principal de la parte pública, en la parte izquierda una columna con las visas disponibles, y en la parte superior, información sobre la sesión, un enlace para cerrar sesión y un selector de idioma.

El contenido propia de cada vista se muestra en el centro de la página, a la derecha de la columna izquierda.

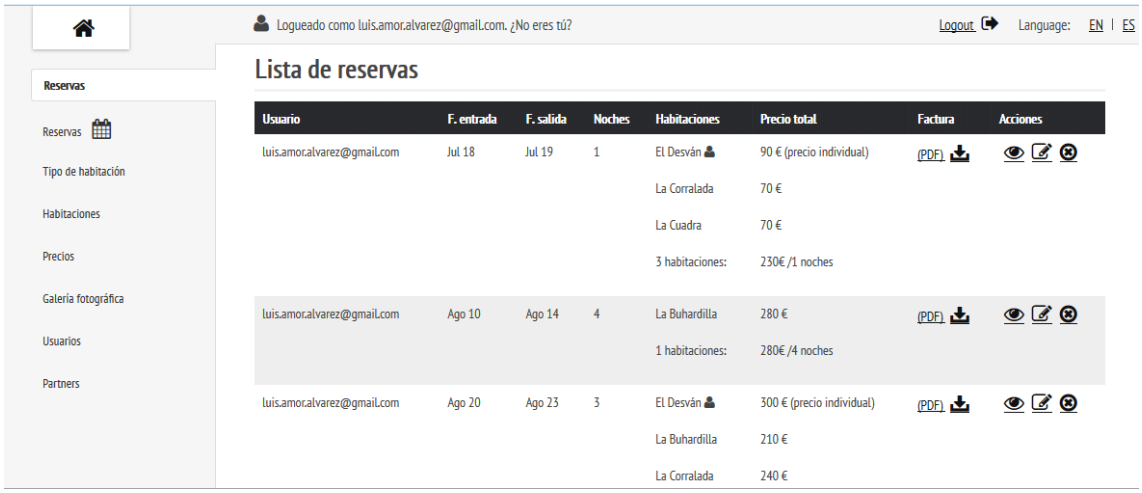


Figura 6.9 Diseño listado de reservas

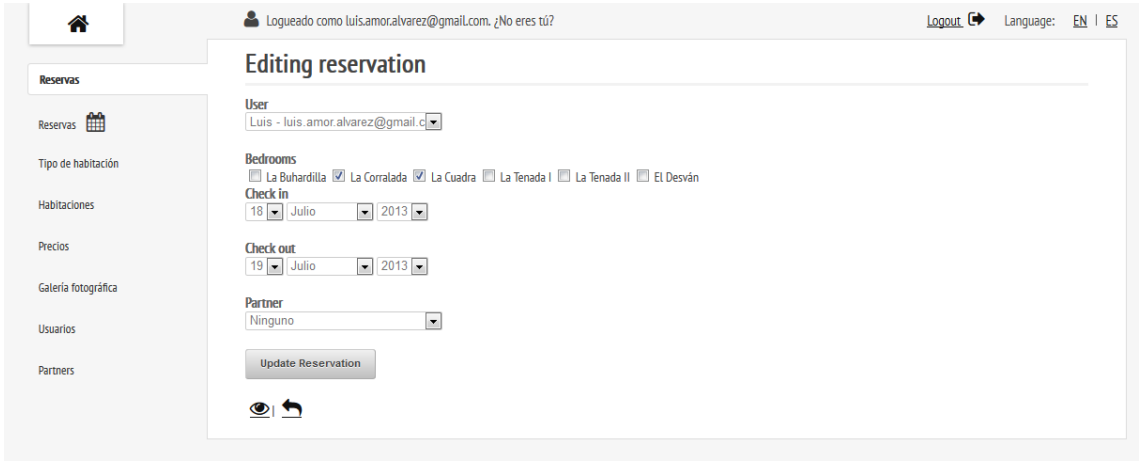


Figura 6.10 Diseño edición de reserva

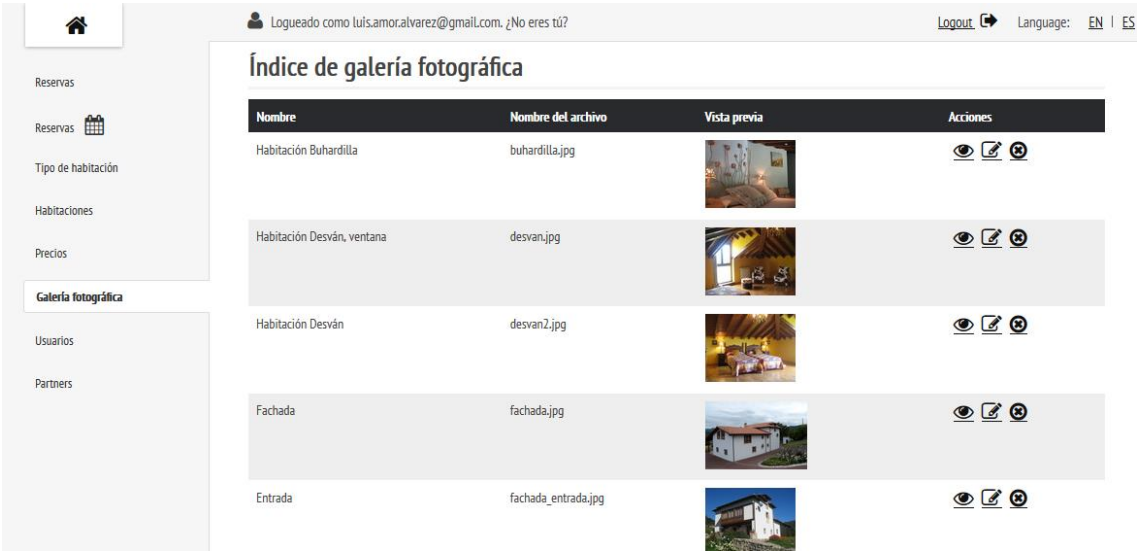


Figura 6.11 Diseño índice de galería fotográfica

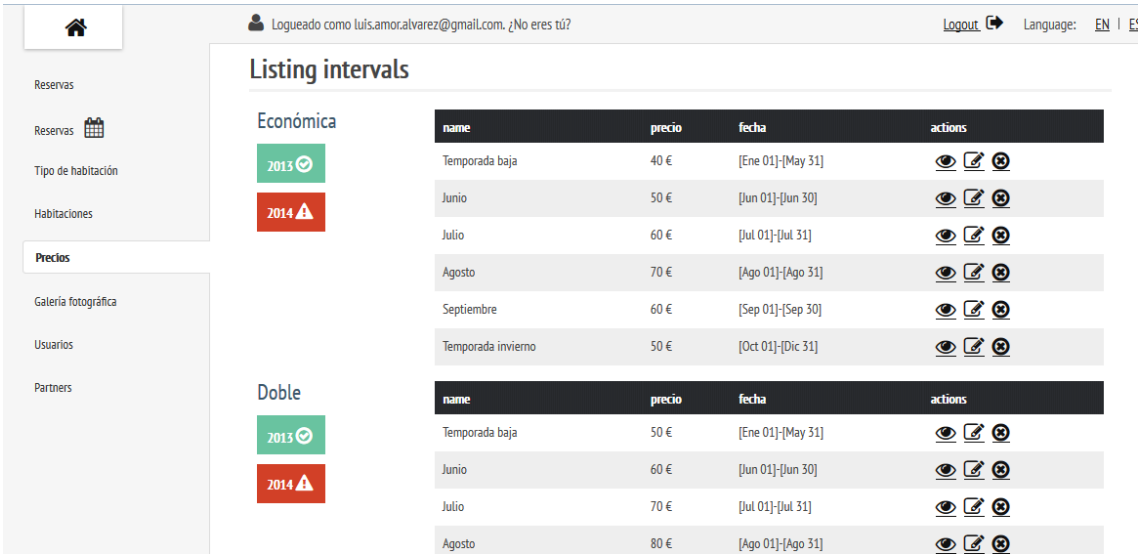


Figura 6.12 Diseño listado de intervalos de precios

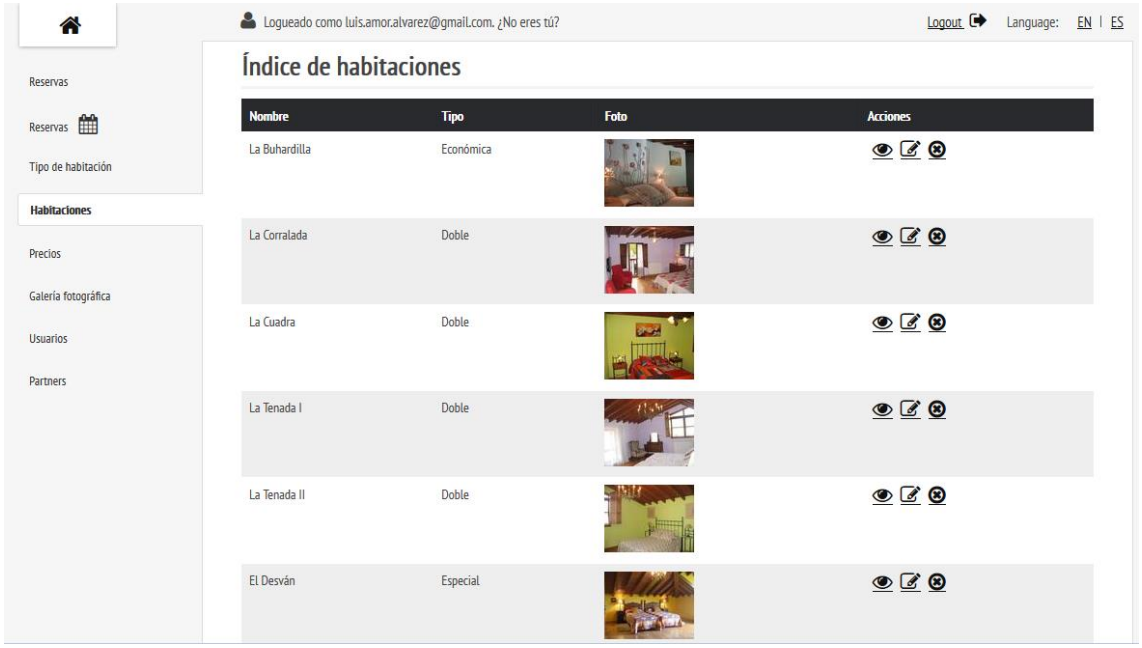


Figura 6.13 Diseño índice de habitaciones

El cliente tendría acceso sólo al panel "Mi cuenta", desde esta vista puede ver sus datos, sus reservas y editar su perfil

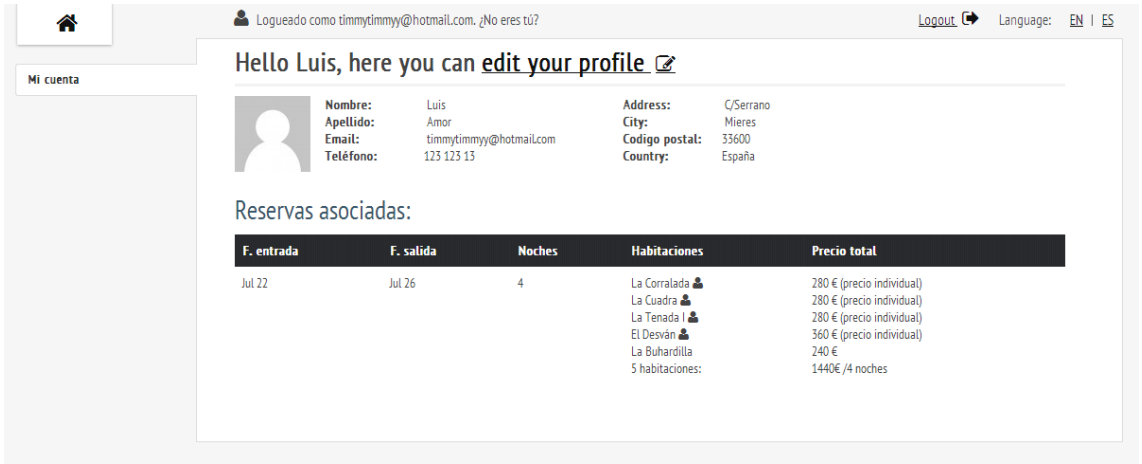


Figura 6.14 Diseño de vista "Mi Cuenta"

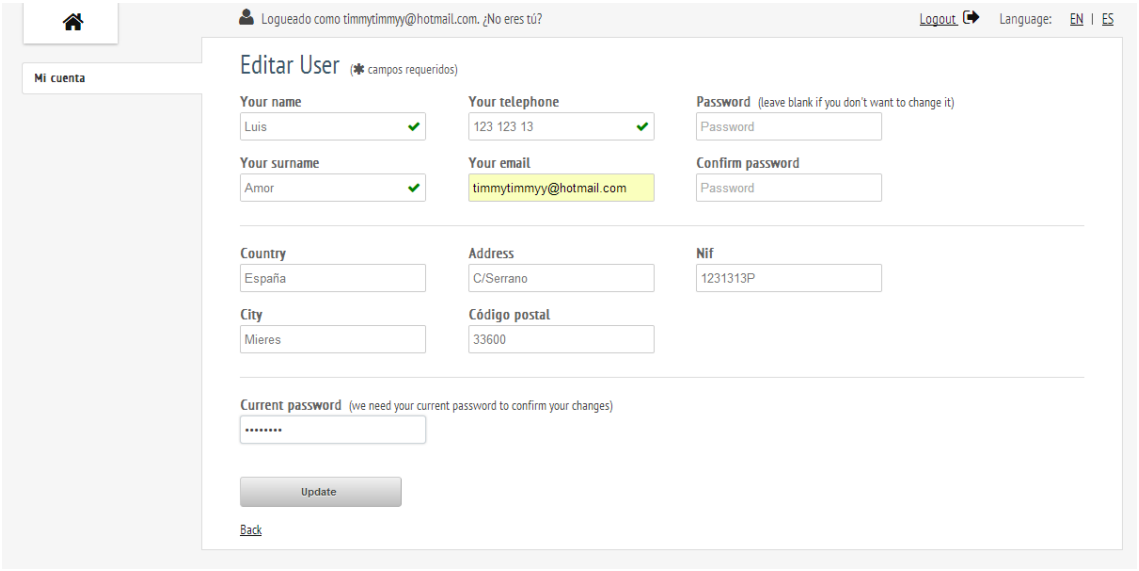


Figura 6.15 Diseño de edición de perfil

Si un usuario intenta acceder a una sección a la cual no está autorizado, se mostrará un mensaje de error:

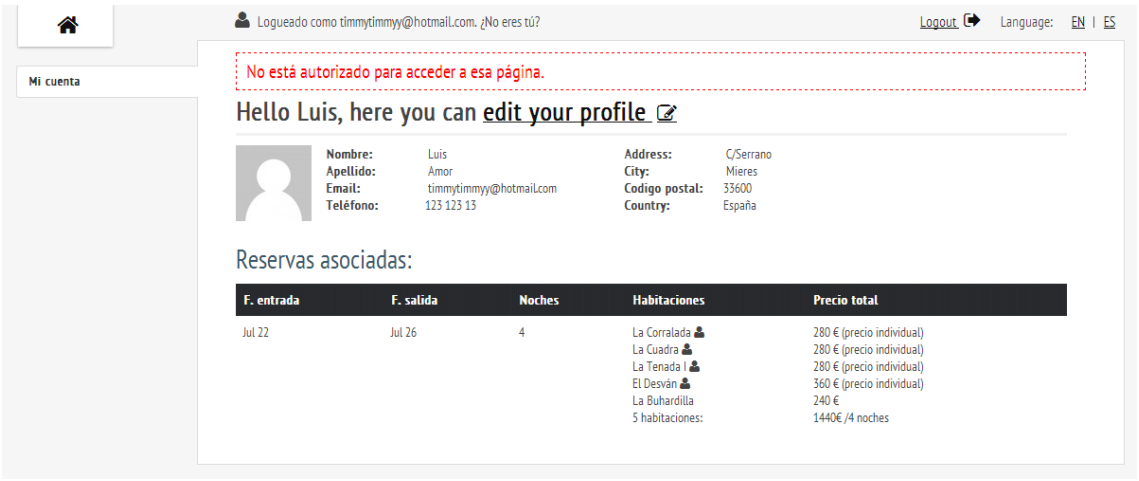


Figura 6.16 Diseño error de acceso

6.5 Especificación Técnica del Plan de Pruebas

Las pruebas de software se han llevado a cabo durante la construcción del mismo para intentar entregar el software con los mínimos fallos posibles.

Para que funcione el desarrollo guiado por pruebas, el sistema que se programa tiene que ser lo suficientemente flexible como para permitir que sea probado automáticamente.

6.5.1 Pruebas Unitarias

Para el desarrollo de las pruebas unitarias se ha utilizado la librería de pruebas **Rspec** para Ruby on Rails. Para algunas funcionalidades de los modelos, especialmente las reservas se ha utilizado un desarrollo guiado por pruebas, también conocido como desarrollo TDD (*Test Driven Development*). Cada prueba será suficientemente pequeña como para que permita determinar unívocamente si el código probado pasa o no la verificación que ésta le impone. El diseño se ve favorecido ya que se evita el indeseado "sobre diseño" de las aplicaciones y se logran interfaces más claras y un código más cohesivo.

Se ha creado una batería de pruebas la cual nos permite saber si al introducir un cambio en un modelo este puede afectar a la funcionalidad del mismo. De este modo el software gana en consistencia al comprobarse que los cambios no estropean el resultado esperado. Es una forma sencilla de encontrar fallos (o casos no detectados que inducen a error) durante la implementación de una modelo.

Los pasos a seguir para el desarrollo de las pruebas unitarias han sido:

1. Diseñar la prueba unitaria para una función de un modelo.
2. Implementar la prueba unitaria para esa función.
3. Implementar la funcionalidad en el modelo.
4. Ejecutar la batería de pruebas.
5. Arreglar posibles fallos.
6. Si hubo fallos, volver a modificar la lógica y volver pasar la batería de pruebas hasta que todo funcione correctamente.

Para este proyecto se han desarrollado clases de test con pruebas unitarias para las funciones más relevantes de la aplicación.

6.5.2 Pruebas de Integración y del Sistema

Las pruebas de integración y sistema se harán al terminar cada módulo. Se verifica que se cumple un requisito y que el sistema es estable y no se han introducido cambios que lleven a errores. Una vez que se han implementado todos los requisitos de un incremento del proceso software se hacen pruebas.

En caso de detectar algún fallo se procederá a solventarlo. De este modo, seguramente el cliente no se encuentre situaciones que no se habían planteado desde un principio y el cliente obtendrá una mayor satisfacción en la versión final del software.

Para la realización de este tipo de pruebas, debido a la naturaleza de nuestro proyecto, se utilizará el navegador web.

6.5.3 Pruebas de Usabilidad y Accesibilidad

Estas pruebas no se han realizado hasta la fase final del desarrollo. De todos modos, se ha intentado intercambiar opiniones sobre la interfaz con usuarios del sector de turismo para ir diseñándola acorde a las necesidades según se fueran dando. Se ha dedicado una pequeña parte del tiempo de desarrollo del proyecto a aplicar conceptos de usabilidad y accesibilidad en el software. Para ello se han realizado unos cuestionarios que se detallan a continuación.

6.5.3.1 Cuestionario de Evaluación

Elaboramos un cuestionario para que los usuarios evalúen distintos aspectos del proyecto y de su interacción este. Los puntos a tocar son:

6.5.3.1.1 Preguntas de carácter general

Se muestra un esbozo de un posible cuestionario, que debemos desarrollar y adaptar a nuestras necesidades:

¿Usa un ordenador frecuentemente?
1. Todos los días 2. Varias veces a la semana 3. Ocasionalmente 4. Nunca o casi nunca
¿Qué tipo de actividades realiza con el ordenador?
1. Es parte de mi trabajo o profesión 2. Lo uso básicamente para ocio 3. Solo empleo aplicaciones estilo Office 4. Únicamente leo el correo y navego ocasionalmente
¿Ha usado alguna vez software como el de esta prueba?
1. Sí, he empleado software similar 2. No, aunque si empleo otros programas que me ayudan a realizar tareas similares 3. No, nunca
¿Qué busca Vd. Principalmente en un programa?
1. Que sea fácil de usar 2. Que sea intuitivo

3. Que sea rápido
4. Que tenga todas las funciones necesarias

6.5.3.1.2 Actividades guiadas

Las actividades guiadas se realizaron al finalizar el proyecto. El proceso es el siguiente: se le explica a un usuario brevemente que hace el sistema, y que tiene que hacer para ponerlo a funcionar, entonces se le deja que use solo, mientras se le va observando y apuntando las sugerencias que haga o en caso de que se no sepa que hacer se apuntará donde sucedió, finalmente se intentará arreglar esas situaciones recogidas en el informe para una siguiente versión, que se volverá a probar por este mismo proceso hasta que el usuario quede contento con el resultado.

6.5.3.1.3 Preguntas Cortas sobre la Aplicación y Observaciones

Facilidad de Uso	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Sabe dónde está dentro de la aplicación?				
¿Existe ayuda para las funciones en caso de que tenga dudas?				
¿Le resulta sencillo el uso de la aplicación?				
Funcionalidad	**Siempre**	**Frecuentemente**	**Ocasionalmente**	**Nunca**
¿Funciona cada tarea como Vd. Espera?				
¿El tiempo de respuesta de la aplicación es muy grande?				
Calidad del Interfaz				
Aspectos gráficos	**Muy Adecuado**	**Adecuado**	**Poco Adecuado**	**Nada Adecuado**
El tipo y tamaño de letra es				
Los iconos e imágenes usados son				
Los colores empleados son				
Diseño de la Interfaz		**Si**	**No**	**A veces**
¿Le resulta fácil de usar?				
¿El diseño de las pantallas es claro y atractivo?				
¿Cree que el programa está bien estructurado?				
Observaciones				
Cualquier comentario del usuario				

Al evaluar la parte concerniente al panel privado del administrador (sólo visible por el administrador), la usabilidad y accesibilidad no han tenido tanto peso como otros aspectos, si bien se ha intentado hacer el sistema lo más usable y accesible posible.

6.5.4 Pruebas de Rendimiento

Para realizar las pruebas de rendimiento se utilizará el complemento de Google ***PageSpeed***, https://developers.google.com/speed/pagespeed concretamente la extensión para Firefox que se integra en el *Firebug*.

Puede descargarse en: https://developers.google.com/speed/docs/insights/using_firefox

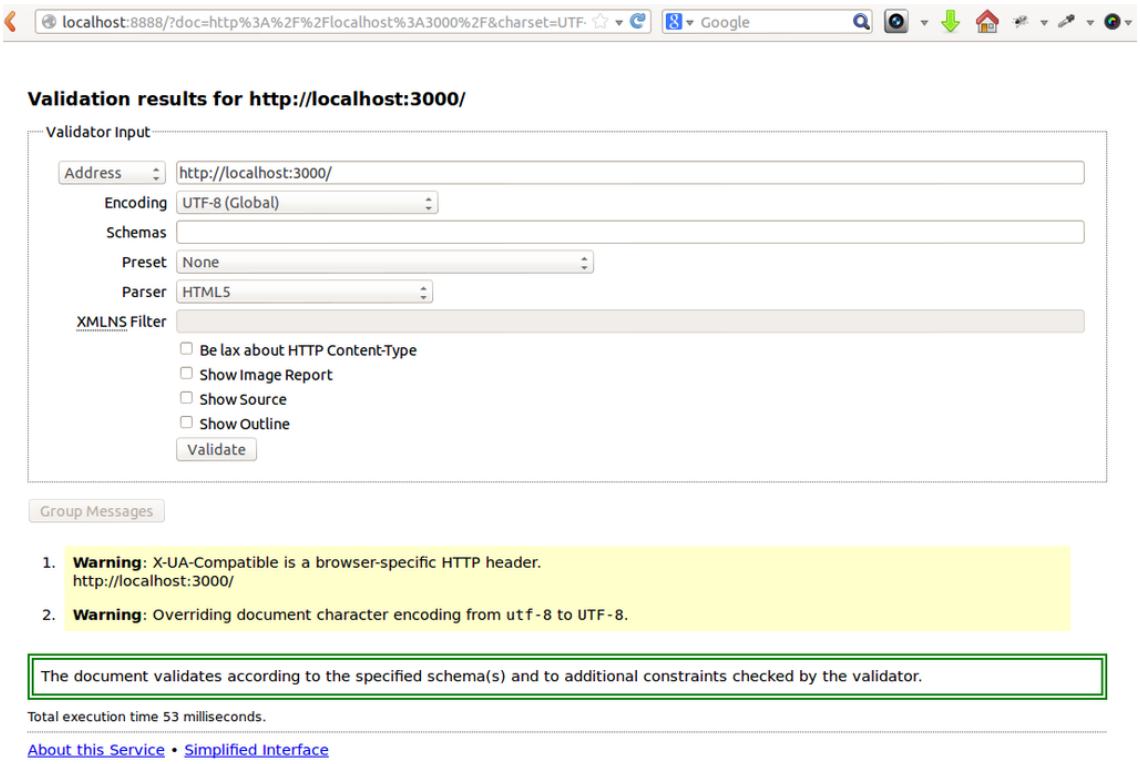
Capítulo 7. Implementación del Sistema

7.1 Estándares y Normas Seguidos

Para el desarrollo de la capa de presentación se ha seguido el estándar W3C para el código HTML. Para su validación y corrección se ha instalado el w3c-walidator en el servidor local de desarrollo, ya que al no estar el proyecto subido a un servidor real, no es posible utilizar el proporcionado por http://validator.w3.org.

Para la instalación se ha seguido el tutorial: http://blog.simplytestable.com/installing-the-w3c-html-validator-with-html5-support-on-ubuntu incluyendo la instalación para que soporta validación HTML5 en local, ya que el sitio web utiliza HTML5.

El resultado:



7.2 Lenguajes de Programación

Para el desarrollo del proyecto se han utilizado los siguientes lenguajes:

- Framework Ruby on Rails
- JavaScript (JSON)
- HTML5
- CSS3 (SASS)

Además se ha utilizado una base de datos SQL (MySQL).

7.2.1 Ruby on Rails

Para el desarrollo de la aplicación se ha utilizado Ruby 1.9.2 y Rails 3.2

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

Los principios fundamentales de Ruby on Rails incluyen No te repitas (del inglés Don't repeat yourself, **DRY**) y Convención sobre configuración.

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

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

En las aplicaciones web orientadas a objetos sobre bases de datos, el **Modelo** consiste en las clases que representan a las tablas de la base de datos.

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

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

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

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

En MVC, las clases del **controlador** responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web.

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general en la forma http://aplicacion/ejemplo/metodo, que invoca a EjemploController#método, y presenta los datos usando el archivo de plantilla /app/views/ejemplo/método.html.erb, a no ser que el método redirija a algún otro lugar.

7.2.2 HTML5

HTML, siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML se escribe en forma de «etiquetas», rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo Javascript), el cual

puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

HTML5 no es simplemente una nueva versión del lenguaje de marcación HTML, sino una agrupación de diversas especificaciones concernientes al desarrollo web. Es decir, HTML5 no se limita sólo a crear nuevas etiquetas, atributos y eliminar aquellas marcas que están en desuso o se utilizan inadecuadamente, sino que va mucho más allá.

- **Estructura del cuerpo**: La mayoría de las webs tienen un formato común, formado por elementos como cabecera, pie, navegadores, etc. HTML 5 permite agrupar todas estas partes de una web en nuevas etiquetas que representarán cada uno de las partes típicas de una página.
- **Etiquetas para contenido específico**: Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.
- **Canvas**: es un nuevo componente que permitirá dibujar, por medio de las funciones de un API, en la página todo tipo de formas, que podrán estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece Flash, pero dentro de la especificación del HTML y sin la necesidad de tener instalado ningún plugin. Puedes conocer más sobre este nuevo elemento en el manual de canvas que estamos creando en DesarrolloWeb.com
- **Bases de datos locales**: el navegador permitirá el uso de una base de datos local, con la que se podrá trabajar en una página web por medio del cliente y a través de un API. Es algo así como las Cookies, pero pensadas para almacenar grandes cantidades de información, lo que permitirá la creación de aplicaciones web que funcionen sin necesidad de estar conectados a Internet.
- **Web Workers**: son procesos que requieren bastante tiempo de procesamiento por parte del navegador, pero que se podrán realizar en un segundo plano, para que el usuario no tenga que esperar que se terminen para empezar a usar la página. Para ello se dispondrá también de un API para el trabajo con los Web Workers.
- Aplicaciones web Offline: Existirá otro API para el trabajo con aplicaciones web, que se podrán desarrollar de modo que funcionen también en local y sin estar conectados a Internet.
- **Geolocalización**: Las páginas web se podrán localizar geográficamente por medio de un API que permita la Geolocalización.
- **Nuevas APIs para interfaz de usuario**: temas tan utilizados como el "drag & drop" (arrastrar y soltar) en las interfaces de usuario de los programas convencionales, serán incorporadas al HTML 5 por medio de un API.
- **Fin de las etiquetas de presentación**: todas las etiquetas que tienen que ver con la presentación del documento, es decir, que modifican estilos de la página, serán eliminadas. La responsabilidad de definir el aspecto de una web correrá a cargo únicamente de CSS.

7.2.3 CSS3

Las hojas de estilo en cascada (en inglés Cascading Style Sheets), CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

Por ejemplo, el elemento de HTML <h1> indica que un bloque de texto es un encabezamiento y que es más importante que un bloque etiquetado como <H2>. Versiones más antiguas de HTML permitían atributos extra dentro de la etiqueta abierta para darle formato (como el color o el tamaño de fuente). No obstante, cada etiqueta <H1> debía disponer de la información si se deseaba un diseño consistente para una página y, además, una persona que leía esa página con un navegador perdía totalmente el control sobre la visualización del texto.

Cuando se utiliza CSS, la etiqueta <H1> no debería proporcionar información sobre cómo será visualizado, solamente marca la estructura del documento. La información de estilo, separada en una hoja de estilo, específica cómo se ha de mostrar <H1>: color, fuente, alineación del texto, tamaño y otras características no visuales, como definir el volumen de un sintetizador de voz, por ejemplo.

La información de estilo puede ser adjuntada como un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo "style".

La especificación de CSS3 viene con interesantes novedades que permitirán hacer webs más elaboradas y más dinámicas, con mayor separación entre estilos y contenidos. Dará soporte a muchas necesidades de las webs actuales, sin tener que recurrir a trucos de diseñadores o lenguajes de programación como JavaScript.

7.2.4 SASS

SASS (Syntactically Awesome Stylesheets) es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es interpretado a CSS.

Sass consiste en dos sintaxis. La sintaxis original, usa la indentación para separar bloques de código y el carácter nueva línea para separar reglas.

La sintaxis más nueva, SCSS usa el formato de bloques como CSS. Este usa llaves para denotar bloques de código y punto y coma (;) para separar las líneas dentro de un bloque. La sintaxis indentada y los ficheros SCSS tienen las extensiones .sass y .scss respectivamente.

La implementación oficial de Sass es software libre y está escrita en Ruby, sin embargo existen otras implementaciones. Es un metalenguaje anidado, lo que es válido en CSS es válido en SCSS con la misma semántica.

7.2.5 JavaScript (jQuery)

En gran cantidad de páginas del sitio web se utiliza JavaScript para enriquecer la interfaz de usuario y dotarla de nuevas funcionalidades, para facilitar esta tarea se utiliza la biblioteca jQuery.

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript

se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

jQuery es una biblioteca o framework de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

JQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. JQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

7.2.6 SQL (MySQL)

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés structured query language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre ella.

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. *MySQL AB* (desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009) desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

Se utiliza la herramienta *MySQL Workbench* para administrar la base de datos de la aplicación.

7.3 Librerías utilizadas

Para el desarrollo de la aplicación se han utilizado librerías tanto en Ruby, para ayudar al desarrollo de la lógica del programa como en *JavaScript*, para enriquecer su interfaz de usuario. A continuación enumeraré las librerías más relevantes utilizadas, así como una breve descripción de su utilidad.

7.3.1 Gemas (librerías Ruby)

Las gemas son *plugins* o librerías para Ruby on Rails, que al agregarlas a nuestros proyectos permiten nuevas funcionalidades tras una correcta integración a nuestro sistema y sus necesidades.

Para encontrar el listado de gemas disponibles puedes ir a ***RubyForge***. Las gemas que utilizaremos en la aplicación, se declaran en el fichero ***Gemfile,*** en la raíz del proyecto. Para su instalación o desinstalación debemos agregarlas o eliminarlas del GEMfile, y luego ejecutar la orden:

```
bundle install
```

Esto hace que *bundle* busque las versiones adecuadas para las gemas, en relación con nuestra versión de Rails.

7.3.1.1 Devise

Devise gestiona la **autenticación** a todos los niveles, cubre además de los modelos, también vistas y controladores.

La arquitectura de Devise es modular y consta de once módulos cada uno de los cuales cubre un aspecto diferente de la autenticación. Por ejemplo el módulo *Rememberable* recuerda la autenticación del usuario en una cookie mientras que otro módulo, *Recoverable*, se ocupa de reiniciar la clave del usuario enviando instrucciones por correo. Este enfoque hace que sea muy fácil escoger exactamente las funcionalidades de autenticación que queramos utilizar.

Fuente: https://github.com/plataformatec/devise

7.3.1.2 CanCan

Con esta gema gestionaremos la **autorización** que tienen los usuarios para realizar acciones, una vez autenticados en el sistema.

CanCan es una biblioteca de autorización para Ruby on Rails, restringe lo que los recursos de un usuario dado está autorizado a acceder. Todos los permisos se definen en un solo lugar, la clase de capacidad (*Ability*) y no se duplican entre controladores, vistas y consultas de bases de datos.

Fuente: https://github.com/ryanb/cancan

7.3.1.3 Mysql2

La gema **mysql2** sirve para establecer la conexión de la aplicación con la base de datos MySQL, hacer consultas y la iteración en los resultados.

También obliga a la utilización de UTF-para la conexión y utiliza la API MySQL

El API se compone de dos clases:

- Client: la conexión con la base de datos.
- Result: devuelve los resultados de las consultas a la base de datos.

Fuente: https://github.com/brianmario/mysql2

7.3.1.4 Hirb

Hirb es un miniframework que nos ayuda a la visualización de datos cuando utilizamos la consola de Rails. Al realizar una consulta a nuestra base de datos, por defecto la consola nos devuelve el objeto con este formato:

```
>> Tag.all :limit=>3, :order=>"id DESC"
=> [#<Tag id: 907, name: "gem:tags=yaml", description: nil, created_at: "2009-03-06
21:10:41",
namespace: "gem", predicate: "tags", value: "yaml">, #<Tag id: 906, name:
"gem:tags=nomonkey",
description: nil, created_at: "2009-03-06 08:47:04", namespace: "gem", predicate:
"tags", value:
"nomonkey">, #<Tag id: 905, name: "article:tags=ruby", description: nil, created_at:
"2009-03-04
00:30:10", namespace: "article", predicate: "tags", value: "ruby">]
```

Lo cual resulta incómodo de leer, activando Hirb la salida sería algo así:

```
>> Tag.all :limit=>3, :order=>"id DESC"
+-----+--------------------------+-------------+------------------+-----------+-----------+----------
+
| id  | created_at               | description | name             | namespace | predicate | value
|
+-----+--------------------------+-------------+------------------+-----------+-----------+----------
+
| 907 | 2009-03-06 21:10:41 UTC |             | gem:tags=yaml    | gem       | tags      | yaml
|
| 906 | 2009-03-06 08:47:04 UTC |             | gem:tags=nomonkey | gem       | tags      | nomonkey
|
| 905 | 2009-03-04 00:30:10 UTC |             | article:tags=ruby | article   | tags      | ruby
|
+-----+--------------------------+-------------+------------------+-----------+-----------+----------
+
3 rows in set
```

Mucho más visual y cómodo.

Fuente: https://github.com/cldwalker/hirb

7.3.1.5 Paperclip

Paperclip pretende ser una sencilla biblioteca de adjuntar archivos en varios formatos. Gestiona validaciones basadas en el tamaño y presencia, si es necesario. Puede transformar la imagen asignada en varios tamaños al subirla al sistema. *Paperclip* utiliza por debajo la ***ImageMagick***, es necesario tener instalado este paquete en el disco duro. Los archivos adjuntos se guardan en el sistema de archivos, en la carpeta */public/system* y se hace referencia a estos desde la vista de manera sencilla, accediendo directamente al tamaño o formato deseado.

Fuente: https://github.com/thoughtbot/paperclip

7.3.1.6 Gravatar image tag

En el panel de control del cliente se mostrará su avatar si su correo electrónico tiene uno asociado. Es una librería muy sencilla, que nos permite acceder directamente al servicio de ***Gravatar*** desde nuestras vistas.

Gravatar (una abreviación para *Gobally Recognized Avatar* en inglés, o avatar reconocido globalmente, en español) es un servicio muy popular que ofrece un avatar único globalmente.

En Gravatar, los usuarios pueden registrar una cuenta basada en su correo electrónico, y subir un avatar para que sea asociado con la cuenta. Los Complementos de *Gravatar* para blogs se encuentran disponibles para los sistemas de blog más populares; cuando el usuario publica un comentario en dicho blog que requiere un correo electrónico, el software del blog revisa si ese correo se encuentra asociado a una cuenta de *Gravatar*. Si es así, el *Gravatar* se muestra junto con el comentario. El soporte para *Gravatar* se encuentra disponible nativamente a partir de la versión 2.5 en *Wordpress*. Soporte para los *Gravatars* también es incluido a través de complementos en *Drupal*.

Fuente: https://github.com/mdeering/gravatar_image_tag

7.3.1.7 Rspec rails

Rspec-rails es un *framework* de *testing* para *Rails* 3. En el apartado 8.1 (pruebas unitarias) se explica su uso en detalle.

Fuente: https://github.com/rspec/rspec-rails

7.3.1.8 Factory Girl

Es una extensión de Rspec, se integra en los test desarrollados en el *framework* de *testing* facilitando el manejo de objetos en los test.

Fuente: https://github.com/thoughtbot/factory_girl_rails

7.3.1.9 Sass Rails

Integra el lenguaje **Sass** en el proyecto Ruby on Rails. Compila automáticamente el css generado e integra opciones de configuración en el proyecto.

Fuente: https://github.com/rails/sass-rails

7.3.1.10 Configatron

Configatron nos permite definir variables globales en la configuración del proyecto. Una vez definidas, estas serán accesibles desde cualquier modelo, controlador o vista de la aplicación. Esto resulta muy útil para no repetir código y facilitar futuros cambios en la aplicación.

Por ejemplo, puedo declarar el correo electrónico del administrador de la aplicación en el application.rb:

```
  configatron.email = "luis.amor.alvarez@gmail.com"
```

Ahora desde cualquier parte de la aplicación, si accedo a

```
  configatron.email
```

Me devolvería "luis.amor.alvarez@gmail.com". Es sencillo pero muy útil.

Fuente: https://github.com/markbates/configatron

7.3.1.11 RailRoady

Con esta gema genero los diagramas de modelos y controladores a partir del código del proyecto. Es una actualización de la gema *RailRoad* para *Ruby* 1.9 y *Rails* 3.

Una vez instalada, desde el directorio del proyecto puedo ejecutar los comandos:

```
railroady -o models.dot -M
  Crea un diagrama de modelos en el archivo 'models.dot'
railroady -a -i -o full_models.dot -M
  Diagrama de modelos con todas las clases y relaciones
railroady -M | dot -Tsvg > models.svg
  Diagrama de modelos en formato SVG
railroady -C | neato -Tpng > controllers.png
  Diagrama de controladores en formato PNG
railroady -h
  Muestra la ayuda
```

Fuentes: http://railroady.prestonlee.com, https://github.com/preston/railroady

7.3.1.12 Wicked_PDF

Con la ayuda de esta librería **genero un archivo PDF** con la información necesaria para la **factura** asociada a cada reserva. Se desglosa el precio por habitación, los impuestos y se incluyen los datos el cliente y los datos fiscales del establecimiento.

Esta librería utiliza por debajo el paquete ***wkhtmltopdf.*** Es necesario tenerlo previamente instalado en el servidor para que funcione correctamente.

Esta gema permite, a partir de una vista en HTML, transformar a PDF el contenido respetando los estilos definidos en la vista HTML.

Fuente: https://github.com/mileszs/wicked_pdf

7.3.2 *Plugins* JavaScript

En el proyecto se integran las siguientes librerías JavaScript:

7.3.2.1 JQuery

jQuery es una biblioteca de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006. JQuery es la biblioteca de JavaScript más utilizada en la actualidad.

Muchos de los *plugins* integrados en el proyecto, necesitan tener preinstalada esta librería para su correcto funcionamiento.

JQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. JQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. De ahí que su lema sea “Write less, do more” (escribe menos haz más).

Fuentes: http://es.wikipedia.org/wiki/JQuery , http://api.jquery.com

7.3.2.2 JQuery form

Los diferentes pasos del formulario de reservas se hacen con AJAX utilizando esta librería.

Este *plugin* permite actualizar fácilmente y de forma discreta formularios HTML mediante la técnica *AJAX*.

Los métodos principales, *ajaxForm* y *ajaxSubmit*, recopilan información desde el formulario para determinar cómo manejar el proceso de envío. Ambos métodos soportan opciones que permiten tener un control total sobre cómo se envían los datos.

Fuente: http://malsup.com/jquery/form/

7.3.2.3 JQuery tools

Es una popular colección de elementos de interacción interfaz-usuario mediante Javascript.

Utilizo esta librería para mostrar en la vista *tooltips* y *datepickers*.

Fuente: http://jquerytools.org

7.3.2.4 Full Calendar

FullCalendar es un potente *plugin* que muestra un calendario de tamaño personalizable, integra eventos AJAX para cambiar de fecha, se le pueden pasar una fuente de eventos para que los integre, su visualización es personalizable mediante hojas de estilo. Es de código abierto bajo la licencia MIT.

En el proyecto lo utilizo en dos escenarios, con diferente función y diferente interfaz de usuario.

Por una parte muestro en el la disponibilidad orientativa de habitaciones:

Figura 7.1 Calendario de disponibilidad

Por otra en el panel de administración se ofrece una vista para ver las reservas en una tabla y otra para verlas renderizadas en un calendario, así se obtiene una referencia más visual de las reservas.

Fuente: http://arshaw.com/fullcalendar

7.3.2.5 Flex Slider

Se utiliza en la página de inicio de la web para mostrar una galería de fotos (requiere jQuery).

Es una librería JavaScript para dinamizar galerías de imágenes. Flex Slider se adapta al tamaño del contenedor que lo integra, con lo que junto con nuestro *responsive design* conseguiremos una gran flexibilidad.

Fuentes: https://github.com/woothemes/flexslider, http://flexslider.woothemes.com

7.3.2.6 Gmap3

Se utiliza en la página de contacto, para mostrar un mapa de la situación del hotel en base a sus coordenadas. Utiliza la API de Google Maps para renderizar el mapa (requiere jQuery).

En lugar de insertar el *iframe* que ofrece Google con la dirección, esta forma es más personalizable y queda integrada en la propia web sin *iframes* a páginas externas. Además es personalizable y permite asociar otros eventos JavaScript que se quiera.

Fuente: http://gmap3.net

7.3.2.7 Pretty Photo

Las fotos que se muestran en la web, en las diferentes páginas (incluyendo la galería) se puede hacer click para verlas a mayor tamaño. Para no mover al usuario de la página se decide cargarlas en un lightbox, colocando la foto en primer plano y añadiéndole una descripción. Para esto se utiliza el plugin Pretty Photo (requiere jQuery).

Fuente: http://www.no-margin-for-errors.com/projects/prettyphoto-jquery-lightbox-clone/#prettyPhoto

7.3.2.8 Modernizr

Modernizr es una librería *Javascript* que nos va a ayudar con la detección del soporte de capacidades *HTML5* y *CSS3* de un navegador. De este modo, Modernizr no añade ninguna funcionalidad al navegador, sólo averigua si la funcionalidad se está implementando funciona en dicho navegador. Esto nos permite experimentar con las nuevas características de HTML5 y CSS3 sin preocuparnos por restar experiencia de usuario o que no se muestre el contenido adecuadamente.

Fuentes: http://modernizr.com, http://lineadecodigo.com/javascript/modernizr-y-las-capacidades-html5-y-css3

7.3.2.9 JQuery Responsive Menu Plugin

Se utiliza para adaptar el menú de navegación a anchos estrechos, convirtiendo este en un elemento HTML `<select>`, en lugar de un listado de opciones.

Fuente: https://github.com/mattkersley/Responsive-Menu

7.3.2.10 JQuery Superfish

Para la creación del menú de navegación de la Web, utilizo el *plugin* ***Superfish***.

Con este plugin, que combina JavaScript y CSS para personalizar la vista, a partir de una lista de elementos HTML …
… se consigue un **menú desplegable**. Existén muchas librerías de este tipo pero esta es una de las más utilizadas.

Algunas de sus características:

- Soporta dispositivos móviles
- Tiempo de espera para desplegar el menú personalizable.
- Soporta animaciones.
- Accesible desde el teclado.
- Permite submenús anidados.
- Opcionalmente se pueden añadir funciones JavaScript a sus acciones y delegación de eventos.

Fuente: http://users.tpg.com.au/j_birch/plugins/superfish

7.3.2.11 HTML5 Shiv

Internet Explorer no reconoce los nuevos elementos del HTML5 (article, section, etc), es por esto que para que los mismos se visualicen correctamente estilizados en el mencionado navegador, es necesario crearlos mediante JavaScript.

```
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Fuente: https://github.com/aFarkas/html5shiv

7.3.3 CSS

7.3.3.1 Skeleton

Skeleton es una pequeña colección de archivos CSS que pueden ayudar a desarrollar rápidamente sitios que usables en cualquier tamaño, ya sea una pantalla de ordenador portátil o un dispositivo móvil. *Skeleton* se basa en tres principios fundamentales:

- **Diseño adaptativo (Response Design):** *Skeleton* tiene tamaño de 960px como base, pero se adapta a tamaños reducidos, tabletas, teléfonos móviles (en horizontal y vertical).
- **Facilita un desarrollo rápido**: Skeleton es una herramienta para el desarrollo rápido. Comience rápidamente con CSS mejores prácticas, una red bien estructurada que hace que la consideración móvil fácil, una estructura de archivo organizado y elementos de la interfaz como formas básicas súper ligera estilo, botones, pestañas y más.
- **Estilo modificable**: Skeleton no es un marco de interfaz de usuario. Se trata de un kit de desarrollo que ofrece los estilos más básicos como base, pero está ideada para adoptarlo a un diseño personalizado.

7.3.4 Fuentes

7.3.4.1 Awesome Font

Awesome Font es una fuente que cargamos en nuestra aplicación web, pero no se utiliza para escribir textos, si no para representar **iconos**. Es una fuente de iconos mediante la que se pueden insertar algunos iconos sociales y símbolos en lugar de imágenes, con lo que la página es más ligera. No se trata de imágenes, como digo, sino de tipografía que se puede estilizar mediante CSS. Las ventajas de utilizar esta fuente en lugar de imágenes son varias:

- Control con CSS. Podemos aplicar degradados, tamaño, colores tanto de fuente como de fondo, transiciones, y todo lo que permita CSS.
- Escalabilidad infinita. No importa a que tamaño necesites el icono, al estar desarrollado con gráficos vectoriales nunca se pixelará.
- Cuenta con una amplia colección de 361 iconos.
- Es totalmente libre para uso comercial.
- Compatible con todos los navegadores. Soporta hasta Internet Explorer 7.

Se utiliza tanto en la parte pública como en la parte privada de administración, pero se le saca más en esta última al necesitar más iconos.

Fuente: http://fortawesome.github.io/Font-Awesome

7.3.4.2 PT Sans Narrow

Fuente base para el texto de la web. Licencia libre.

Fuente: http://www.google.com/fonts/specimen/PT+Sans+Narrow

7.3.5 Otros paquetes

7.3.5.1 Image Magick

Es un paquete para Linux, es necesario para que la gema ***PaperClip*** descrita anteriormente tenga la capacidad de subir una imagen al servidor en varias resoluciones, es decir redimensionarla. Sin este paquete es posible subir imágenes con *PaperClip* pero solo en el tamaño original. Las imágenes se pueden recortar, girar, cambiarlas de color o incluso combinarlas con otras.

Además se pueden aplicar varios efectos y agregar a las imágenes texto, líneas, polígonos, elipses y curvas de Bézier.

ImageMagick es software libre, el cual proporciona el código fuente completo que puede ser libremente usado, copiado, modificado y distribuido. Su licencia es compatible con la licencia GPL. Funciona en la mayoría de los sistemas operativos.

La mayor parte de la funcionalidad de *ImageMagick* puede ser aprovechada en linea de comandos, sin embargo, es más común que se use desde programas escritos en los siguientes lenguajes de programación: *C, Ch, C++, Java, Lisp, Pascal, Perl, PHP, Python, Ruby y Tcl/Tk*. Para cada uno de ellos, *ImageMagick* posee una interfaz la cual hace posible modificar o crear imágenes de forma automática y dinámica.

7.4 Herramientas y Programas Usados para el Desarrollo

7.4.1 NetBeans

En el proyecto se ha utilizado **NetBeans** como entorno de desarrollo para el código Ruby, HTML, CSS y JavaScript. El entrono ha sido desarrollado principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre, de código abierto y gratuito sin restricciones de uso.

NetBeans posee una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

7.4.2 Mercurial

Mercurial es un sistema de control de versiones multiplataforma, para desarrolladores de software. Está implementado principalmente haciendo uso del lenguaje de programación Python, pero incluye una implementación binaria de diff escrita en C.

Mercurial fue escrito originalmente para funcionar sobre Linux. Ha sido adaptado para Windows, Mac OS X y la mayoría de otros sistemas tipo Unix. Mercurial es, sobre todo, un programa para la línea de comandos. Todas las operaciones de Mercurial se invocan como opciones dadas a su programa motor, hg (cuyo nombre hace referencia al símbolo químico del mercurio).

Las principales metas de desarrollo de Mercurial incluyen un gran rendimiento y escalabilidad; desarrollo completamente distribuido, sin necesidad de un servidor, gestión robusta de archivos tanto de textos como binarios, y capacidades avanzadas de ramificación e integración, todo ello manteniendo sencillez conceptual. Incluye una interfaz web integrada.

El código fuente se encuentra disponible bajo los términos de la licencia GNU GPL versión 2, lo que clasifica a Mercurial como software libre.

7.4.3 Bitbucket

Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Bitbucket ofrece planes comerciales y gratuitos. Se ofrece cuentas gratuitas con un número ilimitado de repositorios privados (que puede tener hasta cinco usuarios en el caso de cuentas gratuitas). El servicio está escrito en Python.

7.4.4 Firefox + Firebug

Firebug es una extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y depurar el código fuente, CSS, HTML y JavaScript de una página web de manera instantánea e inline.

Firebug no es un simple inspector como DOM Inspector, además edita y permite guardar los cambios, un paso por delante del conocido Web Developer. Su atractiva e intuitiva interfaz, con solapas específicas para el análisis de cada tipo de elemento (consola, HTML, CSS, Script, DOM y red), permite al usuario un manejo fácil y rápido. Firebug está encapsulado en forma de plug-in o complemento de Mozilla, es software libre y de distribución gratuita.

7.4.5 Ubuntu

El proyecto se desarrolla sobre el sistema operativo Ubuntu, es una distribución Linux basada en Debian que proporciona un sistema operativo para el usuario medio, con un fuerte enfoque en la facilidad de uso e instalación del sistema. Al igual que otras distribuciones se compone de múltiples paquetes de software normalmente distribuidos bajo una licencia libre o de código abierto. Estadísticas web sugieren que el porcentaje de mercado de Ubuntu dentro de las distribuciones Linux es de aproximadamente 50%, y con una tendencia a subir como servidor web.

Está patrocinado por Canonical Ltd., una compañía británica propiedad del empresario sudafricano Mark Shuttleworth que en vez de vender la distribución con fines lucrativos, se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico.

Además, al mantenerlo libre y gratuito, la empresa es capaz de aprovechar los desarrolladores de la comunidad en mejorar los componentes de su sistema operativo. Canonical también apoya y proporciona soporte para cuatro derivaciones de Ubuntu: Kubuntu, Xubuntu, Edubuntu y la versión de Ubuntu orientada a servidores (Ubuntu Server Edition).

Su eslogan es “Linux for human beings” (Linux para seres humanos) y su nombre proviene de la ideología sudafricana Ubuntu (humanidad hacia otros).

7.5 Creación del Sistema

7.5.1 Problemas Encontrados

Implantación de un sistema de *login* seguro con diferentes permisos de usuario: se necesitaba implantar un sistema para autenticar al usuario, se optó por utilizar la dupla Devise + Can Can.

Fuentes:

http://www.phase2technology.com/blog/authentication-permissions-and-roles-in-rails-with-devise-cancan-and-role-model ,
http://edapx.com/2012/04/18/authorization-and-user-management-in-rails

Cargar imágenes en varios tamaños para la galería fotográfica y subirlas al servidor: es necesario subir imágenes nuevas o sustituir las existentes en la galería de fotos por otras. Para esta tarea utilice la gema *PaperClip*, el problema es que las imágenes deben subirse en varias resoluciones, para mostrar una imagen pequeña en la galería general y permitir ampliar la imagen al hacer click sobre ella. Tras instalar y configurar correctamente la gema, esta permitía subir imágenes correctamente al servidor, pero si además se quería subirlas en varios tamaños el formulario daba un error y no lo permitía.

Para solucionarlo, tras investigar por la web se descubrió que si además de subirla se quiere hacer redimensión de la imagen es necesario tener instalada la librería ImageMagick. Se realizó la instalación y la subida de imágenes funcionó perfectamente.

Fuentes:

http://burm.net/2008/10/07/the-ruby-on-rails-paperclip-plugin-tutorial-easy-imageattachments

http://jimneath.org/2008/04/17/paperclip-attaching-files-in-rails.html

http://rmagick.rubyforge.org

Capítulo 8. Desarrollo de las Pruebas

8.1 Pruebas Unitarias

Muchas metodologías ágiles como *Ruby on Rails* son partidarias de lo que se llama "probar primero".

La idea del desarrollo basado en pruebas consiste en utilizar las pruebas de unidad como una pre-verificación y guía para comenzar a escribir el código. Al escribir la prueba primero (o al menos al mismo tiempo que el módulo original), a la larga es más sencillo mantener el programa y el diseño nunca se perjudica al punto que crea dificultades en el control de calidad.

En Ruby on Rails, y en todos los lenguajes interpretados en uso hoy día, la idea de las pruebas de unidad es muy importante. Como casi todos los errores en un programa interpretado sólo ocurren cuando el programa corre (porque no hay etapa de compilación), en esta clase de lenguajes es esencial escribir las pruebas de unidad al mismo tiempo que el código y asegurar una cobertura adecuada.

Ruby on Rails utiliza por defecto la librería **Test::Unit** como estándar para crear pruebas de unidad. Cuando usted utiliza los generadores de Rails para crear cualquier módulo, ya sea modelos o controladores, Rails automáticamente crea una prueba de unidad para dicho controlador o modelo en el directorio /test, incluyendo una que otra prueba básica. El diseño MVC de Rails hace más fácil escribir pruebas de unidad. Cuando se extiende el programa sólo hay que mantener las pruebas.

En este proyecto se utilizara otra librería **Rspec** para las pruebas unitarias, las pruebas estarán en la carpeta /spec. RSpec es más potente y tiene una sintaxis mucho más legible. Se han aplicado pruebas unitarias organizadas por módulos, estas están en la carpeta /test, para su ejecución se siguen los siguientes pasos desde la carpeta del proyecto:

Se crea una base de datos para el entorno de test, previamente configurada en el *database.yml*.

```
bundle exec rake db:test:create
```

Se cargan las tablas del esquema de la base de datos basándose en la estructura del fichero schema.rb. Cada vez que el proyecto tenga modificaciones en la base de datos, deberemos ejecutar esta orden para que la base de datos de desarrollo y de test sean coherentes.

```
bundle exec rake db:test:prepare
```

Cada vez que se quieran ejecutar los test unitarios para los modelos previamente implementados se lanza la orden.

```
bundle exec rspec spec/models
```

Además de pruebas de modelos existen pruebas de enrutado y de helpers (ficheros de ayuda). Para ejecutar todas las pruebas en conjunto podemos ejecutar la orden:

```
bundle exec rspec spec
```

Si todo va bien el resultado sería algo así:

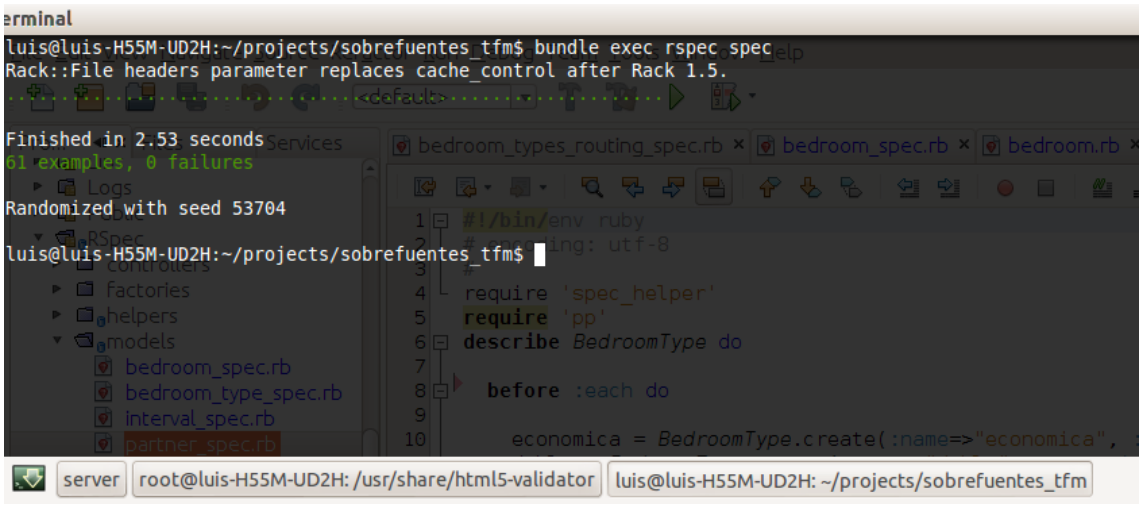


Figura 8.1. Pruebas unitarias

Además de la gema Rspec, se utiliza la gema ***Factory Girl*** para el desarrollo de las pruebas. Esta librería permite generar objetos de manera más sencilla para llevar a cabo las pruebas.

8.2 Pruebas de Integración y del Sistema

Las pruebas del sistema son pruebas de integración de la aplicación al completo, que permiten probar la funcionalidad de todo el sistema en conjunto y las relaciones con otros sistemas si procede. De esta manera verificamos que las especificaciones funcionales y técnicas se cumplan.

A continuación se detalla las pruebas de integración y de sistema sobre los siguientes casos de uso:

Caso de Uso: Gestión usuarios	
Prueba	**Resultado Esperado**
Añadir un usuario no existente	El sistema posee un usuario más
	Resultado Obtenido
	El sistema efectivamente posee un usuario más
Prueba	**Resultado Esperado**
Añadir un usuario que ya existe	El sistema no posee un usuario más y se muestra un dialogo notificándolo
	Resultado Obtenido
	Efectivamente, no se repite el usuario
Prueba	**Resultado Esperado**
Editar información personal de un usuario	Los datos del usuario se actualizan.
	Resultado Obtenido
	Efectivamente, se actualiza el objeto en la base de datos.
Prueba	**Resultado Esperado**
Solicito recuperar password de un usuario	El usuario recibe un email a su dirección de correo con una clave nueva.
	Resultado Obtenido
	Efectivamente, se resetea la contraseña.
Prueba	**Resultado Esperado**
Intento de login con usuario existente	El usuario inicia sesión en el sistema.
	Resultado Obtenido
	Efectivamente, entra en el sistema, salvo que introduzca un *password* incorrecto, entonces recibe una notificación.
Prueba	**Resultado Esperado**
Intento de acceso a una sección no autorizada por el usuario	El usuario no puede acceder, y recibe un mensaje de error.
	Resultado Obtenido
	Efectivamente, no se le permite el acceso.
Prueba	**Resultado Esperado**
El administrador intenta eliminar un usuario	Si no tiene reservas asociadas, el sistema borra al usuario, si tiene reservas, el sistema no permite su eliminación mostrando un mensaje de error.
	Resultado Obtenido
	Se cumple lo esperado.

Caso de Uso : Gestión reservas	
Prueba	**Resultado Esperado**
Cliente selecciona fechas incorrectas (pasadas, o salida anterior a la entrada)	El sistema muestra un mensaje de error
	Resultado Obtenido
	El sistema efectivamente muestra un error y no permite acceder al siguiente paso de reserva.
Prueba	**Resultado Esperado**
Cliente selecciona fechas sin disponibilidad de habitaciones	El sistema muestra un mensaje de error
	Resultado Obtenido
	El sistema efectivamente muestra un error, no muestra ninguna habitación disponible y no permite acceder al siguiente paso de reserva.
Prueba	**Resultado Esperado**
Cliente no rellena los campos de información personal requeridos	El sistema muestra un error asociado a cada campo.
	Resultado Obtenido
	Efectivamente muestra errores y no permite el paso al siguiente paso de la reserva.
Prueba	**Resultado Esperado**
El administrador intenta crear una reserva, pero no hay habitaciones disponibles	El sistema le permite enviar la solicitud al servidor, pero este devuelve un error informando de la imposibilidad de crear la reserva por falta de habitaciones.
	Resultado Obtenido
	Efectivamente el sistema valida la solicitud en el servidor y muestra un error.
Prueba	**Resultado Esperado**
El administrador intenta modificar las fechas de una reserva	El sistema valida si es posible el cambio de fechas, si existe disponibilidad
	Resultado Obtenido
	Si es posible, lo cambia, si no muestra un error.

Caso de Uso : Gestión habitaciones	
Prueba	**Resultado Esperado**
Administrador crea un nuevo tipo de habitación	El sistema crea el tipo de habitación, salvo que su nombre esté repetido.
	Resultado Obtenido
	El sistema efectivamente muestra un error si está repetida y la crea en caso contrario.
Prueba	**Resultado Esperado**
Administrador elimina un	Si existen habitaciones con ese tipo, no debe poder borrarse,

tipo de habitación	en caso contrario sí.
	Resultado Obtenido
	El sistema efectivamente muestra un error si tiene habitaciones asociadas.
Prueba	**Resultado Esperado**
Administrador crea una nueva habitación	El sistema crea el tipo de habitación, salvo que su nombre esté repetido.
	Resultado Obtenido
	El sistema efectivamente muestra un error si está repetida y la crea en caso contrario.
Prueba	**Resultado Esperado**
Administrador elimina una habitación	Si existen reservas con esa habitación, no debe poder borrarse, en caso contrario sí.
	Resultado Obtenido
	El sistema efectivamente muestra un error si tiene reservas asociadas.

Caso de Uso : Gestión intervalos de precios	
Prueba	**Resultado Esperado**
Administrador crea un nuevo intervalo de precio	El sistema crea el intervalo de precio, siempre y cuando no se solape con otro.
	Resultado Obtenido
	El sistema efectivamente muestra un error si se solapa y crea el intervalo en caso contrario.
Prueba	**Resultado Esperado**
Administrador elimina un tipo de habitación	Si existen habitaciones con ese tipo, no debe poder borrarse, en caso contrario sí.
	Resultado Obtenido
	El sistema efectivamente muestra un error si tiene habitaciones asociadas.
Prueba	**Resultado Esperado**
Administrador crea una nueva habitación	El sistema crea el tipo de habitación, salvo que su nombre esté repetido.
	Resultado Obtenido
	El sistema efectivamente muestra un error si está repetida y la crea en caso contrario.
Prueba	**Resultado Esperado**
Administrador elimina una habitación	Si existen reservas con esa habitación, no debe poder borrarse, en caso contrario sí.
	Resultado Obtenido
	El sistema efectivamente muestra un error si tiene reservas asociadas.

8.3 Pruebas de Usabilidad y Accesibilidad

8.3.1 Pruebas de Usabilidad

En el interfaz público el menú desplegable se sustituye por un selector de página, para los tamaños de pantalla más estrechos.

En la interfaz privada siempre se conserva el mismo estilo y estructura del panel de administración, moviéndose de una sección a otra por medio de pestañas que están a la vista en todo momento. Se intenta conseguir una interfaz sencilla y clara para facilitar su uso en el sistema de altas, bajas y modificaciones de datos. La pestaña seleccionada cambia de estilo, así el usuario siempre tiene información relativa a la sección que se encuentra. Para facilitar las acciones de visualizar, crear, eliminar y editar se han utilizado iconos intuitivos utilizando los iconos que proporciona la fuente *AwesomeFont*.

He seguido una guía de usabilidad para el interfaz público.

Criterios	¿Cumplido?
Generales	
¿Cuáles son los objetivos del sitio web? ¿Son concretos y bien definidos? ¿Los contenidos y servicios que ofrece se corresponden con esos objetivos?	**SI**
¿Tiene una URL correcta, clara y fácil de recordar? ¿Y las URL de sus páginas internas? ¿Son claras y permanentes?	**SI**
¿Muestra de forma precisa y completa qué contenidos o servicios ofrece realmente el sitio web? El diseño de la página de inicio debe ser diferente al resto de páginas y cumplir la función de 'escaparate' del sitio.	**SI**
¿La estructura general del sitio web está orientada al usuario? Los sitios web deben estructurarse pensando en el usuario, sus objetivos y necesidades. La estructura interna de la empresa u organización, cómo funciona o se organiza no interesan al usuario.	**SI**
¿El *look & feel* general se corresponde con los objetivos, características, contenidos y servicios del sitio web? Ciertas combinaciones de colores ofrecen imágenes más o menos formales, serias o profesionales.	**SI**
¿Es coherente el diseño general del sitio web? Se debe mantener una coherencia y uniformidad en las estructuras y colores de todas las páginas. Esto sirve para que el usuario no se desoriente en su navegación.	**SI**
¿Es reconocible el diseño general del sitio web? Cuánto más se parezca el sitio web al resto de sitios web, más fácil será de usar.	**SI**
¿El sitio web se actualiza periódicamente? ¿Indica cuándo se actualiza? Las fechas que se muestren en la página deben corresponderse con actualizaciones, noticias, eventos...no con la fecha del sistema del usuario.	**Pocos cambios**
Identidad e Información	
¿Se muestra claramente la identidad de la empresa-sitio a través de todas las páginas?	**SI**
El Logotipo, **¿es significativo, identificable y suficientemente visible?**	**Lo será (pendiente**

	de diseño)
El eslogan o *tagline*, **¿expresa realmente qué es la empresa y qué servicios ofrece?**	**SI, varios eslóganes en el carrusel de imágenes del index**
¿Se ofrece algún enlace con información sobre la empresa, sitio web, 'webmaster',...?	**SI**
¿Se proporciona mecanismos para ponerse en contacto con la empresa? (email, teléfono, dirección postal, fax...)	**SI**
¿Se proporciona información sobre la protección de datos de carácter personal de los clientes o los derechos de autor de los contenidos del sitio web?	**SI**
En artículos, noticias, informes... **¿Se muestra claramente información sobre el autor, fuentes y fechas de creación y revisión del documento?**	**No aplica**
Lenguaje y Redacción	
¿El sitio web habla el mismo lenguaje que sus usuarios? Se debe evitar usar un lenguaje corporativista. Así mismo, hay que prestarle especial atención al idioma, y ofrecer versiones del sitio en diferentes idiomas cuando sea necesario.	**SI**
¿Emplea un lenguaje claro y conciso?	**SI**
¿Es amigable, familiar y cercano? Es decir, lo contrario a utilizar un lenguaje constantemente imperativo, mensajes crípticos, o tratar con "desprecio" al usuario.	**SI**
¿1 párrafo = 1 idea? Cada párrafo es un objeto informativo. Trasmita ideas, mensajes...Se deben evitar párrafos vacíos o varios mensajes en un mismo párrafo.	**SI**
Rotulado	
Los rótulos, **¿son significativos?** Ejemplo: evitar rótulos del tipo "haga clic aquí".	**SI**
¿Usa rótulos estándar? Siempre que exista un "estándar" comúnmente aceptado para el caso concreto, como "Mapa del Sitio" o "Acerca de...".	**SI**
¿Usa un único sistema de organización, bien definido y claro? No se deben mezclar diferentes. Los sistemas de organización son: alfabético, geográfico, cronológico, temático, orientado a tareas, orientado al público y orientado a metáforas.	**SI**
¿Utiliza un sistema de rotulado controlado y preciso? Por ejemplo, si un enlace tiene el rótulo "Quiénes somos", no puede dirigir a una página cuyo encabezamiento sea "Acerca de"	**SI**
El título de las páginas, **¿Es correcto? ¿Ha sido planificado?** Relacionado con la capacidad para poder buscar y encontrar el sitio *web*.	**SI**
Estructura y Navegación	
La estructura de organización y navegación, **¿Es la más adecuada?** Hay varios tipos de estructuras: jerárquicas, hipertextual, facetada,...	**SI**
En el caso de estructura jerárquica, **¿Mantiene un equilibrio entre Profundidad y Anchura?**	**SI**

En el caso de ser puramente hipertextual, **¿Están todos los clúster de nodos comunicados?** Aquí se mide la distancia entre nodos.	**SI**
¿Los enlaces son fácilmente reconocibles como tales? ¿Su caracterización indica su estado (visitados, activos,...)? Los enlaces no sólo deben reconocerse como tales, sino que su caracterización debe indicar su estado, y ser reconocidos como una unidad	**SI**
En menús de navegación, **¿Se ha controlado el número de elementos y de términos por elemento para no producir sobrecarga memorística?** No se deben superar los 7±2 elementos, ni los 2 o, como mucho, 3 términos por elemento.	**SI**
¿Es predecible la respuesta del sistema antes de hacer clic sobre el enlace? Relacionado con el nivel de significación del rótulo del enlace, aunque también con: el uso de globos de texto, información contextual, la barra de estado del navegador,...	**SI**
¿Se ha controlado que no haya enlaces que no llevan a ningún sitio? Enlaces que no llevan a ningún sitio: Los enlaces rotos, y los que enlazan con la misma página que se está visualizando (por ejemplo enlaces a la "home" desde la misma página de inicio)	**SI**
¿Existen elementos de navegación que orienten al usuario acerca de dónde está y cómo deshacer su navegación? ...como *breadcrumbs*, enlaces a la página de inicio,...recuerde que el logo también es recomendable que enlace con la página de inicio.	**SI**
Las imágenes enlace, **¿se reconocen como clicables? ¿Incluyen un atributo 'title' describiendo la página de destino?** En este sentido, también hay que cuidar que no haya imágenes que parezcan enlaces y en realidad no lo sean.	**SI**
¿Se ha evitado la redundancia de enlaces?	**SI**
¿Se ha controlado que no haya páginas "huérfanas"? Páginas huérfanas: que aún siendo enlazadas desde otras páginas, éstas no enlacen con ninguna.	**SI**
Layout de la Página	
¿Se aprovechan las zonas de alta jerarquía informativa de la página para contenidos de mayor relevancia? (como por ejemplo la zona central)	**SI**
¿Se ha evitado la sobrecarga informativa? Esto se consigue haciendo un uso correcto de colores, efectos tipográficos y agrupaciones para discriminar información. Los grupos diferentes de objetos informativos de una página deben ser 7±2.	**SI**
¿Es una interfaz limpia, sin ruido visual?	**SI**
¿Existen zonas en "blanco" entre los objetos informativos de la página para poder descansar la vista?	**SI**
¿Se hace un uso correcto del espacio visual de la página? Es decir, que no se desaproveche demasiado espacio con elementos de decoración, o grandes zonas en "blanco", y que no se adjudique demasiado espacio a elementos de menor importancia.	**SI**
¿Se utiliza correctamente la jerarquía visual para expresar las relaciones del tipo "parte de" entre los elementos de la página? (La jerarquía visual se utiliza para orientar al usuario)	**SI**

¿Se ha controlado la longitud de página? Se debe evitar en la medida de lo posible el *scrolling*. Si la página es muy extensa, se debe fraccionar.	**SI**
Búsqueda (si es necesario, por la extensión del sitio, incorporar un buscador interno)	
¿Se encuentra fácilmente accesible? Es decir: directamente desde la home, y a ser posible desde todas las páginas del sitio, y colocado en la zona superior de la página.	**SI**
¿Es fácilmente reconocible como tal?	**SI**
¿Permite la búsqueda avanzada? (siempre y cuando, por las características del sitio web, fuera de utilidad que la ofreciera)	**NO**
¿Muestra los resultados de la búsqueda de forma comprensible para el usuario?	**-**
¿La caja de texto es lo suficientemente ancha?	**-**
¿Asiste al usuario en caso de no poder ofrecer resultados para una consultada dada?	**-**
Elementos Multimedia	
¿Las fotografías están bien recortadas? ¿Son comprensibles? ¿Se ha cuidado su resolución?	**SI**
¿Las metáforas visuales son reconocibles y comprensibles por cualquier usuario? (prestar especial atención a usuarios de otros países y culturas)	**SI**
¿El uso de imágenes o animaciones proporciona algún tipo de valor añadido?	**SI**
¿Se ha evitado el uso de animaciones cíclicas?	**SI**
Ayuda	
Si posee una sección de Ayuda, **¿Es verdaderamente necesaria?** Siempre que se pueda prescindir de ella simplificando los elementos de navegación e interacción, debe omitirse esta sección.	**-**
En enlace a la sección de Ayuda, **¿Está colocado en una zona visible y "estándar"?** La zona de la página más normal para incluir el enlace a la sección de Ayuda, es la superior derecha.	**-**
¿Se ofrece ayuda contextual en tareas complejas? (transferencias bancarias, formularios de registro...)	**-**
Si posee *FAQs*, **¿Es correcta tanto la elección como la redacción de las preguntas? ¿Y las respuestas?**	**-**
Accesibilidad (debería cubrirse con los test de Accesibilidad posteriores)	
¿El tamaño de fuente se ha definido de forma relativa, o por lo menos, la fuente es lo suficientemente grande como para no dificultar la legibilidad del texto?	**SI**
¿El tipo de fuente, efectos tipográficos, ancho de línea y alineación empleadas facilitan la lectura?	**SI**
¿Existe un alto contraste entre el color de fuente y el fondo?	**SI**
¿Incluyen las imágenes atributos 'alt' que describan su contenido?	**SI**
¿Es compatible el sitio web con los diferentes navegadores? ¿Se visualiza correctamente con diferentes resoluciones de pantalla? Se debe prestar atención a: *JScript*, *CSS*, tablas, fuentes...	**SI**
¿Puede el usuario disfrutar de todos los contenidos del sitio web sin necesidad de tener que descargar e instalar *plugins* adicionales?	**SI**

¿Se ha controlado el peso de la página? Se deben optimizar las imágenes, controlar el tamaño del código *JScript*...	**SI**
¿Se puede imprimir la página sin problemas? Leer en pantalla es molesto, por lo que muchos usuarios preferirán imprimir las páginas para leerlas. Se debe asegurar que se puede imprimir la página (no salen partes cortadas), y que el resultado es legible.	**SI**
Control y Retroalimentación	
¿Tiene el usuario todo el control sobre el interfaz? Se debe evitar el uso de ventanas pop-up, ventanas que se abren a pantalla completa, banners intrusivos...	**SI**
¿Se informa constantemente al usuario acerca de lo que está pasando? Si el usuario tiene que esperar hasta que se termine una operación, se debe mostrar un mensaje indicándoselo y que debe esperar, con el tiempo de espera estimado o una barra de progreso.	**SI**
¿Se informa al usuario de lo que ha pasado? Por ejemplo, cuando un usuario valora un artículo o responde a una encuesta, se le debe informar de que su voto ha sido procesado correctamente.	**SI**
Cuando se produce un error, **¿se informa de forma clara y no alarmista al usuario de lo ocurrido y de cómo solucionar el problema?** Siempre es mejor intentar evitar que se produzcan errores a tener que informar al usuario del error.	**SI**
¿Posee el usuario libertad para actuar? NO restringir la libertad del usuario: Uso de animaciones que no pueden ser "saltadas", páginas en las que desaparecen los botones de navegación, no impida al usuario poder usar el botón derecho de su ratón...	**SI**
¿Se ha controlado el tiempo de respuesta? Esto tiene que ver con el peso de cada página (accesibilidad) y tiene relación con el tiempo que tarda el servidor en finalizar una tarea y responder. El tiempo máximo que esperará un usuario son 10 segundos	**SI**
Aclaraciones	
¿Se ha evaluado adecuadamente la orientación del usuario? (Donde estoy, como volver, que he visitado, que va a pasar)	**SI**
¿Se ha usado correctamente la publicidad?	**-**

8.3.2 Pruebas de Accesibilidad

Un correcto cumplimiento de los estándares de accesibilidad hubiera supuesto un sobrecoste en el proyecto el cual el cliente no estaba dispuesto a asumir. Cumplir los estándares AAA de accesibilidad queda por lo tanto fuera del alcance del proyecto y recogido en futuras ampliaciones. No se ha puesto especial énfasis en que la aplicación sea accesible, pero si se ha tenido en cuenta la accesibilidad durante el desarrollo del interfaz.

No obstante se han utilizado herramientas para calcular el nivel de accesibilidad actual de la interfaz pública a la que tienen acceso los visitantes:

Como la parte privada está orientado a un grupo muy pequeño y previamente definido de usuarios se ha dado prioridad a la usabilidad frente a la accesibilidad.

Se ha comprobado la correcta visualización de la interfaz en diferentes equipos, navegadores y resoluciones entre 1900px de ancho y 320px, a continuación enumero los navegadores en los que se ha comprobado:

- Mozilla Firefox 3 y 4
- Internet Explorer 8
- Google Chrome
- Opera

Herramientas automáticas de evaluación de accesibilidad

Se ha evaluado la página con la herramienta de análisis automático HERA, ofreciendo los siguientes resultados:

Status of checkpoints

Priority	Needs checking	Pass	Fail	N/A
HERA P1 WCAG 1.0	8	--	--	9
HERA P2 WCAG 1.0	17	3 ✔	2 ✘	7
HERA P3 WCAG 1.0	11	2 ✔	1 ✘	5

Figura 8.2. Test HERA

Los fallos de prioridad 2 detectados han sido:

- Punto 3.2 - Resultado: incorrecto
  - CSS**: código de hoja de estilos incorrecto**.

Que la hoja de estilos CSS no valide es debido a notaciones CSS3 que pueden no estar reconocidas por el analizar automático de HERA.

- Punto 3.4 - Resultado: incorrecto
    - **Unidades absolutas en CSS**: Se detectaron unidades absolutas (in|cm|mm|pt|pc) o tamaños de fuente definidos en px en los valores de las hojas de estilo.

Al tener un diseño que se adapta al ancho del interfaz, considero que las unidades absolutas no son tan críticas como en una interfaz con un solo diseño.

Los errores de prioridad 3 detectados han sido:

- Punto 9.5 - Resultado: incorrecto
    - **Atajos de teclado**: No se proporcionan atajos de teclado.

8.4 Pruebas de Rendimiento

Se ha procurado que el interfaz de administración sea ligero adaptando la resolución de fotografías a varios tamaños y tomando las siguientes medidas buscando rapidez:

- **Cargar las hojas de estilo (CSS) en el inicio**. Al final de varias pruebas e investigación se ha encontrado que para cuestiones de desempeño se debe de colocar las CSS's en el HEAD usando la etiqueta LINK.
- **Poner los scripts al final**. Lo opuesto a las CSS, es mejor mover los scripts hacia el final de las páginas (siempre y cuando sea posible). Esto permite la renderización y descarga paralelamente.
- **Evitar expresiones CSS inline**. Las expresiones CSS son una poderosa (y peligrosa) forma de poner propiedades dinámicamente. Un problema es que no todos los browser las soportan y otro es que podrían ser re-evaluadas cada vez que la página cambia, por redimensionamiento, scrolling, movimiento del mouse, clicks, etc.
- **Hacer externos los archivos Javscripts y CSS**. Esta regla hace que los componentes Javascript y CSS puedan ser cachados en el browser del usuario y ponerlos internos hace ineficiente la ventaja del cache del browser.
- **Minimizar el JavaScript**. Minificación es la práctica de remover los caracteres innecesarios del código para reducir su tamaño y de esta manera mejorar el tiempo de carga.
- **Evitar redireccionamientos**. Un redireccionamiento (redirect) es usado para enrutar los usuarios de una URL a otra. Regularmente son para documentos HTML pero también para requerir componentes (imagenes, scripts, etc.) en las páginas.
- **Remover scripts duplicados**. Incluir dos veces el mismo archivo Javascript afecta el desempeño. Este error no es raro, una revisión de sitios web mostró que CNN y YouTube contenían un script duplicado. Asegurarnos que los scripts sean incluidos una sola vez.
- **Evitar el uso de imágenes decorativas**. Utilizar HTML y CSS en la medida de lo posible para elementos decorativos como banners o botones. Estos elementos son más ligeros y descriptivos para buscadores que una imagen.

Se ha ejecutado la herramienta de Google **PageSpeed** con los siguientes resultados:

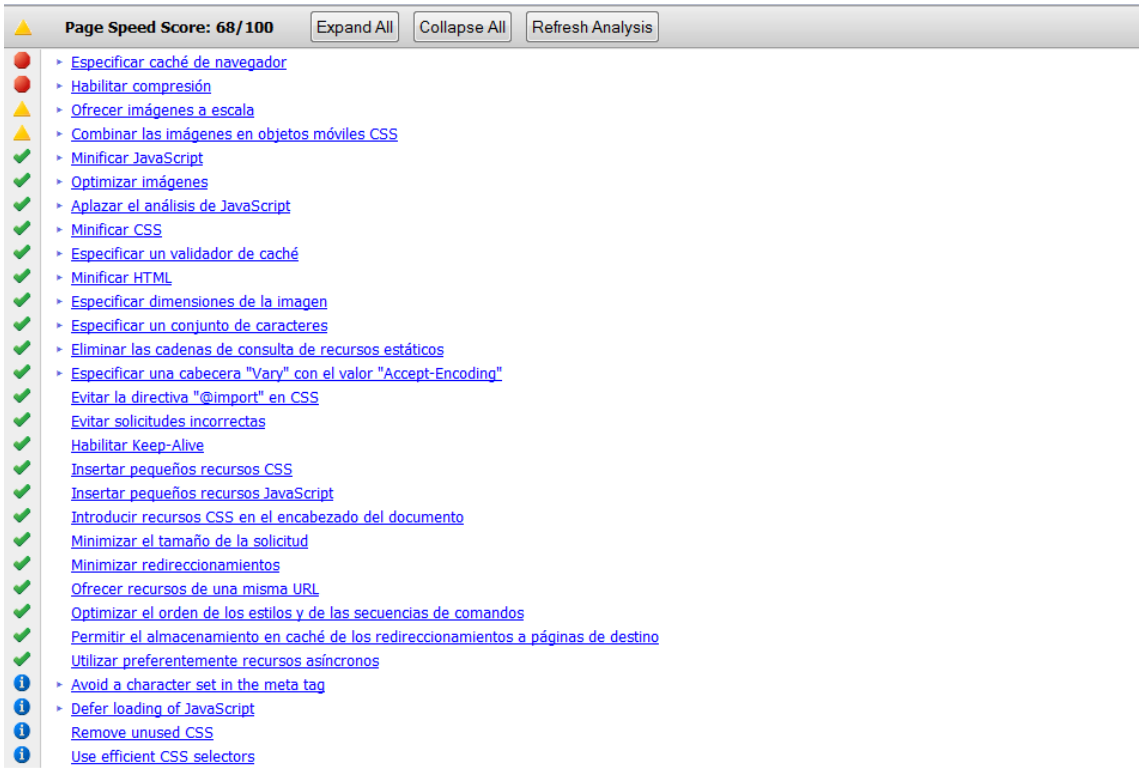


Figura 8.3. Herramienta Google Page Speed

La compresión de código se habilitará cuando el proyecto se suba a producción, no en un entorno de desarrollo. Las imágenes no están a escala porque deben adaptarse a diferentes anchos.

Capítulo 9. Manuales del Sistema

9.1 Manual de Instalación

El desarrollo se ha realizado sobre una distribución Linux, concretamente Ubuntu 12.04. La guía de instalación del entorno se desarrollará con esta distribución como sistema operativo base.

Para reproducir la aplicación necesitamos tener instalado *Ruby* 1.9.2 y *Rails* 3.2.11 y un servidor de base de datos. MySQL en este caso.

En primer lugar instalaremos la base de datos MySQL y el *Workbench*, por si queremos gestionar posteriormente las bases de datos.

```
luis@sam:~/ sudo apt-get install mysql-server mysql-workbench
```

Ahora necesitaremos el entorno *Ruby on Rails*. Para esto tenemos varias opciones, instalar la versión concreta de cada herramienta, es decir, por una parte Ruby, y por otra *Rails* o instalar RVM, que nos permitirá tener varias versiones de Ruby instaladas en el equipo, y cambiar de una a otra de manera sencilla. Personalmente prefiero esta segunda. RVM es un gestor de versiones de Ruby: https://rvm.io

Instalación individual:

```
luis@sam:~/projects/sobrefuentes$ sudo apt-get install ruby -v 1.9.2
```

Si todo va bien podemos comprobar la versión de Ruby con el comando:

```
luis@sam:~/projects/sobrefuentes$ ruby -v
ruby 1.9.2p320 (2012-04-20 revision 35421) [i686-linux]
```

Después instalamos la gema Rails.

```
luis@sam:~/projects/sobrefuentes$ sudo gem install rails -v 3.2.11
```

```
luis@sam:~/projects/sobrefuentes$ rails -v
Rails 3.2.11
```

Instalación semiautomática mediante RVM: https://rvm.io/rvm/install

Para la instalación de RVM debemos tener preinstalado en nuestro equipo el paquete *curl*.

```
sudo apt-get install curl
```

Una vez tenemos curl, podemos instalar RVM:

```
luis@sam:~/projects/sobrefuentes$ \curl -L https://get.rvm.io | bash -s stable --ruby
```

Esto nos instalará la última versión estable de Ruby, a nosotros nos interesa la versión 1.9.2, entonces le indicamos que la agregue:

```
luis@sam:~/projects/sobrefuentes$ rvm install 1.9.2
```

Luego, podemos escoger versión mediante el comando:

```
luis@sam:~/projects/sobrefuentes$  rvm use --default 1.9.2
```

Si este comando no nos funcionara, debemos editar el fichero bashrc de nuestra carpeta personal y añadir las líneas:

```
[[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm" # Load RVM into a
shell session *as a function*
PATH=$PATH:$HOME/bin:$HOME/.rvm/bin # Add RVM to PATH for scripting
```

Ahora nos resulta muy sencillo cambiar de versión de ruby entre diferentes aplicaciones con el comando *rvm use –default versión.*

Una vez tenemos Ruby instalado no necesitamos instalar Rails, la gema Bundler lo hará por nosotros. Nos situamos en la carpeta del proyecto y ejecutamos la orden: **bundle install.** Esto lo que hace es instalar todas las gemas que tenemos declaradas en el fichero *Gemfile* y todas sus dependencias.

```
luis@sam:~/projects/sobrefuentes$ bundle install
Using rake (10.0.3)
Using i18n (0.6.1)
Using multi_json (1.5.0)
Using activesupport (3.2.11)
Using builder (3.0.4)
Using activemodel (3.2.11)
Using erubis (2.7.0)
Using journey (1.0.4)
Using rack (1.4.4)
Using rack-cache (1.2)
Using rack-test (0.6.2)
Using hike (1.2.1)
Using tilt (1.3.3)
Using sprockets (2.2.2)
Using actionpack (3.2.11)
Using mime-types (1.19)
Using polyglot (0.3.3)
Using treetop (1.4.12)
Using mail (2.4.4)
Using actionmailer (3.2.11)
Using arel (3.0.2)
Using tzinfo (0.3.35)
Using activerecord (3.2.11)
Using activeresource (3.2.11)
Using addressable (2.3.2)
Using bcrypt-ruby (3.0.1)
Using cancan (1.6.10)
Using cocaine (0.4.2)
Using coffee-script-source (1.4.0)
Using execjs (1.4.0)
```

```
Using coffee-script (2.2.0)
Using rack-ssl (1.3.3)
Using json (1.7.6)
Using rdoc (3.12)
Using thor (0.17.0)
Using railties (3.2.11)
Using coffee-rails (3.2.2)
Using yamler (0.1.0)
Using configatron (2.13.0)
Using orm_adapter (0.4.0)
Using warden (1.2.1)
Using devise (2.2.3)
Using diff-lcs (1.1.3)
Using launchy (2.2.0)
Using email_spec (1.4.0)
Using factory_girl (4.2.0)
Using factory_girl_rails (4.2.1)
Using ffi (1.4.0)
Using gravatar_image_tag (1.1.3)
Using hirb (0.7.1)
Using jquery-rails (3.0.1)
Using mysql2 (0.3.11)
Using paperclip (2.8.0)
Using railroady (1.1.0)
Using bundler (1.3.5)
Using rails (3.2.11)
Using rb-inotify (0.8.8)
Using rspec-core (2.12.2)
Using rspec-expectations (2.12.1)
Using rspec-mocks (2.12.2)
Using rspec-rails (2.12.2)
Using sass (3.2.5)
Using sass-rails (3.2.6)
Using uglifier (1.3.0)
Using wicked_pdf (0.9.6)
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
luis@sam:~/projects/sobrefuentes$
```

Si este comando no funcionara, deberíamos instalar la gema *bundler* a mano mediante la orden:

```
luis@sam:~/projects/sobrefuentes$ sudo gem install bundler
```

Tras esto volver a repetir la orden anterior. Lo que hace *bundler* es buscar en los repositorios de la gema, la versión correcta para todas las librerías declaradas en el archivo Gemfile de nuestro proyecto. En función de nuestra versión de Rails el sistema instalará una u otra, es importante instalar las gemas a través de bundler, porque una instalación directa con la orden gem install, podría provocar incompatibilidades y fallos en el sistema

Además de los pasos anteriores, para la correcta ejecución de las subidas y transformaciones de imágenes, y la generación de PDFs, es necesario tener instaladas los siguientes paquetes en nuestra distribución Linux:

```
sudo  apt-get install wkhtmltopdf
sudo  apt-get install libmagick++-dev
sudo  apt-get install imagemagick
```

9.2 Manual de Ejecución

Una vez tengamos el entorno correctamente instalado, necesitamos arrancar la aplicación pero antes debemos generar la base de datos. Esta operación sólo es necesario hacerla la primera vez, luego la base de datos se queda generada, salvo que volvamos a generarla, que se resetearía perdiendo los cambios que hubiéramos hecho.

Antes de crearla, debemos configurar la base de datos de nuestra aplicación en el fichero *config/database.yml* indicándole el nombre que queremos darle, y las credenciales de acceso a la base de datos.

```
development:
  adapter: mysql2
  encoding: utf8
  reconnect: false
  database: devise_sobrefuentes
  pool: 5
  username: root
  password: xxxx
  socket: /var/run/mysqld/mysqld.sock
  timeout: 5000
```

Una vez configurada, nos situamos en la carpeta de nuestro proyecto y ejecutamos las órdenes:

```
luis@sam:~/projects/sobrefuentes$ bundle exec rake db:create
luis@sam:~/projects/sobrefuentes$ bundle exec rake db:reset
```

La primera orden creará la base de datos vacía, y la 2ª cargará los objetos declarados en el contenido de nuestro fichero */db/sedes.rb*

```
# encoding: UTF-8
# This file should contain all the record creation needed to seed the database with its
default values.
# The data can then be loaded with the rake db:seed (or created alongside the db with
db:setup).
#
# Examples:
#
#   cities = City.create([{ name: 'Chicago' }, { name: 'Copenhagen' }])
#   Mayor.create(name: 'Emanuel', city: cities.first)

admin = Role.create(:name => "Admin")
Role.create(:name => "Member")
Role.create(:name => "Robber")


luis = User.create(:email=>"luis.amor.alvarez@gmail.com", :password=>"xxxx")
timmy = User.create(:email=>"timmytimmyy@hotmail.com", :password=>"xxxx")

luis.role_ids = [1]

economica = BedroomType.create(:name=>"Econónica", :price =>50)
doble = BedroomType.create(:name=>"Doble",         :price =>60)
especial = BedroomType.create(:name=>"Especial",   :price =>70)

#Economic intervals
Interval.create(:price=>40, :name=>"Temporada baja", :start=>Date.parse('01-01-2013'),
:end=>Date.parse('31-05-2013'), :bedroom_type_id=>economica.id)
Interval.create(:price=>50, :name=>"Junio", :start=>Date.parse('01-06-2013'),
:end=>Date.parse('30-06-2013'), :bedroom_type_id=>economica.id)
Interval.create(:price=>60, :name=>"Julio", :start=>Date.parse('01-07-2013'),
```

```
:end=>Date.parse('31-07-2013'), :bedroom_type_id=>economica.id)
Interval.create(:price=>70, :name=>"Agosto",  :start=>Date.parse('01-08-2013'),
:end=>Date.parse('31-08-2013'), :bedroom_type_id=>economica.id)
Interval.create(:price=>60, :name=>"Septiembre",  :start=>Date.parse('01-09-2013'),
:end=>Date.parse('30-09-2013'), :bedroom_type_id=>economica.id)
Interval.create(:price=>50, :name=>"Temporada invierno", :start=>Date.parse('01-10-
2013'), :end=>Date.parse('31-12-2013'), :bedroom_type_id=>economica.id)

#Doble intervals
Interval.create(:price=>50, :name=>"Temporada baja", :start=>Date.parse('01-01-2013'),
:end=>Date.parse('31-05-2013'), :bedroom_type_id=>doble.id)
Interval.create(:price=>60, :name=>"Junio", :start=>Date.parse('01-06-2013'),
:end=>Date.parse('30-06-2013'), :bedroom_type_id=>doble.id)
Interval.create(:price=>70, :name=>"Julio", :start=>Date.parse('01-07-2013'),
:end=>Date.parse('31-07-2013'), :bedroom_type_id=>doble.id)
Interval.create(:price=>80, :name=>"Agosto", :start=>Date.parse('01-08-2013'),
:end=>Date.parse('31-08-2013'), :bedroom_type_id=>doble.id)
Interval.create(:price=>70, :name=>"Septiembre", :start=>Date.parse('01-09-2013'),
:end=>Date.parse('30-09-2013'), :bedroom_type_id=>doble.id)
Interval.create(:price=>60, :name=>"Temporada invierno", :start=>Date.parse('01-10-
2013'), :end=>Date.parse('31-12-2013'), :bedroom_type_id=>doble.id)

#Especial intervals
Interval.create(:price=>70, :name=>"Temporada baja", :start=>Date.parse('01-01-2013'),
:end=>Date.parse('31-05-2013'), :bedroom_type_id=>especial.id)
Interval.create(:price=>80, :name=>"Junio", :start=>Date.parse('01-06-2013'),
:end=>Date.parse('30-06-2013'), :bedroom_type_id=>especial.id)
Interval.create(:price=>90, :name=>"Julio", :start=>Date.parse('01-07-2013'),
:end=>Date.parse('31-07-2013'), :bedroom_type_id=>especial.id)
Interval.create(:price=>100, :name=>"Agosto",  :start=>Date.parse('01-08-2013'),
:end=>Date.parse('31-08-2013'), :bedroom_type_id=>especial.id)
Interval.create(:price=>90, :name=>"Septiembre",  :start=>Date.parse('01-09-2013'),
:end=>Date.parse('30-09-2013'), :bedroom_type_id=>especial.id)
Interval.create(:price=>80, :name=>"Temporada invierno", :start=>Date.parse('01-10-
2013'), :end=>Date.parse('31-12-2013'), :bedroom_type_id=>especial.id)


Bedroom.create(:name=>"La Buhardilla", :bedroom_type_id=>economica.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/buhardilla.jpg')))
Bedroom.create(:name=>"La Corralada",  :bedroom_type_id=>doble.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/lacorralada.jpg')))
Bedroom.create(:name=>"La Cuadra",     :bedroom_type_id=>doble.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/lacuadra.jpg')),
:handicapped=>true)
Bedroom.create(:name=>"La Tenada I",   :bedroom_type_id=>doble.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/tenada1.jpg')))
Bedroom.create(:name=>"La Tenada II",  :bedroom_type_id=>doble.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/tenada2.jpg')))
Bedroom.create(:name=>"El Desván",     :bedroom_type_id=>especial.id,
:main_photo=>File.open(File.join(Rails.root, '/public/seeds/bedrooms/eldesvan.jpg')))
```

Con esto obtenemos una base de datos con el contenido necesario para utilizar la aplicación, de manera rápida. Además podemos añadir el contenido que deseemos, como fotografías, etc.

Una vez tenemos el entorno y la base de datos, sólo falta ejecutar la aplicación. Nos colocamos en la carpeta del proyecto y ejecutamos la orden: **rails server**, por defecto se lanza en el puerto 3000, con el parámetro –p podemos indicarle otro puerto, por ejemplo: rails server –p 3333.

```
luis@sam:~/projects/sobrefuentes$ rails server
=> Booting WEBrick
=> Rails 3.2.11 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
/home/luis/.rvm/gems/ruby-1.9.2-p320@rails3/gems/actionpack-
3.2.11/lib/action_dispatch/http/mime_type.rb:102: warning: already initialized constant
PDF
[2013-07-15 21:32:22] INFO  WEBrick 1.3.1
```

```
[2013-07-15 21:32:22] INFO  ruby 1.9.2 (2012-04-20) [i686-linux]
[2013-07-15 21:32:22] INFO  WEBrick::HTTPServer#start: pid=7748 port=3000
```

Una vez arrancado el servidor podemos acceder a la aplicación en: http://localhost:3000.

Para pararlo, debemos colocarnos en la terminal y ejecutar: CTRL + C, o cerrar la terminal.

9.3 Manual del Programador

La aplicación ha sido desarrollada en su totalidad utilizando el IDE de desarrollo NetBeans 7.0.1. Se recomienda que para futuras modificaciones o revisiones se utilice una versión igual o superior.

Netbeans a partir de la versión 7.0 no soporta el lenguaje Ruby, para su integración se han añadido el siguientes *plugin*: http://plugins.netbeans.org/plugin/38549/ruby-and-rails además de este se integran *plugins* para reconocimiento sintáctico del lenguaje SASS http://plugins.netbeans.org/plugin/34929 y para copiar texto de código fuente a un documento de texto respetando la sintaxis del código fuente: http://plugins.netbeans.org/plugin/39977.

El proyecto sigue la siguiente estructura:

- Dentro de la carpeta *app/* se encuentra el núcleo de la aplicación:
  - Los **modelos** en la carpeta *app/models*
  - los **helpers** en la carpeta *app/helpers*
  - Los **controladores** en la carpeta *app/controllers*
  - Las vistas y plantillas HTML y ERB en la carpeta *app/views*
  - Los recursos para la vista como imágenes, librerías JavaScript y ficheros CSS, fuentes, etc. En la carpeta *app/assets*
- En la carpeta */config* se encuentran los archivos de configuración del sistema.
- En la carpeta */db* todo lo relacionado con la base de datos y migraciones.
- En la carpeta */spec* están las pruebas unitarias del proyecto.
- En el fichero *Gemfile* se declaran todas las gemas (librerías Ruby) a utilizar por la aplicación.

En la carpeta *vendor/plugins* se encuentran los programas de terceros incluidos en forma de *plugins* que añaden nuevas funcionalidades a la aplicación. Al instalarse siempre quedan instalados en esta carpet a.

Capítulo 10. Conclusiones y Ampliaciones

10.1 Conclusiones

Podemos concluir que se ha construido un sitio web con un desarrollo orientado a test (*TDD*) y con una interfaz de usuario enriquecida, en la que se integran multitud de tecnologías de desarrollo web: *AJAX*, *JSON*, *Response Design*, *SASS*, *CSS3* más la configuración e integración de varias librerías JavaScript.

Además de realizar un proyecto para un cliente real (que pronto será subido a un *hosting* con soporte para *Ruby on Rails*, probablemente una micro instancia en Amazon) a nivel personal he podido familiarizarme con las diferencias entre Ruby on Rails 2 y 3 ya que habitualmente en mi trabajo utilizo *Rails 2* por motivos de mantenibilidad de software.

Además de tener un producto para un cliente concreto, el proyecto es flexible y escalable, fácilmente se podría adaptar a otros establecimientos de similares características cambiando los contenidos de la parte pública.

10.2 Ampliaciones

A continuación comentare algunas posibles ampliaciones que se podrían realizar a partir del actual proyecto:

10.2.1 Pasarela de pago

Implementar una pasarela de pago a través de algún banco o de PayPal podría permitir solicitar al cliente un % del coste total de la reserva para formalizar la reserva de esta. O el pago de la totalidad de la reserva si así lo estiman oportuno los gestores del establecimiento.

Esta opción ha quedado fuera del alcance del proyecto por varios motivos:

- Aumentaría el coste temporal del proyecto más de lo estipulado.
- Implicaría modificar la actual política de devoluciones ya que si un cliente cancela una reserva no se le cobra nada. Si se hubiera solicitado un adelanto para la formalización de la reserva, una cancelación implicaría su devolución, ocasionando gasto de tiempo y dinero en comisiones para el dueño.
- Supondría unos costes adicionales tanto en cuota o alta de la pasarela de pago como en comisiones. Sería tarea del dueño de la casa rural decidir si le merece la pena o no asumir esos costes adicionales.

10.2.2 Exportar reservas en PDF o CSV

Actualmente se pueden generar y exportar a PDF las facturas asociadas a las reservas. Una posible ampliación sería automatizar la **exportación de la lista de reservas** pendientes a formato PDF o CSV para tenerlas guardadas en el disco duro ante un posible fallo de red. Ahora mismo podría hacerse guardando el contenido de la página web, pero resultaría más cómodo y limpio un sistema como el mencionado.

10.2.3 Importar reservas desde buzón de correo

En un negocio de estas características, es posible que las reservas lleguen a la casa rural por varias fuentes. Es importante estar al tanto de todas las reservas para evitar un overbooking.

Booking.com es uno de los principales proveedores de clientes para el establecimiento que se ha desarrollado el proyecto, Casa Rural Sobrefuentes. Tiene su propia base de datos y su disponibilidad.

Es tarea del dueño del establecimiento actualizar la ocupación de la base de datos de Booking.com si recibe una reserva a través de la página web de su establecimiento, y viceversa, si la reserva la recibe a través de *Booking*.

Para formalizar una reserva, *Booking* emplea el FAX como único medio de comunicación con el dueño del establecimiento. La idea sería:

- Enviar los fax entrantes a un **FAX** virtual. El fax virtual recogería el contenido del FAX, y generaría un correo electrónico automáticamente, con la información del FAX adjunta en un archivo en formato PDF. Esto ya está hecho por parte del cliente.
- Habría que implementar una **tarea que haga consultas periódicas al buzón** de correo que contiene los emails generados por FAX, obtenga los ficheros adjuntos y a través de reconocimiento OCR (aprovechando que todos los FAX tienen el mismo formato) **extraer los datos**, tanto de la reserva como del cliente.
- Una vez tengamos todos los datos, **se generaría una nueva reserva** en la base de datos del sitio web.

Con esto conseguiríamos una actualización casi inmediata de nuestra base de datos de reservas con las nuevas reservas entrantes por parte de Booking. Además ahorraríamos trabajo al gestor del establecimiento. Ha quedado fuera del alcance de proyecto ya que aumentaría el coste temporal del proyecto más allá de la fecha límite estipulada.

10.2.4 Módulo social

Ver **opiniones** de otros clientes en la página web inspira confianza a posibles visitantes, especialmente si son positivas. La Casa Rural Sobrefuentes es un establecimiento pequeño, con sólo 6 habitaciones, pero la gente que lo visita suele quedar con una buena impresión y suele recomendarlo. No en vano tiene una **puntuación de 9,1 sobre 10** basándose en la nota media de 27 opiniones, en el portal Booking.com:

http://www.booking.com/hotel/es/sobrefuentes-casa-de-aldea.en.html#hash-blockdisplay4

Sería aconsejable aprovechar que los usuarios tienen cuenta de usuario, para solicitarles opinión y publicarla en la web una vez finalizada su estancia. Podría ser mediante un correo automáticamente generado teniendo en cuenta la fecha de salida del establecimiento, o mediante alguna librería social. Ha quedado fuera del proyecto por falta de tiempo.

10.2.5 Despliegue a producción

Queda pendiente poner la página web en producción, en un servidor real para que los clientes puedan acceder a ella. El dominio sería www.sobrefuentes.es y sustituiría a la actual.

Para el despliegue se utilizará la librería ***Capistrano***: https://github.com/capistrano/capistrano y probablemente una micro instancia en *Amazon: http://aws.amazon.com/es/ec2* , donde tener libertad para instalar librerías necesarias, gestionar *backups* automáticas y tareas de *background* para implementar futuras ampliaciones.

Los propietarios del establecimiento están a la espera de una **subvención** para un **reportaje fotográfico profesional**, del cual se extraerán las fotos profesionales que ocuparán finalmente la web. Las actuales son temporales. Esto retrasará la definitiva subida a producción.

http://www.alojamientosconectados.es/turismo

10.2.6 Accesibilidad

Una posible ampliación sería conseguir un nivel AAA de accesibilidad en la web.

Capítulo 11. Presupuesto

Se ha utilizado software libre para el desarrollo a excepción de un Windows 7 y un Microsoft Office con licencia de estudiante, para realizar la documentación. El desarrollo se divide en días de análisis, diseño, implementación, pruebas y documentación.

Es el tiempo de días total estimado para cada apartado, pese a que en un mismo día se podían realizar tareas de cualquier parte del desarrollo.

El hardware se considera que se amortiza en un 25% en la realización de este proyecto. También se incluye gastos generales y de software, que en este caso al haber utilizado licencias de código libre, el coste es de 0€.

Para el cálculo de este presupuesto, se ha estimado una jornada laboral de 3 horas diarias durante toda la duración del proyecto, (días no laborables incluidos). A un precio de 12€/horas, sería 36€/día dedicado a cada apartado. Durante el desarrollo del proyecto, un 70% de la jornada se utiliza para realizar la planificación de ese día, un 20% a hacer pruebas y un 10% a hacer la documentación. Finalmente ha quedado un presupuesto de 15.104,43 € para el desarrollo del proyecto que se estima que tenga una duración de 169 días.

Item	Subitem	Concepto	Cantidad (días)	Precio unitario	Total
1	1	**Desarrollo de la aplicación**			10.476,00 €
	1	Análisis	12	36,00 €	432,00 €
	2	Diseño	15	36,00 €	540,00 €
	3	Implementación	86	36,00 €	3.096,00 €
	4	Simulación y Pruebas	88	36,00 €	3.168,00 €
	5	Documentación	80	36,00 €	2.880,00 €
	6	Subida a producción	10	36,00 €	360,00 €
2		**Formación**			900,00 €
	1	Rol Programador	15	36,00 €	540,00 €
	2	Rol diseñador	10	36,00 €	360,00 €
					- €
3		**Licencias software**			- €
	1	Sistema operativo	1	- €	- €
	2	IDE Desarrollo	1	- €	- €
					- €
4		**Otros conceptos**			1.107,00 €
	1	Proveedor Internet	169	3,00 €	507,00 €
	2	Amortización hardware	2	300,00 €	600,00 €
	3	Material oficina			- €
				Subtotal	**12.483,00 €**
				IVA (21%)	**2.621,43 €**
				TOTAL	**15.104,43 €**

Figura 11.1 Cuadro presupuesto

Capítulo 12. Referencias Bibliográficas

12.1 Libros y Artículos

Durante el desarrollo del proyecto se han consultado los siguientes libros:

The Ruby programming language
David Flanagan, Yukihiro Matsumoto
O'Reilly Media, 2008

The Rails Way
Obi Fernandez
Addison-Wesley, 2007

Javascript: The good parts
Douglas Crockford
O'Reilly Media, 2008

Ejemplo para la plantilla de TFM
Redondo L., J
Universidad de Oviedo, 2007

12.2 Referencias en Internet

Páginas Web consultadas para cualquier aspecto relacionado con el desarrollo del sistema o su documentación.

A lo largo del desarrollo se han consultado dudas de desarrollo y teóricas en infinidad de ocasiones. Resultaría casi imposible tener un registro de todas ellas, a continuación nombro algunas de las páginas que más útiles me han resultado.

[StackOverflow] Es una comunidad de desarrolladores informáticos, en la cual otros desarrolladores pueden encontrar soluciones a problemas de programación en diferentes lenguajes. La he utilizado para la consulta de diferentes dudas, tanto consultas para el desarrollo de código de servidor Ruby, como de integración de librerías JavaScript y gemas de Ruby. http://stackoverflow.com.

[Api dock] Página de consulta de *helpers* para la vista de formularios desarrollados con Ruby on Rails (*checkboxes, labels, selects, radio buttons*, formularios, etc.) http://apidock.com/rails.

[Ruby doc] Documentación de desarrollo Ruby puro, sin interacción con el framework Ruby on Rails http://ruby-doc.org/core-1.9.3 .

[RubyForge] Pagina de proyectos Ruby http://rubyforge.org .

[RailsGuides] Guías, tutoriales y documentación sobre Ruby on Rails http://guides.rubyonrails.org.

[RubyGems] Guías y documentación sobre gemas para Rails http://rubygems.org.

[Git Hub] Repositorio de código y documentación para librerías, tanto JavaScript como gemas de Rails. En la sección 7.3 se detallan las librerías consultadas de forma más concreta. https://github.com .

[Jquery.com] Documentación y ejemplos de la librería jQuery, utilizada en la mayor parte del código JavaScript implementado. http://api.jquery.com.

Webs de inspiración y ejemplos para el diseño de la página:

- **[Codrops]** Web de recursos HTML5, CSS3, ejemplos JavaScript y diseños http://tympanus.net/codrops
- **[Colour Lovers]** Web de diseño y elección de colores para web http://www.colourlovers.com
- **[Themeforest]** Web de ejemplos prácticos de plantillas http://themeforest.net
- **[Elegant themes]** Web de ejemplos prácticos de plantillas http://www.elegantthemes.com

Capítulo 13. Apéndices

13.1 Contenido Entregado en el CD-ROM

13.1.1 Contenidos

Los directorios del CD se estructuran de la siguiente manera:

- Una carpeta */proyecto* con todo el código fuente original del proyecto.
- Una carpeta */documentación*, con este documento en varios formatos así como una hoja de cálculo con el presupuesto y la planificación temporal. Dentro tendrá una subcarpeta con las imágenes utilizadas en el proyecto.

de directorios de ejemplo para el CD que se puede seguir para distribuir todos sus contenidos en el mismo. Conviene por tanto tenerla en cuenta desde el principio de la implementación.

13.1.1.1.1 Estructura general directorios del CD

Directorio	Contenido
./ Directorio raíz del CD	Contiene un fichero leeme.txt explicando toda esta estructura.
./proyecto/Sobrefuentes_tfm	Contiene toda la estructura de directorios y código fuente del proyecto.
./documentación	Contiene toda la documentación asociada al proyecto en formato PDF y .docx. También se incluye el presupuesto en formato Excel y el diagrama Gantt en formato .mpp
./documentacion/img	Directorio que contiene las imágenes utilizadas en la documentación.

13.2 Índice Alfabético

J

L

M

N

P

R

S

T

U

V

W

13.3 Código Fuente

Debido al gran número de archivos que genera el *framework*, el código fuente completo está disponible en la carpeta /proyecto del CD. A continuación solamente se muestra el código los controladores, los modelos, algunos archivos de configuración y el schema.rb cuya estructura se corresponde con la estructura de la base de datos de la aplicación.

13.3.1 Controladores (/app/controllers)

En orden alfabético por jerarquía de carpetas:

13.3.1.1 Admin Controller (admin/admin_controller.rb):

```
class Admin::AdminController < ApplicationController
  layout "admin_panel"

  before_filter :verify_admin

  def verify_admin
    :authenticate_user!
    redirect_to root_url unless has_role?(current_user, 'admin')
  end

  def current_ability
    @current_ability ||= AdminAbility.new(current_user)
  end
end
```

13.3.1.2 User Controller (admin/user_controller.rb):

```
# encoding: UTF-8
class Admin::UsersController < Admin::AdminController

  load_and_authorize_resource

  # GET /users
  # GET /users.json
  def index
    @users = User.all

    respond_to do |format|
      format.html{ render :layout => "admin_panel" } # index.html.erb
      format.json { render :json => @users }
    end
  end

  # GET /users/1
  # GET /users/1.json
  def show
    @user = User.find(params[:id])

    respond_to do |format|
      format.html { render :layout => "admin_panel"}
      format.json { render :json => @user }
    end
  end

  # GET /users/new
  # GET /users/new.json
  def new
    @user = User.new
    respond_to do |format|
```

```
      format.html # new.html.erb
      format.json { render :json => @user }
    end
  end

  # GET /users/1/edit
  def edit
    @user = User.find(params[:id])

  end

  # POST /users
  # POST /users.json
  def create
    @user.attributes = params[:user]
    @user.role_ids = params[:user][:role_ids] if params[:user]
    @user = User.new(params[:user])

    respond_to do |format|
      if @user.save
        flash[:notice] = flash[:notice].to_a.concat @user.errors.full_messages
        format.html { redirect_to admin_users_path, :notice => 'User was successfully
created.', :layout=>"admin_panel" }
        format.json { render :json => @user, :status => :created, :location => @user }
      else
        flash[:notice] = flash[:notice].to_a.concat @user.errors.full_messages
        format.html { render :action => "new", :layout=>"admin_panel"}
        format.json { render :json => @user.errors, :status => :unprocessable_entity }
      end
    end
  end

  # PUT /users/1
  # PUT /users/1.json
  def update
    @user = User.find(params[:id])
    if params[:user][:password].blank?
      params[:user].delete(:password)
      params[:user].delete(:password_confirmation)
    end

    respond_to do |format|
      if @user.update_attributes(params[:user])
        format.html { redirect_to admin_users_path, :notice => 'User was successfully
updated.', :layout=>"admin_panel" }
        format.json { head :ok }
      else
        format.html { render :action => "edit" , :layout=>"admin_panel"}
        format.json { render :json => @user.errors, :status => :unprocessable_entity }
      end
    end
  end

  # DELETE /users/1
  # DELETE /users/1.json
  def destroy
    @user = User.find(params[:id])

    if @user.destroy

      respond_to do |format|
        format.html { redirect_to admin_users_url, :notice => 'Usuario eliminado.'  }
        format.json { head :ok }
      end
    else
      respond_to do |format|
        format.html { redirect_to admin_users_url, :alert => 'No es posible eliminar
este usuario, podría tener reservas asociadas.' }
        format.json { head :ok }
      end
    end
  end
end
```

13.3.1.3 Application Controller (application_controller.rb)

```
class ApplicationController < ActionController::Base

  layout :layout_by_resource

  before_filter :set_locale

  protect_from_forgery

  rescue_from CanCan::AccessDenied do |exception|
    if current_user
      redirect_to user_root_path, :alert => exception.message
    else
      redirect_to root_url, :alert => exception.message
    end
  end

  def has_role?(current_user, role)
    return !!current_user.roles.find_by_name(role.to_s.camelize)
  end

  def set_locale
    I18n.locale = params[:locale] || I18n.default_locale
  end


  protected
  def layout_by_resource

    if controller_name == 'registrations' && action_name == 'edit'
      return  "admin_panel"
    elsif controller_name == 'registrations' && action_name == 'update'
      return  'admin_panel'
      #    elsif controller_name == 'registrations' && action_name == 'create'
      #      return 'devise_login'
    elsif controller_name == 'sessions' && action_name == 'new'
      return 'devise_login'
    end

    if devise_controller?
      "devise_login"
    else
      "admin_panel"
    end
  end


  private
  # Overwriting the sign_out redirect path method
  def after_sign_out_path_for(resource_or_scope)
    root_path
  end

end
```

13.3.1.4 Bedroom Types Controller

```
class BedroomTypesController < ApplicationController

  load_and_authorize_resource

  # GET /bedroom_types
  # GET /bedroom_types.json
  def index
    @bedroom_types = BedroomType.all
    @intervals = Interval.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @bedroom_types }
    end
  end

  # GET /bedroom_types/1
  # GET /bedroom_types/1.json
  def show
    @bedroom_type = BedroomType.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @bedroom_type }
    end
  end

  # GET /bedroom_types/new
  # GET /bedroom_types/new.json
  def new
    @bedroom_type = BedroomType.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @bedroom_type }
    end
  end

  # GET /bedroom_types/1/edit
  def edit
    @bedroom_type = BedroomType.find(params[:id])
  end

  # POST /bedroom_types
  # POST /bedroom_types.json
  def create
    @bedroom_type = BedroomType.new(params[:bedroom_type])

    respond_to do |format|
      if @bedroom_type.save
        format.html { redirect_to @bedroom_type, notice: 'Bedroom type was successfully
created.' }
        format.json { render json: @bedroom_type, status: :created, location:
@bedroom_type }
      else
        format.html { render action: "new" }
        format.json { render json: @bedroom_type.errors, status: :unprocessable_entity }
      end
    end
  end

  # PUT /bedroom_types/1
  # PUT /bedroom_types/1.json
  def update
    @bedroom_type = BedroomType.find(params[:id])

    respond_to do |format|
      if @bedroom_type.update_attributes(params[:bedroom_type])
        format.html { redirect_to @bedroom_type, notice: 'Bedroom type was successfully
updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @bedroom_type.errors, status: :unprocessable_entity }
```

```
      end
    end
  end

  # DELETE /bedroom_types/1
  # DELETE /bedroom_types/1.json
  def destroy
    @bedroom_type = BedroomType.find(params[:id])
    @bedroom_type.destroy

    respond_to do |format|
      format.html { redirect_to bedroom_types_url }
      format.json { head :no_content }
    end
  end
end
```

13.3.1.5 Bedrooms Controller

```
class BedroomsController < ApplicationController

  load_and_authorize_resource

  # GET /bedrooms
  # GET /bedrooms.json
  def index
    @bedrooms = Bedroom.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @bedrooms }
    end
  end

  # GET /bedrooms/1
  # GET /bedrooms/1.json
  def show
    @bedroom = Bedroom.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @bedroom }
    end
  end

  # GET /bedrooms/new
  # GET /bedrooms/new.json
  def new
    @bedroom = Bedroom.new
    @bedroom_types = BedroomType.all
    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @bedroom }
    end
  end

  # GET /bedrooms/1/edit
  def edit
    @bedroom = Bedroom.find(params[:id])
    @bedroom_types = BedroomType.all
  end

  # POST /bedrooms
  # POST /bedrooms.json
  def create
    @bedroom = Bedroom.new(params[:bedroom])
    @bedroom_types = BedroomType.all

    respond_to do |format|
      if @bedroom.save
        format.html { redirect_to @bedroom, notice: 'Bedroom was successfully created.'
```

```
}
        format.json { render json: @bedroom, status: :created, location: @bedroom }
      else
        format.html { render action: "new" }
        format.json { render json: @bedroom.errors, status: :unprocessable_entity }
      end
    end
  end

  # PUT /bedrooms/1
  # PUT /bedrooms/1.json
  def update
    @bedroom = Bedroom.find(params[:id])

    respond_to do |format|
      if @bedroom.update_attributes(params[:bedroom])
        format.html { redirect_to @bedroom, notice: 'Bedroom was successfully updated.'
}
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @bedroom.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /bedrooms/1
  # DELETE /bedrooms/1.json
  def destroy
    @bedroom = Bedroom.find(params[:id])
    @bedroom.destroy

    respond_to do |format|
      format.html { redirect_to bedrooms_url }
      format.json { head :no_content }
    end
  end


end
```

13.3.1.6 Intervals Controller

```
class IntervalsController < ApplicationController

  load_and_authorize_resource

  # GET /intervals
  # GET /intervals.json
  def index
    @intervals = Interval.all


    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @intervals }
    end
  end

  # GET /intervals/1
  # GET /intervals/1.json
  def show
    @interval = Interval.find(params[:id])

    respond_to do |format|
      format.html { render :layout => "admin_panel"}
      format.json { render json: @interval }
    end
  end

  # GET /intervals/new
```

```
  # GET /intervals/new.json
  def new
    @interval = Interval.new
    @bedroom_types = BedroomType.all
    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @interval }
    end
  end

  # GET /intervals/1/edit
  def edit
    @interval = Interval.find(params[:id])
    @bedroom_types = BedroomType.all
  end

  # POST /intervals
  # POST /intervals.json
  def create
    parse_dates
    @interval = Interval.new(params[:interval])
    @bedroom_types = BedroomType.all
    respond_to do |format|
      if @interval.save
        format.html { redirect_to @interval, notice: 'Interval was successfully
created.' }
        format.json { render json: @interval, status: :created, location: @interval }
      else
        format.html { render action: "new" }
        format.json { render json: @interval.errors, status: :unprocessable_entity }
      end
    end
  end

  # PUT /intervals/1
  # PUT /intervals/1.json
  def update
    parse_dates
    @interval = Interval.find(params[:id])
    @bedroom_types = BedroomType.all
    respond_to do |format|
      if @interval.update_attributes(params[:interval])
        format.html { redirect_to @interval, notice: 'Interval was successfully
updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @interval.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /intervals/1
  # DELETE /intervals/1.json
  def destroy
    @interval = Interval.find(params[:id])
    @interval.destroy

    respond_to do |format|
      format.html { redirect_to intervals_url }
      format.json { head :no_content }
    end
  end



  def parse_dates
    params[:interval][:start] &&= Date.parse(params[:interval][:start])
    params[:interval][:end] &&= Date.parse(params[:interval][:end])
  end


end
```

13.3.1.7 Pages Controller

```
class PagesController < ApplicationController

  before_filter :authenticate_user!, :except => [:index, :show, :rooms, :rooms2,
:contact, :availability, :gallery, :about, :gallery_full_screen ]

  load_and_authorize_resource  :except => [:index, :show, :rooms, :rooms2, :contact,
:availability, :gallery, :about, :gallery_full_screen ]

  def index

    @economica = BedroomType.find(1)
    @doble = BedroomType.find(2)
    @especial = BedroomType.find(3)

    @buhardilla = Bedroom.find(1)
    @tenada = Bedroom.find(4)
    @desvan = Bedroom.find(6)

    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def rooms2
    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def contact
    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def gallery
    @photos = Photo.all

    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def about
    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end
  def gallery_full_screen
    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def rooms
    @available_bedrooms = Bedroom.all.length
    @reservations = Reservation.all

    @economica = BedroomType.find(1)
    @doble = BedroomType.find(2)
    @especial = BedroomType.find(3)

    @buhardilla = Bedroom.find(1)
    @tenada = Bedroom.find(4)
    @desvan = Bedroom.find(6)
    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

  def availability
    @available_bedrooms = Bedroom.all.length
    @reservations = Reservation.all
    @bedrooms = Bedroom.all
```

```
    @bedroom_types = BedroomType.all


    respond_to do |format|
      format.html { render :layout => "public_page"}
    end
  end

end
```

13.3.1.8 Partners Controller

```
class PartnersController < ApplicationController

  load_and_authorize_resource


  # GET /partners
  # GET /partners.json
  def index
    @partners = Partner.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @partners }
    end
  end

  # GET /partners/1
  # GET /partners/1.json
  def show
    @partner = Partner.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @partner }
    end
  end

  # GET /partners/new
  # GET /partners/new.json
  def new
    @partner = Partner.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @partner }
    end
  end

  # GET /partners/1/edit
  def edit
    @partner = Partner.find(params[:id])
  end

  # POST /partners
  # POST /partners.json
  def create
    @partner = Partner.new(params[:partner])

    respond_to do |format|
      if @partner.save
        format.html { redirect_to @partner, notice: 'Partner was successfully created.'
}
        format.json { render json: @partner, status: :created, location: @partner }
      else
        format.html { render action: "new" }
        format.json { render json: @partner.errors, status: :unprocessable_entity }
      end
    end
  end
```

```
  # PUT /partners/1
  # PUT /partners/1.json
  def update
    @partner = Partner.find(params[:id])

    respond_to do |format|
      if @partner.update_attributes(params[:partner])
        format.html { redirect_to @partner, notice: 'Partner was successfully updated.'
}
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @partner.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /partners/1
  # DELETE /partners/1.json
  def destroy
    @partner = Partner.find(params[:id])
    @partner.destroy

    respond_to do |format|
      format.html { redirect_to partners_url }
      format.json { head :no_content }
    end
  end
end
```

13.3.1.9 Photos Controller

```
class PhotosController < ApplicationController

  load_and_authorize_resource


  # GET /photos
  # GET /photos.json
  def index
    @photos = Photo.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @photos }
    end
  end

  # GET /photos/1
  # GET /photos/1.json
  def show
    @photo = Photo.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @photo }
    end
  end

  # GET /photos/new
  # GET /photos/new.json
  def new
    @photo = Photo.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @photo }
    end
  end
```

```
  # GET /photos/1/edit
  def edit
    @photo = Photo.find(params[:id])
  end

  # POST /photos
  # POST /photos.json
  def create
    @photo = Photo.new(params[:photo])

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

  # PUT /photos/1
  # PUT /photos/1.json
  def update
    @photo = Photo.find(params[:id])

    respond_to do |format|
      if @photo.update_attributes(params[:photo])
        format.html { redirect_to @photo, notice: 'Photo was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @photo.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /photos/1
  # DELETE /photos/1.json
  def destroy
    @photo = Photo.find(params[:id])
    @photo.destroy

    respond_to do |format|
      format.html { redirect_to photos_url }
      format.json { head :no_content }
    end
  end
end
```

13.3.1.10 Profiles Controller

```
class ProfilesController < ApplicationController
  before_filter :authenticate_user!

  def dashboard
    @user = User.find(current_user.id)
    if @user.role? :admin
      redirect_to admin_users_path, :notice => "Bienvenido Admin!"
    else
      respond_to do |format|
        format.html # dashboard.html.erb
      end

    end
  end

end
```

13.3.1.11 Reservations Controller

```
class ReservationsController < ApplicationController

  load_and_authorize_resource :except => [:reservation_nights, :reservation_user,
:reservation_confirm, :reservation_final]


  before_filter :authenticate_user!, :except => [:reservation_nights, :reservation_user,
:reservation_confirm, :reservation_final]


  def reservation_nights
    @reservation = Reservation.new

    begin
      @reservation[:check_in]= Date.parse(params[:checkin])
      @reservation[:check_out]= Date.parse(params[:checkout])

      @same_date = @reservation.same_date
      @inverse_dates = @reservation.inverse_dates
      @back_to_past = @reservation.is_past
      @no_bedrooms = !(@reservation.available_bedrooms_for_new_reservation(1))

    rescue Exception
      # handle everything else
      @error_format_date = true
    end



    @has_errors = @error_format_date || @same_date || @inverse_dates || @back_to_past ||
@no_bedrooms

    @bedrooms = Bedroom.all
    @bedroom_types = BedroomType.all

    respond_to do |format|
      format.html{ render :layout => false }
    end
  end


  def reservation_user
    @reservation = Reservation.new

    begin
      @reservation[:check_in]= Date.parse(params[:reservation][:check_in] )
      @reservation[:check_out]= Date.parse(params[:reservation][:check_out])


      @same_date = @reservation.same_date
      @inverse_dates = @reservation.inverse_dates
      @back_to_past = @reservation.is_past
      @no_bedrooms = !(@reservation.available_bedrooms_for_new_reservation(1))

    rescue Exception
      # handle everything else
      @error_format_date = true
    end

    @has_errors = @error_format_date || @same_date || @inverse_dates || @back_to_past ||
@no_bedrooms


    @double_bedrooms = params[:rooms][:double_rooms]
    @single_bedrooms = params[:rooms][:single_rooms]
    @user = User.new

    respond_to do |format|
      format.html{ render :layout => false }
    end
  end

  def reservation_confirm
    @reservation = Reservation.new
```

```
    @reservation[:check_in]= Date.parse(params[:reservation][:check_in])
    @reservation[:check_out]= Date.parse(params[:reservation][:check_out])

    @double_bedrooms = params[:rooms][:double_rooms]
    @single_bedrooms = params[:rooms][:single_rooms]


    if params[:user][:id]
      @user = User.find(params[:user][:id])
      respond_to do |format|
        format.html{ render :layout => false }
      end
    else
      @user = User.new
      @user.name = params[:user][:name]
      @user.surname = params[:user][:surname]
      @user.email = params[:user][:email]
      @user.telephone = params[:user][:telephone]
      @user.password = params[:user][:password]
      @user.password_confirmation = params[:user][:password_confirmation]

      if @user.save
        #   throw emails
        UserMailer.registration_confirmation_to_admin(@user).deliver
        UserMailer.registration_confirmation_to_user(@user).deliver

        respond_to do |format|
          format.html{ render :layout => false }
        end
      else
        render "reservations/reservation_user", :layout=>false
      end

    end
  end

  def reservation_final

    @reservation = Reservation.new

    @double_bedrooms = params[:rooms][:double_rooms]
    @single_bedrooms = params[:rooms][:single_rooms]

    @reservation.check_in = Date.parse(params[:reservation][:check_in])
    @reservation.check_out = Date.parse(params[:reservation][:check_out])
    @user = User.find(params[:user][:id])
    @reservation.user = @user
    @reservation.asign_bedrooms_for_reservation(@single_bedrooms, @double_bedrooms)


    if @reservation.save
      # lanzar email

      UserMailer.reservation_confirmation_to_admin(@reservation).deliver
      UserMailer.reservation_confirmation_to_user(@reservation).deliver

      respond_to do |format|
        format.html{ render :layout => false }
      end
    else
      render "reservations/reservation_confirm", :layout=>false
    end
  end



  # GET /reservations
  # GET /reservations.json
  def index
    @reservations = Reservation.all.select{ |r| r.check_out.future?}.sort_by(&:check_in)

    respond_to do |format|
      format.html{ render :layout => "admin_panel" } # index.html.erb
      format.json { render json: @reservations }
    end
  end
```

```
  # GET /reservations/1
  # GET /reservations/1.json
  def show
    @reservation = Reservation.find(params[:id])

    respond_to do |format|
      format.html{ render :layout => "admin_panel" }
      format.pdf do
        render :pdf =>
(@reservation.id+9000).to_s+"_date_"+(I18n.l(@reservation.check_out, :format=> :file)),
          :template => 'reservations/bill.pdf.erb'
      end

      #       format.json { render json: @reservation }
    end
  end




  # GET /reservations/new
  # GET /reservations/new.json
  def new
    @reservation = Reservation.new

    @bedrooms = Bedroom.all
    @users = User.all
    @partners = Partner.all

    respond_to do |format|
      format.html{ render :layout => "admin_panel" } # new.html.erb
      format.json { render json: @reservation }
    end
  end

  # GET /reservations/1/edit
  def edit
    @reservation = Reservation.find(params[:id])

    @bedrooms = Bedroom.all
    @users = User.all
    @partners = Partner.all

  end

  # POST /reservations
  # POST /reservations.json
  def create
    @reservation = Reservation.new(params[:reservation])

    @bedrooms = Bedroom.all
    @users = User.all
    @partners = Partner.all

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

  # PUT /reservations/1
  # PUT /reservations/1.json
  def update
    @reservation = Reservation.find(params[:id])
    @bedrooms = Bedroom.all
    @users = User.all
    @partners = Partner.all
    respond_to do |format|
      if @reservation.update_attributes(params[:reservation])
```

```
        format.html { redirect_to @reservation, notice: 'Reservation was successfully
updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @reservation.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /reservations/1
  # DELETE /reservations/1.json
  def destroy
    @reservation = Reservation.find(params[:id])
    if @reservation.destroy

      respond_to do |format|
        format.html { redirect_to reservations_url, :notice=>'La reserva ha sido
eliminada. Reservation was successfully removed.' }
        format.json { head :no_content }
      end
    else
      format.html { redirect_to reservations_url, :notice=>'No se ha podido eliminar la
reserva.' }
    end
  end
end
```

13.3.2 Modelos (app/models)

13.3.2.1 Ability

```
class Ability
  include CanCan::Ability

  def initialize(user)
    user ||= User.new # guest user
    if user.role? :admin
      can :manage, :all
    elsif user.role? :member
      can :manage, User do |user|
        user.user_id == user.id
      end
    else
      can :read, :all
    end
    # Define abilities for the passed in user here. For example:
    #
    #   user ||= User.new # guest user (not logged in)
    #   if user.admin?
    #     can :manage, :all
    #   else
    #     can :read, :all
    #   end
    #
    # The first argument to `can` is the action you are giving the user permission to
do.
    # If you pass :manage it will apply to every action. Other common actions here are
    # :read, :create, :update and :destroy.
    #
    # The second argument is the resource the user can perform the action on. If you
pass
    # :all it will apply to every resource. Otherwise pass a Ruby class of the resource.
    #
    # The third argument is an optional hash of conditions to further filter the
objects.
    # For example, here the user can only update published articles.
    #
```

```
    #   can :update, Article, :published => true
    #
    # See the wiki for details: https://github.com/ryanb/cancan/wiki/Defining-Abilities
  end
end
```

13.3.2.2 Admin Ability

```
class AdminAbility

  include CanCan::Ability

  def initialize(user)
    if user.role? :admin
      can :manage, :all
    end
  end
end
```

13.3.2.3 Bedroom

```
class Bedroom < ActiveRecord::Base

  has_many :bedroom_reservation
  has_many :reservations, :through => :bedroom_reservation

  #  has_and_belongs_to_many :reservations
  belongs_to :bedroom_type
  has_many :photos

  attr_accessible :name, :bedroom_type_id, :bedroom_type
  attr_accessible :supports_extra_bed, :tv, :parking, :wifi, :bath, :breakfast,
:handicapped
  attr_accessible :main_photo,  :main_photo_file_name, :main_photo_content_type,
:main_photo_file_size, :main_photo_updated_at

  has_attached_file :main_photo,
    :styles => {
    :thumb  => "130x70>",
    :home => "220>"
  }

  #  http://lindsaar.net/2010/1/31/validates_rails_3_awesome_is_true
  #  http://lindsaar.net/2010/1/31/validates_rails_3_awesome_is_true
  #  http://guides.rubyonrails.org/active_record_validations_callbacks.html#working-
with-validation-
errorshttp://guides.rubyonrails.org/active_record_validations_callbacks.html#working-
with-validation-errors


  validates :name, :presence => {:message => "Es necesario un nombre."}, :length => {
:minimum => 3 }, :uniqueness => { :case_sensitive => false }, :allow_blank => false


    validate(:name, :presence => true, :message => "Es necesario un nombre.")
    validates :name, :uniqueness =>{  :message => "El nombre debe ser único."}
    validate(:bedroom_type, :presence => true, :message => "Debe tener un tipo de
habitación.")



end
```

13.3.2.4 Bedroom Reservation

```
class BedroomReservation < ActiveRecord::Base

  belongs_to :reservation
  belongs_to :bedroom

  attr_accessible :assigned_at, :bedroom, :reservation, :bedroom_id, :num_guest
end
```

13.3.2.5 Bedroom Type

```
class BedroomType < ActiveRecord::Base

  has_many :intervals

  attr_accessible :name, :price, :bedroom_id, :individual_difference

  def get_range_price(checkin, checkout, individual)
    final_price = 0

    intervals_for_date = self.intervals.all.select{ |i| ((i.start <= checkin) &&
(checkout <=  i.end))}

    intervals_for_date.each do |interval|
      checkin.upto(checkout.yesterday) do |date|
        if(interval.contain_day(date))
          if individual>=2
          final_price += interval.price
          else
            final_price += (interval.price - self.individual_difference)
          end
        end
      end
    end
    return final_price.to_i
  end

end
```

13.3.2.6 Interval

```
#!/bin/env ruby
# encoding: utf-8
#
class Interval < ActiveRecord::Base

  belongs_to :bedroom_type

  attr_accessible :price, :name, :start, :end, :bedroom_type_id

  validates :price, :presence => true
  validates :name, :presence => true, :length => {:minimum => 3, :maximum => 24}

  validates :start, :presence => true
  validates :end,   :presence => true

  validate :interval_should_be_unique
  validate :unique_name_for_interval
  validate :right_dates

  def right_dates
    errors.add(:end, "el fin del intervalo debe ser despues del comienzo") if (self.end
```

```
< self.start)
  end

  def unique_name_for_interval
    Interval.all.select{ |i|  i.bedroom_type_id==self.bedroom_type_id  }.each do
|interval|
      if (interval.name==self.name && interval!=self)
        errors.add(:name, "nombre de intervalo ya existente para este tipo de
habitación")
      end
    end
  end

  def interval_should_be_unique
    end_inside = false
    start_inside= false

    Interval.all.select{ |i| i.end.future? && i.bedroom_type_id==self.bedroom_type_id
}.each do |interval|
      start_inside = (interval.start < self.start) && (self.start < interval.end)
      end_inside = (interval.start < self.end) && (self.end < interval.end)
    end

    errors.add(:end, " hay intervalos pisandose la fecha de inicio para el mismo tipo de
habitacion")  if start_inside
    errors.add(:end, " hay intervalos pisandose la fecha de fin para el mismo tipo de
habitacion")  if end_inside

    return true
  end


  def contain_day(day)
#    Interval.all.each do |interval|
      # if((interval.start..interval.end).cover?(day))
      if((self.start <= day) && (day <= self.end))
        return true
      end
#    end
    return false;
  end


end
```

13.3.2.7 Partner

```
class Partner < ActiveRecord::Base

  has_many :reservation


  attr_accessible :logo, :logo_content_type, :logo_file_name, :logo_file_size,
:logo_updated_at, :name, :telephone, :website

  has_attached_file :logo,
    :styles => {
    :thumb  => "150x150>",
    :medium => "300x300>"
  }

end
```

13.3.2.8 Photo

```
class Photo < ActiveRecord::Base

  attr_accessible  :image, :image_content_type, :image_file_name, :image_file_size,
:image_updated_at, :name

  has_attached_file :image,
    :styles => {
    :thumb  => "130x70>",
    :gallery => "220>x140",
    :home => "640x640>",
    :full_gallery => "1900>x1200>"
  }

  validates_attachment_size :image, :less_than => 5.megabytes

end
```

13.3.2.9 Reservation

```
class Reservation < ActiveRecord::Base

  attr_accessible :user_id, :user, :bedrooms, :bedroom_ids, :partner_id, :check_in,
:check_out , :bedroom_reservation

  has_many :bedroom_reservation
  has_many :bedrooms, :through => :bedroom_reservation

  belongs_to :user
  belongs_to :partner

  accepts_nested_attributes_for :bedroom_reservation

  validates_associated  :bedrooms
  validates :check_in,  :presence => true
  validates :check_out, :presence => {:message => 'hace falta checkout.'}
  validates :bedroom_reservation, :length => { :minimum => 1 , :message => 'Debe
seleccionar al menos una habitacion.'}
  validate :right_dates
  validate :reservation_format

  def right_dates
    errors.add(:check_out, "la salida no puede ser antes que la entrada") if
(self.nights <= 0)
  end


  def nights
    (self.check_out - self.check_in).to_i
  end

  def reservation_format
    errors.add(:bedrooms, "No se pueden reservas mas habitaciones de las existentes.")
if self.bedroom_ids.length > Bedroom.all.length
    errors.add(:bedrooms,"No se pueden reservar habitaciones dos veces.") if
self.bedroom_ids != self.bedroom_ids.uniq
  end

  def price_for_type(bt, individual)
    bt.get_range_price(self.check_in, self.check_out, individual)
  end

  def price
    price = 0
    self.bedroom_reservation.each do |bed|
      bedroom = Bedroom.find(bed.bedroom_id)
      price += bedroom.bedroom_type.get_range_price(self.check_in, self.check_out,
bed.num_guest)
    end
```

```
    price
  end

  def available_bedrooms_for_new_reservation(n_bedrooms)
    self.check_in.upto(self.check_out.yesterday) do |day|
      if(available_bedrooms(day) < n_bedrooms)
        return false
      end
    end
    return true
  end

  def available_bedrooms(day)
    total_rooms = Bedroom.all.length
    rooms_availables = 0
    Reservation.all.select{ |r| r.check_out.future? }.each{ |r|
      if((r.check_in..r.check_out.yesterday).cover?(day))
        rooms_availables += r.bedrooms.length
      end
    }
    total_rooms-rooms_availables
  end

  def available_bed_types_for_new_reservation(n_bedrooms, bedroom_type)
    max_bedrooms_availables = 6
    self.check_in.upto(self.check_out.yesterday) do |day|
      free_beds = available_bedrooms_for_type(day, bedroom_type)
      max_bedrooms_availables = [max_bedrooms_availables, free_beds].min
      if(free_beds < n_bedrooms)
        return [false, max_bedrooms_availables]
      end
    end
    return [true, max_bedrooms_availables]
  end

  def available_bedrooms_for_type(day, b_type)
    total_bedroom_type_beds = Bedroom.all.select{|b| b.bedroom_type==b_type}.length
    rooms_busy = 0

    pending_bookings_with_my_bedroom_type = Reservation.all.select{ |r|
r.check_out.future? && r.has_bedroom_type(b_type) }

    pending_bookings_with_my_bedroom_type.each{ |r|
      if((r.check_in..r.check_out.yesterday).cover?(day))
        rooms_busy += r.bedrooms.length
      end
    }
    total_bedroom_type_beds-rooms_busy
  end

  def has_bedroom_type(b_type)
    self.bedrooms.each do |bed|
      if bed.bedroom_type== b_type
        return true
      end
    end
    return false
  end


  def asign_bedrooms_for_reservation(single_bedrooms, double_bedrooms)
    all_rooms = Bedroom.all

    singles = single_bedrooms.map do |type, num|
      single_beds = all_rooms.select{ |b|
(b.bedroom_type.id==type.to_i)}.first(num.to_i)
      single_beds.each {|s| all_rooms.delete(s)}
    end

    doubles = double_bedrooms.map do |type, num|
      double_beds = all_rooms.select{ |b|
(b.bedroom_type.id==type.to_i)}.first(num.to_i)
      double_beds.each {|s| all_rooms.delete(s)}
    end

    s = singles.flatten.map{|x| {:num_guest => 1, :bedroom_id => x.id}}
    d = doubles.flatten.map{|y| {:num_guest => 2, :bedroom_id => y.id}}
```

```
    parsed = s.concat(d)

    self.bedroom_reservation_attributes = parsed

  end

  #######  controller validations
  def same_date
    return (self.check_in == self.check_out && (self.check_in) && (self.check_out))
  end

  def inverse_dates
    return(self.check_in > self.check_out)
  end

  def is_past
    return self.check_in < Date.today
  end

end
```

13.3.2.10 Role

```
class Role < ActiveRecord::Base
  has_and_belongs_to_many :users
  attr_accessible :name
end
```

13.3.2.11 User

```
class User < ActiveRecord::Base

  has_many :reservation

  has_and_belongs_to_many :roles

  before_save :setup_role

  before_destroy :check_for_reservations

  # Include default devise modules. Others available are:
  # :token_authenticatable, :confirmable,
  # :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
    :recoverable, :rememberable, :trackable, :validatable

  # Setup accessible (or protected) attributes for your model
  attr_accessible :dni, :role_id, :role_ids, :username, :email, :password,
:password_confirmation, :remember_me, :admin

  attr_accessible  :reservation_id, :confirm_email, :telephone, :name, :surname,
:address , :city, :post_code, :country
  # attr_accessible :title, :body

  def role?(role)
    return !!self.roles.find_by_name(role.to_s.camelize)
  end

  # Default role is "Member"
  def setup_role
    if self.role_ids.empty?
      self.role_ids = [2]
    end
  end
```

```
  def check_for_reservations
    if ((Reservation.all.select{ |r| r.user_id==self.id}.length) > 0)
      self.errors[:base] << "No se pueden eliminar usuarios que tengan reservas
asociadas."
      return false
    else
      return true
    end
  end

end
```

13.3.3 Schema (db/schema.rb)

```
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on another
# system, you should be using db:schema:load, not running all the migrations
# from scratch. The latter is a flawed and unsustainable approach (the more migrations
# you'll amass, the slower it'll run and the greater likelihood for issues).
#
# It's strongly recommended to check this file into your version control system.

ActiveRecord::Schema.define(:version => 20130622124006) do

  create_table "bedroom_reservations", :force => true do |t|
    t.integer  "reservation_id"
    t.integer  "bedroom_id"
    t.date     "assigned_at"
    t.datetime "created_at",                    :null => false
    t.datetime "updated_at",                    :null => false
    t.integer  "num_guest",      :default => 2
  end

  add_index "bedroom_reservations", ["bedroom_id"], :name =>
"index_bedroom_reservations_on_bedroom_id"
  add_index "bedroom_reservations", ["reservation_id"], :name =>
"index_bedroom_reservations_on_reservation_id"

  create_table "bedroom_types", :force => true do |t|
    t.string   "name",                  :default => ""
    t.integer  "price",                 :default => 60
    t.integer  "bedroom_id"
    t.datetime "created_at",                                :null => false
    t.datetime "updated_at",                                :null => false
    t.integer  "individual_difference", :default => 0
  end

  create_table "bedrooms", :force => true do |t|
    t.string   "name",                      :default => ""
    t.boolean  "supports_extra_bed",        :default => false
    t.boolean  "tv",                        :default => true
    t.boolean  "parking",                   :default => true
    t.boolean  "wifi",                      :default => true
    t.boolean  "bath",                      :default => true
    t.boolean  "breakfast",                 :default => true
    t.boolean  "handicapped",               :default => false
    t.datetime "created_at",                                    :null => false
    t.datetime "updated_at",                                    :null => false
    t.integer  "bedroom_type_id"
    t.string   "main_photo_file_name"
    t.string   "main_photo_content_type"
    t.integer  "main_photo_file_size"
    t.datetime "main_photo_updated_at"
  end

  create_table "intervals", :force => true do |t|
    t.string   "name",            :default => ""
    t.integer  "price"
    t.integer  "bedroom_type_id"
    t.datetime "created_at",                      :null => false
    t.datetime "updated_at",                      :null => false
    t.date     "start"
    t.date     "end"
  end

  create_table "partners", :force => true do |t|
    t.string   "name"
    t.string   "website"
    t.integer  "telephone"
    t.string   "logo_file_name"
    t.string   "logo_content_type"
    t.integer  "logo_file_size"
```

```
    t.datetime "logo_updated_at"
    t.datetime "created_at",         :null => false
    t.datetime "updated_at",         :null => false
  end

  create_table "photos", :force => true do |t|
    t.string   "name"
    t.integer  "bedroom_id"
    t.string   "image_file_name"
    t.string   "image_content_type"
    t.integer  "image_file_size"
    t.datetime "image_updated_at"
    t.datetime "created_at",          :null => false
    t.datetime "updated_at",          :null => false
  end

  create_table "reservations", :force => true do |t|
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
    t.integer  "user_id"
    t.integer  "partner_id"
    t.date     "check_in"
    t.date     "check_out"
  end

  create_table "roles", :force => true do |t|
    t.string   "name"
    t.datetime "created_at", :null => false
    t.datetime "updated_at", :null => false
  end

  create_table "roles_users", :id => false, :force => true do |t|
    t.integer "role_id"
    t.integer "user_id"
  end

  create_table "users", :force => true do |t|
    t.string   "email",                  :default => "", :null => false
    t.string   "encrypted_password",     :default => "", :null => false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          :default => 0
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
    t.datetime "created_at",                           :null => false
    t.datetime "updated_at",                           :null => false
    t.string   "confirm_email"
    t.string   "name"
    t.string   "telephone"
    t.string   "surname"
    t.string   "address"
    t.string   "city"
    t.integer  "post_code"
    t.string   "country"
    t.string   "username"
    t.string   "dni"
  end

  add_index "users", ["email"], :name => "index_users_on_email", :unique => true
  add_index "users", ["reset_password_token"], :name =>
"index_users_on_reset_password_token", :unique => true

end
```

13.3.4 Application.rb (config/application.rb)

```
require File.expand_path('../boot', __FILE__)

require 'rails/all'
require 'configatron'



if defined?(Bundler)
  # If you precompile assets before deploying to production, use this line
  Bundler.require(*Rails.groups(:assets => %w(development test)))
  # If you want your assets lazily compiled in production, use this line
  # Bundler.require(:default, :assets, Rails.env)
end

module DeviseSobrefuentes
  class Application < Rails::Application
    # Settings in config/environments/* take precedence over those specified here.
    # Application configuration should go into files in config/initializers
    # -- all .rb files in that directory are automatically loaded.

    # Custom directories with classes and modules you want to be autoloadable.
    # config.autoload_paths += %W(#{config.root}/extras)

    # Only load the plugins named here, in the order given (default is alphabetical).
    # :all can be used as a placeholder for all plugins not explicitly named.
    # config.plugins = [ :exception_notification, :ssl_requirement, :all ]

    # Activate observers that should always be running.
    # config.active_record.observers = :cacher, :garbage_collector, :forum_observer

    # Set Time.zone default to the specified zone and make Active Record auto-convert to
this zone.
    # Run "rake -D time" for a list of tasks for finding time zone names. Default is
UTC.
    # config.time_zone = 'Central Time (US & Canada)'

    # The default locale is :en and all translations from config/locales/*.rb,yml are
auto loaded.
    # config.i18n.load_path += Dir[Rails.root.join('my', 'locales', '*.{rb,yml}').to_s]
    # config.i18n.default_locale = :de

    # Configure the default encoding used in templates for Ruby 1.9.
    config.encoding = "utf-8"

    config.assets.paths << "#{Rails.root}/app/assets/fonts"
    config.assets.paths << "#{Rails.root}/app/assets/images-frontend"
    config.assets.paths << "#{Rails.root}/app/assets/images"
    config.assets.paths << "#{Rails.root}/app/assets/theme"

    # Configure sensitive parameters which will be filtered from the log file.
    config.filter_parameters += [:password]

    # Enable escaping HTML in JSON.
    config.active_support.escape_html_entities_in_json = true

    # Use SQL instead of Active Record's schema dumper when creating the database.
    # This is necessary if your schema can't be completely dumped by the schema dumper,
    # like if you have constraints or database-specific column types
    # config.active_record.schema_format = :sql

    # Enforce whitelist mode for mass assignment.
    # This will create an empty whitelist of attributes available for mass-assignment
for all models
    # in your app. As such, your models will need to explicitly whitelist or blacklist
accessible
    # parameters by using an attr_accessible or attr_protected declaration.
    config.active_record.whitelist_attributes = true

    # Enable the asset pipeline
    config.assets.enabled = true

    # Version of your assets, change this if you want to expire all your assets
    config.assets.version = '1.0'
```

```
    config.i18n.default_locale = :es

    config.time_zone = 'UTC'


    config.action_mailer.raise_delivery_errors = true
    config.action_mailer.perform_deliveries = true
    config.action_mailer.delivery_method = :smtp
    config.action_mailer.smtp_settings = {
      :enable_starttls_auto => true,
      :address        => 'smtp.gmail.com',
      :port           => 587,
      :domain         => 'gmail.com',
      :authentication => :plain,
      :user_name      => 'luis.amor.alvarez@gmail.com',
      :password       => xxxx
    }



    config.action_view.field_error_proc = Proc.new { |html_tag, instance|
      "#{html_tag}".html_safe
    }


     configatron.email = 'luis.amor.alvarez@gmail.com'
     configatron.telephone = '+34 985 942 832'
  end
end
```