Trabajo Fin de Carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Facultat de Matemàtiques
Universitat de Barcelona

APLICACIÓN WEB PARA COMPARTIR COCHE, MEDIANTE LAS TECNOLOGÍAS JAVA, SPRING Y JPA

Francisco Pérez Rodríguez

Director: Lluis Garrido Osterman
Realizado en: Departament de Matemàtica Aplicada i Anàlisi. UB
Barcelona, 20 de Junio de 2013

Contenido

Introducción

Como vemos cada día la sociedad evoluciona hacia un mundo más conectado, más en red, donde la presencia de las redes sociales, cada vez tienen más peso, hoy en día toda aplicación tiene su parte social, con esa idea en mente me propuse crear una herramienta más social para el problema de la sobre saturación de vehículos individuales, creando una herramienta donde poder compartir tu vehículo, con el resto de usuarios de la misma.

Este proyecto de fin de carrera tiene como objetivo crear una aplicación Web que creará el entorno donde los usuarios podrán encontrar a otras personas dispuestas a compartir su coche ya sea porque hacen el mismo recorrido o solo una parte de él, con los beneficios que eso reporta a ambos, gastos compartidos, menos contaminación, un trayecto más ameno, entre muchos otros. Pero no solo esto, ya que los usuarios podrán hablar de su experiencia en el viaje, así como enviar mensajes directamente al resto de usuarios.

Uno de los puntos diferenciales de mi proyecto con aplicaciones del mismo estilo es la posibilidad de añadir puntos intermedios en el trayecto, de esa forma un viaje creado en mi aplicación no es solo un origen y destino si no que también es posible que tenga diferentes puntos intermedios. Además de integrarse con Google Maps creando la ruta de tu viaje en uno de los mapas para mejor visualización del trayecto.

La aplicación ofrece las siguientes funcionalidades:

1. Encontrar los viajes que te interesen.
2. Crear tus propios viajes compartidos.
3. Poder tener una conversación con el propietario del vehículo que hace el trayecto que tu quieres compartir.
4. Editar tu perfil personal

La aplicación hará uso de un patrón de desarrollo llamado MVC (modelo-vista-controlador) implementado en JAVA, y usando SPRING en la capa de controlador, páginas JSPs para crear las vistas de la aplicación y entidades JPA como nuestro modelo, todo ello sobre un servidor de aplicaciones Glassfish.

En el capítulo de **Tecnologías utilizadas** haré una comparación con el resto de tecnologías para desarrollar aplicaciones web. Y una breve explicación de las diferentes tecnologías que necesitaremos conocer.

En el apartado **Diseño** haré una explicación de los Diagramas de clases, los casos de uso, los diagramas de secuencia y la estructura de base de datos.
Y para terminar mostraré los resultados conseguidos con la aplicación y mi conclusión final.

Tecnologías utilizadas

Para la realización de este proyecto investigué sobre qué tecnologías son las más utilizadas en las aplicaciones web y encontré que hoy en día se utilizan muchos tipos, pero sobre las que más documentación fueron PHP, ASP.NET y JAVA. Además aparte del lenguaje de programación existen muy diversos frameworks para el desarrollo de aplicaciones web, sobre los que también estuve investigando y explicaré a continuación.

Revisando documentación sobre PHP y documentándome encontré que a día de hoy es una de las tecnologías más empleadas y además su código fuente es libre y esto siempre hace que exista una buena comunidad de desarrolladores entorno a esta tecnología, además ofrece ciertas facilidades al desarrollador, por contra PHP es un lenguaje de script no compilado y eso siempre dificulta el desarrollo, ya que se pueden escapar errores, debido a la ausencia del compilador. Además no me gustó que la orientación a objetos fuera deficiente en grandes aplicaciones y que el código fuera menos legible al unir código HTML con el propio de PHP.
Por otra parte ASP .NET vi que tiene una separación completa entre lo que el usuario ve y el código, y tener la lógica de negocio separada es lo que se debe hacer. Por último me documenté sobre JAVA y encontré que mediante la utilización del Framework SPRING MVC tenía las virtudes de ASP y obviamente la orientación a objetos es tan potente como el propio JAVA.

Después de toda esta búsqueda me decidí a utilizar JAVA como mi lenguaje de programación, pero añadiendo al proyecto web el framework SPRING MVC, que consiste en un software que permite una división limpia entre el modelo, la vista y el controlador. Además se utiliza JPA que es el framework de JAVA para conectar la aplicación con el servidor de base de datos, que en mi caso es MySQL. JPA se utiliza configurando un fichero XML llamado persistence.xml y colocando una serie de anotaciones en las clases JAVA como puede ser @Entity, @Column entre muchas otras, que en el apartado de Estructura de base de datos explicaré más en detalle.

JAVA

Según Wikipedia: *El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en el 1995 como un componente fundamental de la plataforma Java de Sun Microsystems.*

Su sintaxis deriva mucho de C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java pueden ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir del 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados

El lenguaje Java se creó con cinco objetivos principales:

1. *Debería usar el paradigma de la programación orientada a objetos.*
2. *Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.*
3. *Debería incluir por defecto soporte para trabajo en red.*
4. *Debería diseñarse para ejecutar código en sistemas remotos de forma segura.*
5. *Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.*

JSP

Según Wikipedia: *JavaServer Pages (JSP) es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML entre otros tipos de documentos.*
Las JSPs son en realidad una forma alternativa de crear servlets ya que el código JSP se traduce a código de servlet Java la primera vez que se le invoca y en adelante es el código del nuevo servlet el que se ejecuta produciendo como salida el código HTML que compone la página web de respuesta.

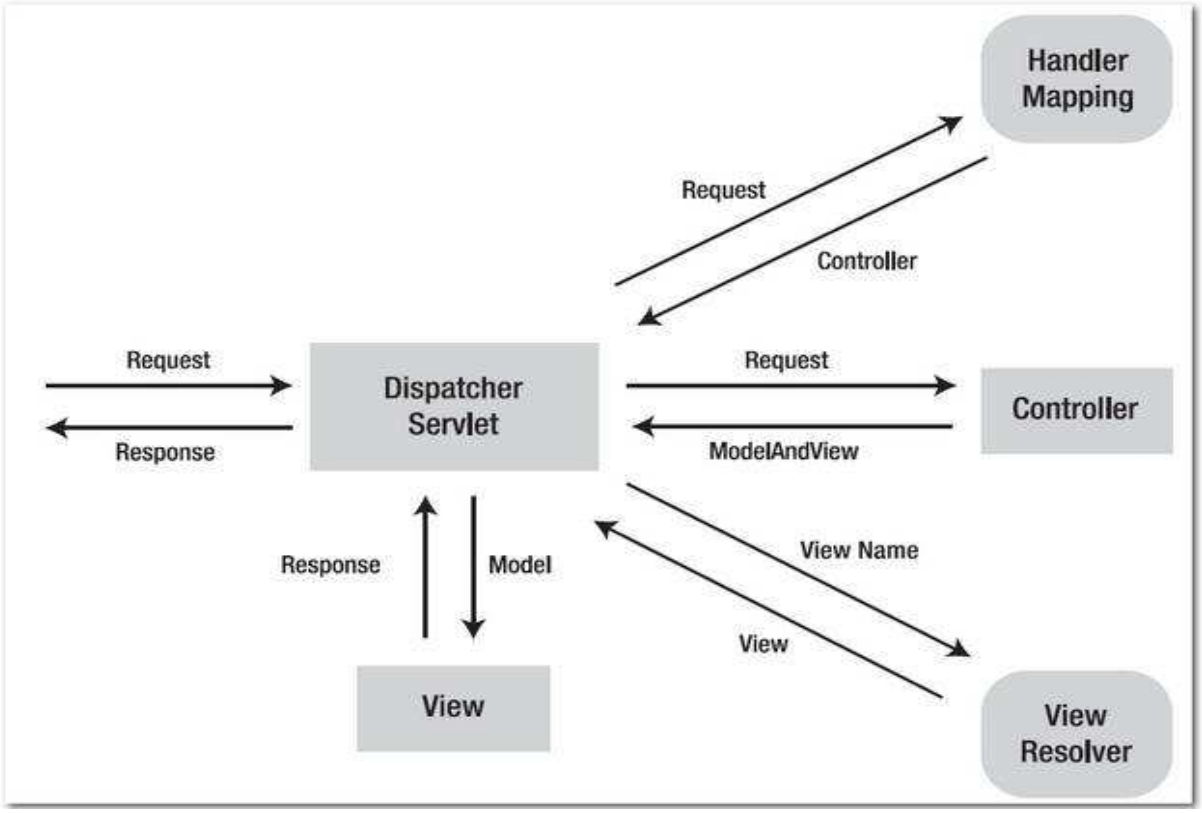
En la aplicación existirán páginas JSP que se encargan de procesar los datos que le envía el servidor y crean un HTML que después se enviará al cliente, estás páginas JSP contienen dentro código HTML, CSS, Javascript y tags JSTL.

SPRING

Según Wikipedia: *Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java.*

Spring Framework comprende diversos módulos que proveen un rango de servicios muy amplio pero el que nos interesa para este proyecto es el MVC: que nos dota de un framework basado en HTTP y servlets, que provee herramientas para la extensión y personalización de aplicaciones web y servicios web REST.

Explicaré que significa el patrón MVC y como está implementado en este proyecto. El patrón MVC busca la división completa entre la lógica de datos, el modelo de negocio y las vistas de la aplicación. Para conseguir eso implementé el framework Spring MVC que funciona de la siguiente manera:



Spring tiene como eje central su Servlet Dispatcher que será el encargado de centralizar todas las peticiones y las reenviará al controlador que sea el responsable de responder a la URL de esa petición. Toda petición pasa por el HadlerMapping que recoge la petición y encuentra que controlador es el responsable de atender esa petición. El controlador responderá la petición y creará todos los objetos que necesite la vista donde deba terminar esa petición y se envía al cliente hacia la vista que nos está pidiendo.

Supongamos que tenemos el siguiente caso: Un usuario quiere dirigirse a la url www.proyecto.com/miProyecto/login, para que el cliente termine viendo la pantalla de login a la aplicación pasará todo lo siguiente.

Esa petición llegará al Servlet Dispatcher él comprobará si existe esta URL en su contexto, y decidirá que debe atender esta petición el controlador Login, debido que es este controlador quien tiene implementada la anotación @PeticiónMapping (value = "/login") al inicio de la clase. Ahora se comprobará si la petición es GET o POST y en función de esto se enviará al método del controlador Login que debe atender esta petición.

Cuando se ejecuta este método, creará el objeto de formulario, que sabe que necesita la vista (JSP) para crear la página de entrada a la aplicación, una vez termina de crear este objeto le dice al Dispatcher que debe encaminar al usuario hacia la vista login.jsp y que además esa vista va a necesitar el objeto que acaba de crear.

Una vez el Dispatcher recibe esa información se la pasa a la vista adecuada, que comenzará inmediatamente a crear el HTML final que nos había solicitado el cliente.

En la aplicación se requerirán controladores SPRING, para dar visibilidad a las diferentes URL de la aplicación, para la gestión de las peticiones del cliente y la respuesta a las llamadas asíncronas de la página.
Por llamada asíncrona se entiende **Ajax**, esta tecnología se utiliza para que las páginas no necesiten recargar por completo toda la página con el gasto que eso conlleva para el servidor (crear de nuevo todos los objetos necesarios para la jsp, crear otra vez el html final, etc.) sino que mediante una petición asíncrona al servidor se le solicitan los nuevos datos y el servidor de los devuelve mediante un **JSON**. Una vez la página ha recibido esta nueva información usando javascript se recarga el contenido con los nuevos datos.

JPA

Según Wikipedia: *Java Persistence API, más conocida por sus siglas JPA, es la API de persistencia desarrollada para la plataforma Java EE.*

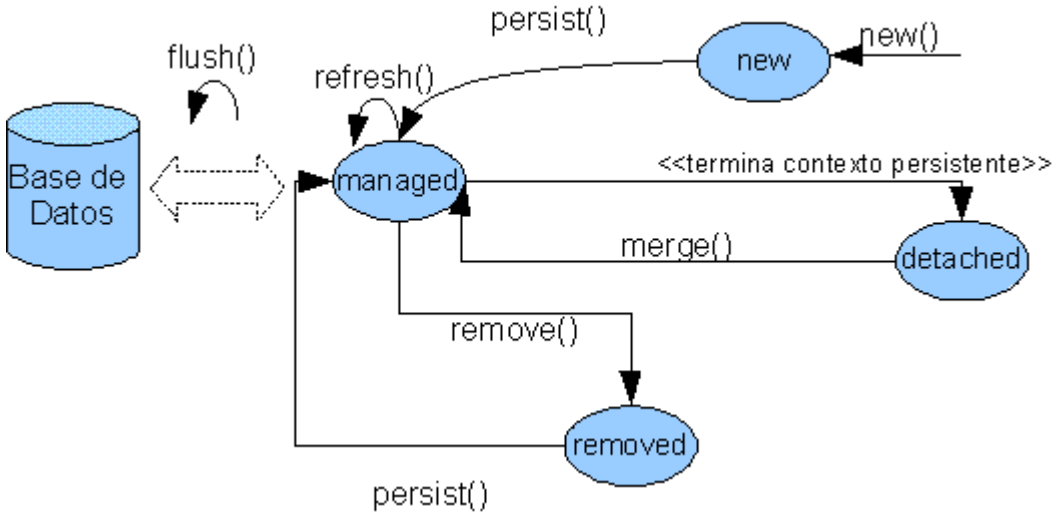
Es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE).

Persistencia en este contexto cubre tres áreas:

1. *La API en sí misma, definida en el paquete javax.persistence*
2. *El lenguaje de consulta Java Persistence Query Language (JPQL)*
3. *Metadatos objeto/relacional*

El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional), como sí pasaba con EJB2, y permitir usar objetos regulares (conocidos como POJOs).

Explicaré con un gráfico cómo funciona el ciclo de vida de una entidad JPA:



Para entender cómo funciona el entorno de persistencia, se ha de entender el objeto que lo gestiona, llamado EntityManager.

Un EntityManager es una instancia de **EntityManagerFactory** en **JPA**. Dicha factoría lo que hace es representar la configuración para acceder a la base de datos que utilice nuestra aplicación. Él será quien nos permita crear, editar, destruir y buscar sobre nuestra base de datos.
El EntityManager funciona mediante transacciones, se ha de hacer siempre una transacción para enviar los cambios a la base de datos.
Las entidades pueden estar en cualquiera de los 4 estados que se representan en el diagrama, al crear un objeto marcado como @Entity pasa al estado new, mientras que una vez que finaliza la transacción el objeto queda en estado managed, y si una vez a finalizada la transacción se vuelve a cambiar el objeto se marca como detached, y cuando se borra va a parar al estado removed y el garbage collector lo acabará borrando definitivamente.

En la aplicación tendremos varias clases POJO que son la representación de una de nuestras tablas del servidor MySQL. Para la creación de estas clases es necesario aplicarles anotaciones especiales de JPA.

Describiré las anotaciones más básicas:

@Cacheable: Especifica que la entidad necesita ser cacheada
@Entity: Define que la clase es una entidad JPA, se coloca al principio de la clase.
@Id: Define que ese atributo es la clave primaria de la tabla
@ManyToMany: Define una relación N-N
@ManyToOne: Define la relación N-1
@OneToMany: Define la relación 1-N
@OneToOne: Define la relación 1-1

MYSQL

Según Wikipedia: *MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.*

En el caso de la aplicación será necesario configurar el fichero persistence.xml que utilizará un jdbc para MySQL y se conectará al servidor que estará alojado en localhost:3306 y a la base de datos engine. Además también establecemos las conexiones máximas y mínimas hacia la base de datos.
En el anexo se podrá ver que formato tiene.

GLASSFISH

Según Wikipedia: *GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito, de código libre y se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL. La versión comercial es denominada Oracle GlassFish Enterprise Server.*

El servidor es quien se encargará de desplegar todos los componentes (mediante la compilación de un fichero WAR). Y será él quien se encargue de mantener online todos nuestros controladores, y los recursos necesarios para la aplicación (imágenes, ficheros CSS, ficheros Javascript, etc.).

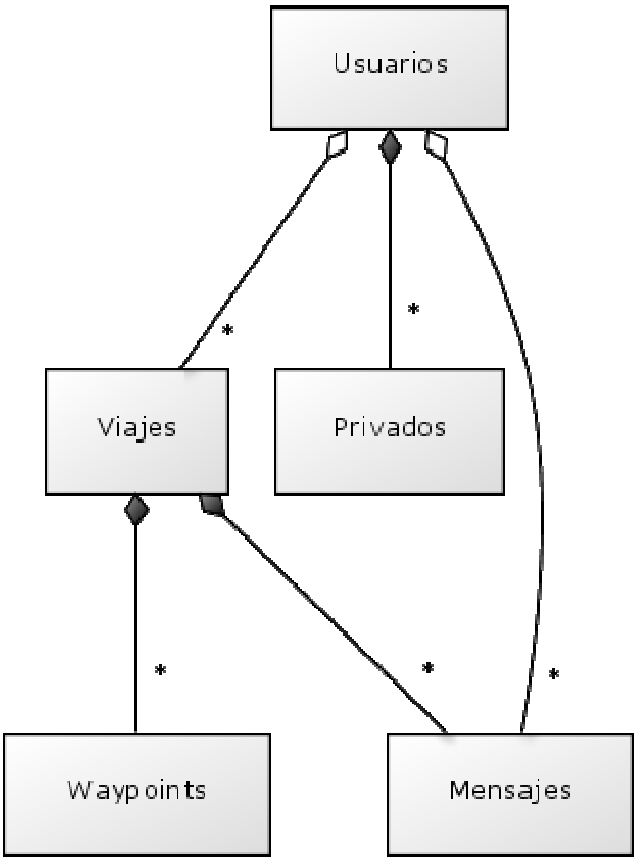
Patrón de presentación view helper

También utilizo el patrón de presentación view helper eso quiere decir que para la creación de las vistas existe una clase que mediante diferentes métodos ayuda a la JSPs a crear el HTML final que se envía al cliente. Estos ayudantes se insertan en la JSPs mediante un taglib, es decir que cree mi propia librería de tags para que se beneficien todas mis JSPs de los métodos que desarrollé en esa clase.

Diseño

Dentro de esta sección se explica el diagrama de clases de la aplicación los casos de uso, los diagramas de secuencia y la estructura de la base de datos.

Diagrama de clases



Usuarios: esta clase es la encargada de gestionar todos los datos del usuario, nombre, apellidos, login, password. También contiene una lista de objetos Privados, para la gestión de los mensajes privados del usuario. Además tiene el campo securityCode, al crear un nuevo usuario el sistema coge la hora actual en formato timestamp (mediante System.currentmillis) y se crea una Hash mediante el algoritmo MD5 que después se guarda en este campo. Este atributo es el que se verifica cuando el usuario quiere activar su usuario mediante el link del correo que recibirá, una vez comprobado se pondrá a verdadero el atributo active que también tiene la clase usuario.

Privados: esta clase representa a un mensaje desde un usuario a otro usuario. Contiene como atributos dos objetos Usuario, uno es el usuario que envía el mensaje y el otro el destinatario. Además de un campo Date para guardar cuando fue enviado el mensaje.

Mensajes: esta clase es la representación de los mensajes que los diferentes usuarios irán dejando en una conversación dentro de un viaje. Tiene un atributo de tipo Viaje que nos sirve para identificar a que viaje pertenece el mensaje.

Viajes: esta clase contiene la información necesaria para el control de un viaje (origen, destino, etc.), y además tiene una lista de todos los mensajes necesarios para controlar la conversación de ese viaje. Además también contiene una lista de puntos intermedios.

Waypoints: esta clase representa a un punto intermedio en un viaje y contiene la dirección de este punto intermedio, además de un objeto de tipo viaje que sirve para conocer a que viaje pertenece este punto intermedio.

Casos de uso

En esta sección describiré los casos de uso más importantes en la aplicación.

Login

Definición: El usuario quiere acceder a la aplicación.

Actor: Usuario

Requerimientos implementados:

- Se comprueba que los datos introducidos mediante el formulario son correctos (usuario y password) concediendo el acceso a los usuarios debidamente registrados o impidiendo la entrada a los non gratos.

Registrarse

Definición: El usuario quiere conseguir acceso a la aplicación.

Actor: Usuario

Requerimientos implementados:

- Se presenta el formulario de registro
- Una vez completado el envío de la información desde el formulario, el servidor crea una nueva entrada en la tabla de usuarios y la marca como pendiente de activación
- El servidor crea un email y lo envía al destinatario con una URL de activación de su cuenta

Activación de la cuenta

Definición: El usuario quiere activar la cuenta de la aplicación.

Actor: Usuario

Requerimientos implementados:

- El usuario ha entrado la URL que lo valida cómo el usuario que completó el formulario de registro y se marca la cuenta asociada como válida.

Ver tus viajes

Definición: El usuario quiere navegar a través de sus viajes.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal tiene una lista paginada donde están todos sus propios viajes.

Borrar uno de tus viajes

Definición: El usuario quiere eliminar uno de sus viajes.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón borrar, en alguno de sus viajes
- El sistema encuentra el viaje seleccionado y lo borra
- El sistema devuelve al usuario a su página principal

Ver comentarios de uno de tus viajes

Definición: El usuario quiere ver los comentarios que han dejado sobre uno de sus viajes.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón comentarios en uno de sus viajes
- El sistema encuentra todos los comentarios para ese viaje en concreto
- El sistema presenta una nueva ventana donde se pueden observar todos los comentarios.

Comentar en uno de tus viajes

Definición: El usuario quiere comentar en uno de sus viajes como respuesta al resto de comentarios.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón comentarios en uno de sus viajes
- El sistema presenta una nueva ventana donde se pueden observar todos los comentarios
- Allí dentro el usuario hace clic en el botón contestar
- El sistema le presenta una ventana donde escribir su replica
- A partir de ese momento ese nuevo comentario forma parte de la conversación del viaje donde conteste

Buscar un viaje

Definición: El usuario quiere buscar un viaje según un origen y un destino.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón buscar
- El sistema le presenta una caja donde el usuario escribirá el origen y el destino, de los viajes que quiere encontrar.
- El usuario escribe los datos de búsqueda
- El sistema le presenta los viajes encontrados

Crear un viaje

Definición: El usuario quiere crear un viaje.

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón crear viaje
- El sistema presenta un formulario
- El usuario completa el formulario
- El sistema le informa mediante un mensaje que el viaje se creó correctamente, y lo envía a su página principal

Mi canal

Definición: El usuario quiere ver su canal

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón mi canal
- El sistema presenta una ventana donde se pueden ver todas las conversaciones donde el usuario a escrito

Contestar en una de tus conversaciones

Definición: El usuario comentar en alguna de sus conversaciones

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón mi canal
- El sistema presenta una ventana donde se pueden ver todas las conversaciones donde el usuario a escrito
- El usuario hace clic en el botón contestar, al lado de la conversación donde desee comentar.
- El sistema le presenta una ventana donde puede escribir su comentario
- A partir de ese momento ese comentario pasa a formar parte de los comentarios de esa conversación

Ver mensajes privados

Definición: El usuario quiere ver sus mensajes privados

Actor: Usuario

Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón privados
- El sistema presenta una ventana donde se pueden ver todos sus mensajes privados

Contestar un mensaje privado

Definición: El usuario quiere contestar uno de los mensajes privados que tenga

Actor: Usuario

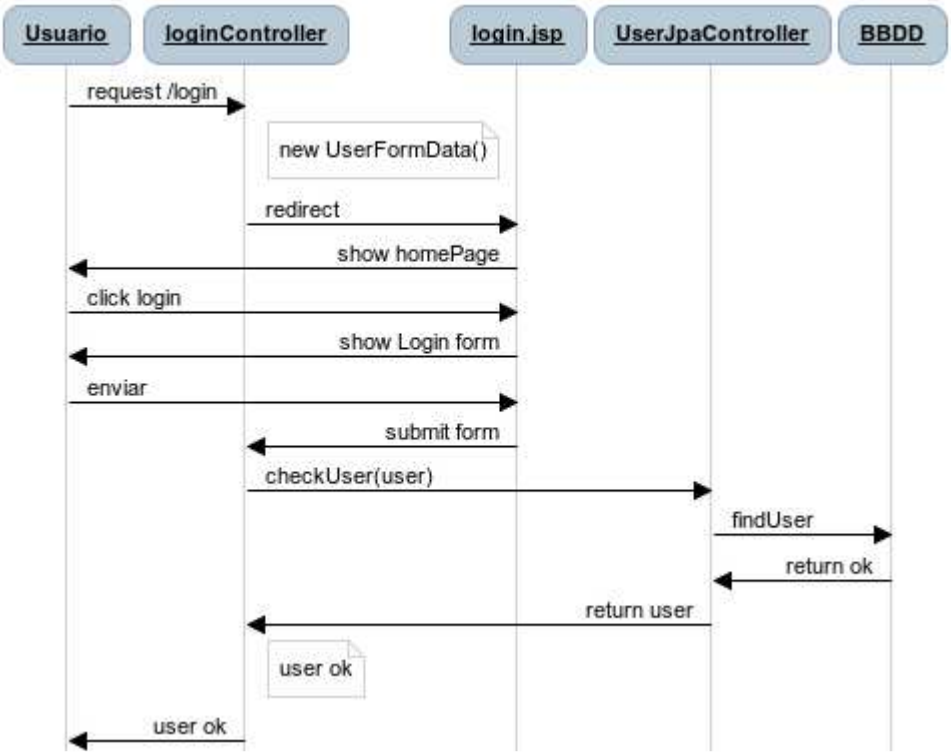
Requerimientos implementados:

- El usuario desde la página principal hace clic en el botón privados
- El sistema presenta una ventana donde se pueden ver todos sus mensajes privados
- El usuario hace clic en el botón contestar
- El sistema presenta una ventana donde puede introducir el mensaje privado.

Diagramas de secuencia

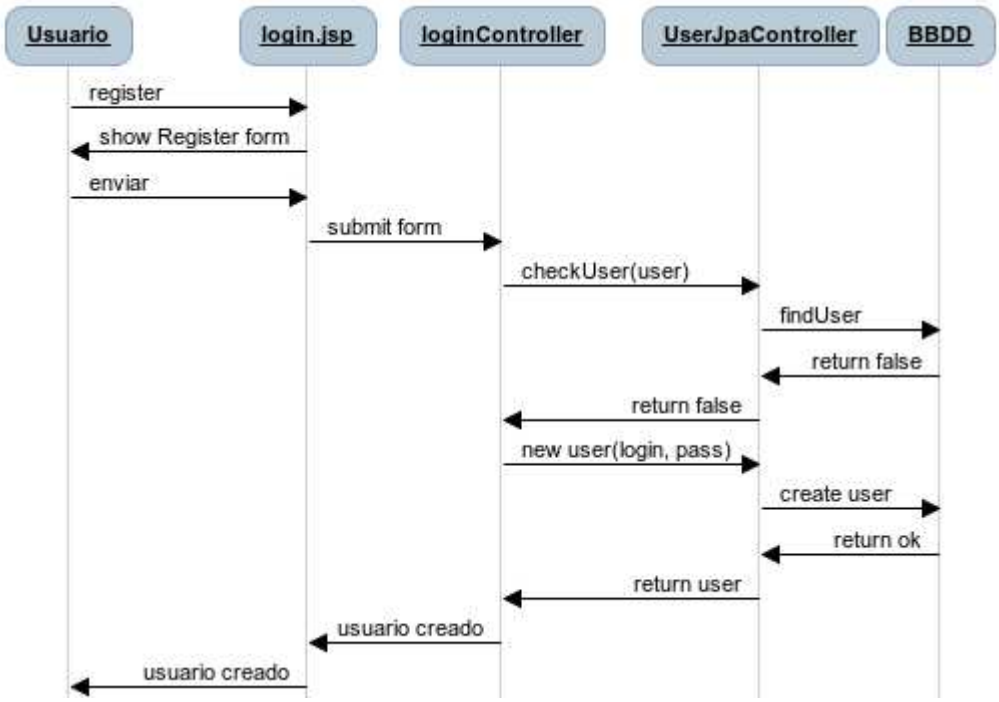
En esta sección se comentan los diagramas de secuencia más importantes de la aplicación.

Login



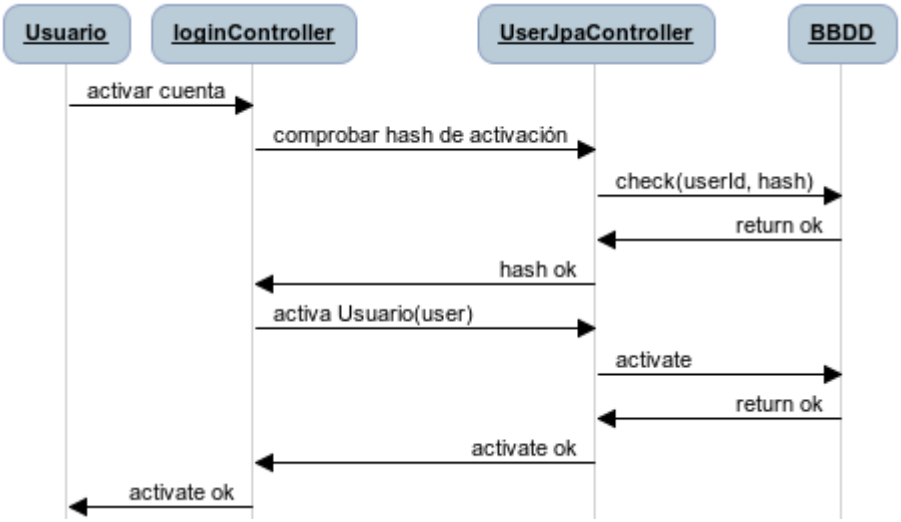
El usuario quiere acceder a la aplicación, y pide ir a la URL /login, el controlador loginController recogerá su petición creará el objeto de formulario y re direccionará al usuario a la vista login.jsp. Una vez aquí el usuario completará la información del formulario y le dará a enviar, haciendo clic en login. La vista enviará toda la información del formulario al controlador y el verificará en la base de datos que si ese usuario coincide con alguno existente o no, finalizando el proceso correctamente o invalidando la entrada al usuario si no coinciden los datos.

Registrarse



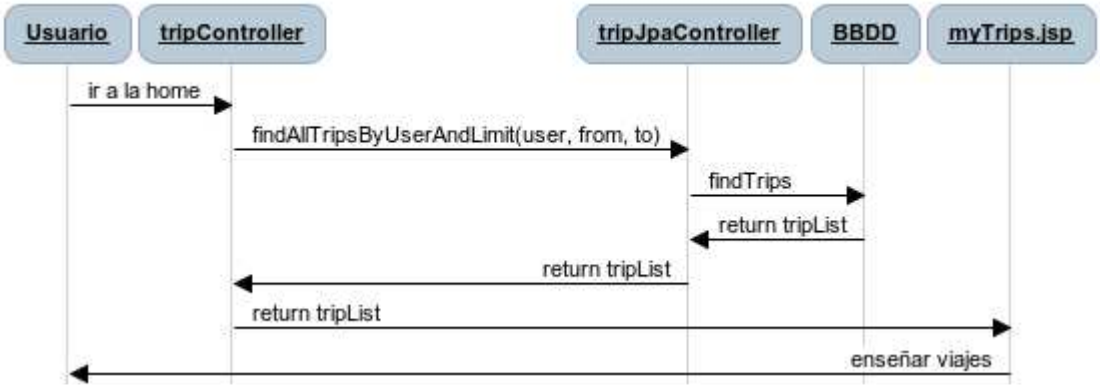
El usuario quiere registrarse en la aplicación, y para ello debe completar el formulario de registro que la vista login.jsp le habrá mostrado, una vez llegue la información del formulario al controlador loginController, comprobará que ese usuario no esté ya dado de alta en el sistema y si la consulta a la base de datos es negativa procederá a la creación del usuario que se está registrando y lo marcará como pendiente de activación y enviará un email al usuario con una URL donde puede activarlo.

Activación de la cuenta



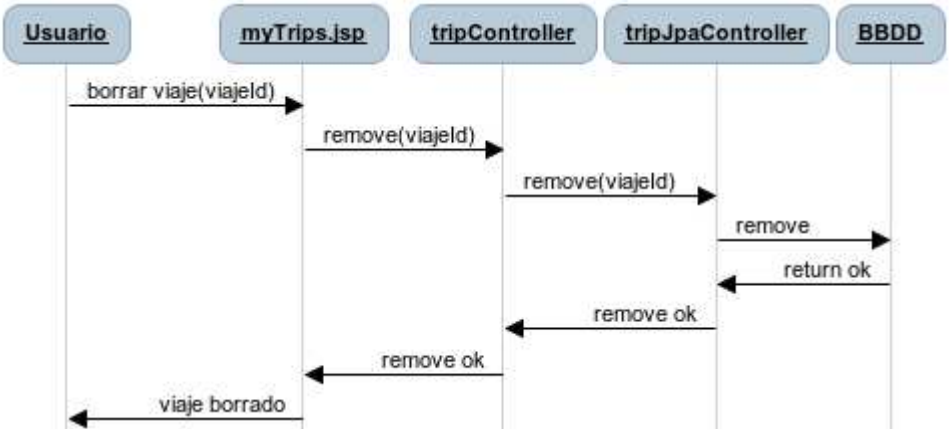
El usuario habrá recibido un email del sistema indicando que debe activar su usuario y él habrá hecho clic en el link, una vez hace eso es el loginController quien recibe la petición y comprobará que el código de activación es el correcto y que el identificador del usuario también lo es. Una vez se comprueba que esto es cierto, se procede a activar el usuario en la aplicación.

Ver tus viajes



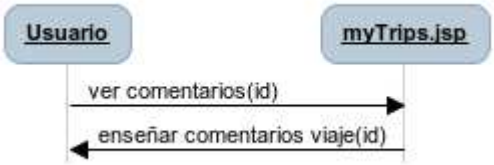
El usuario entra en su apartado home de la aplicación, el sistema cogerá los primeros viajes del usuario mediante una consulta a la base de datos y creará la lista que enviará a la vista myTrips.jsp que será la que finalmente acabe viendo el usuario.

Borrar uno de tus viajes



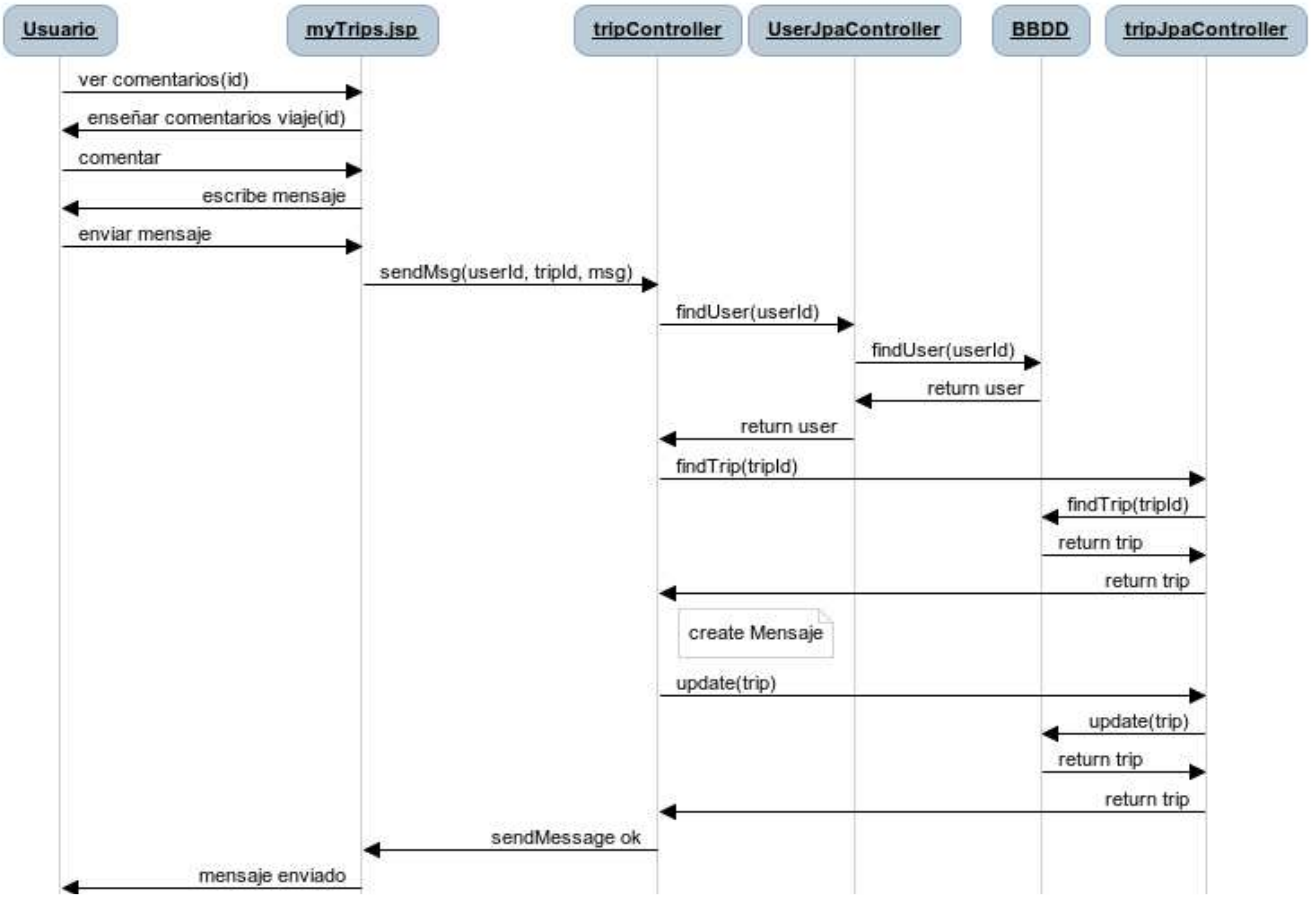
El usuario desde su página de home hace clic en el botón de borrar uno de sus viajes, en ese momento se envía mediante Ajax al controlador tripController la petición del cliente, y el controlador recibirá como parámetro el identificador del viaje a borrar, y procederá a borrarlo de la base de datos.

Ver comentarios de uno de tus viajes



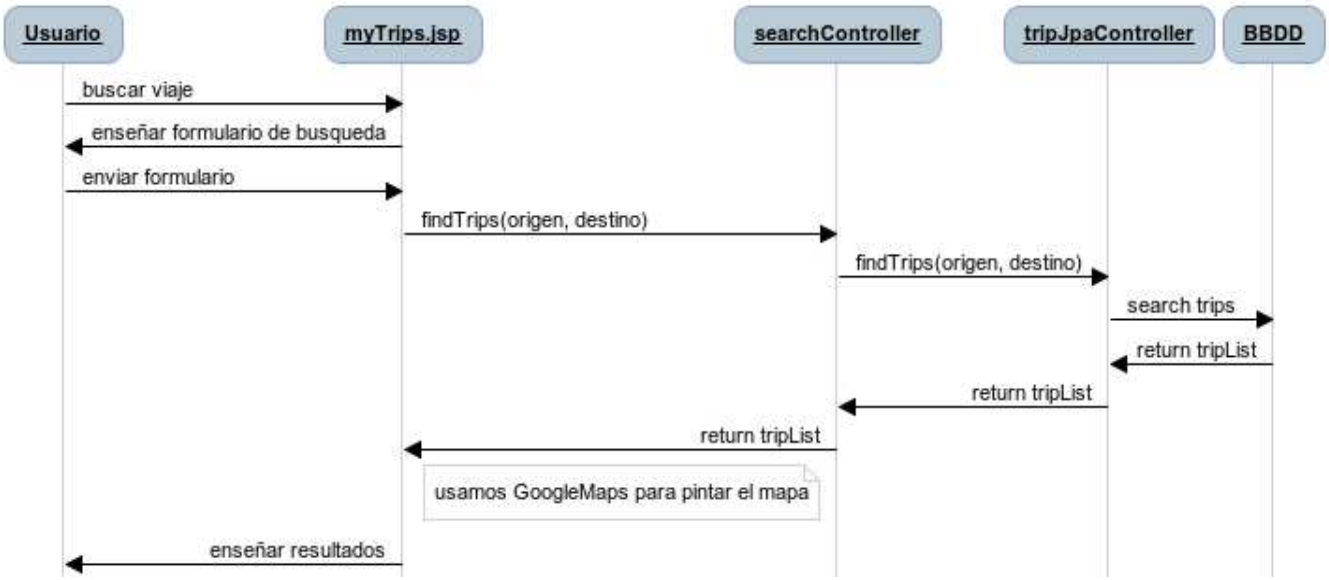
El usuario quiere ver los comentarios de uno de sus viajes, para ello la vista myTrips.jsp, previamente tenia cargados los mensajes de ese viaje, y cuando el usuario desea verlos, nada más tiene que enseñárselos.

Comentar en uno de tus viajes



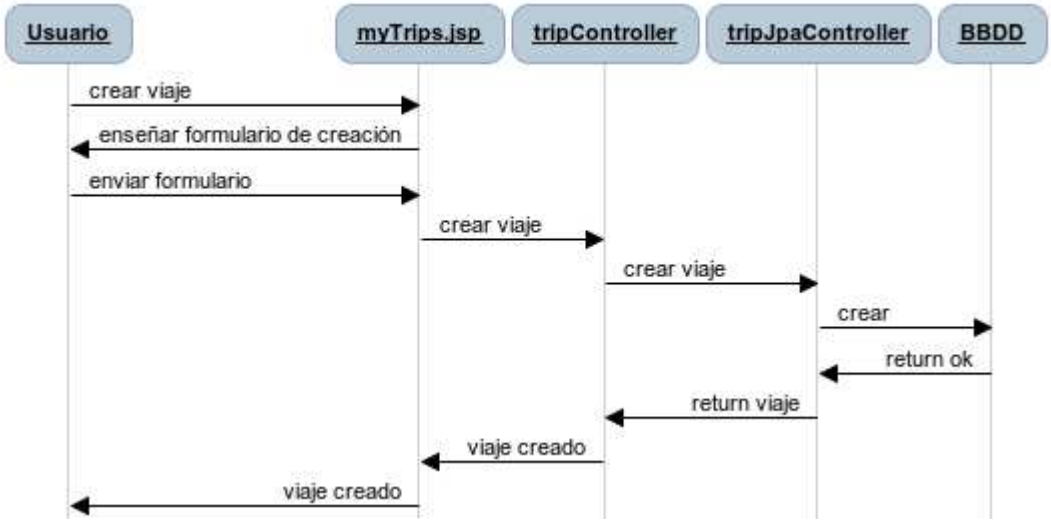
El usuario hace clic en el botón de comentar en uno de sus viajes, el sistema presenta la ventana donde el usuario puede escribir su mensaje y lo envía al tripController, que buscará si es correcto el usuario y el viaje donde se va a escribir el mensaje, y entonces crea el mensaje y actualiza el viaje con la última información.

Buscar un viaje



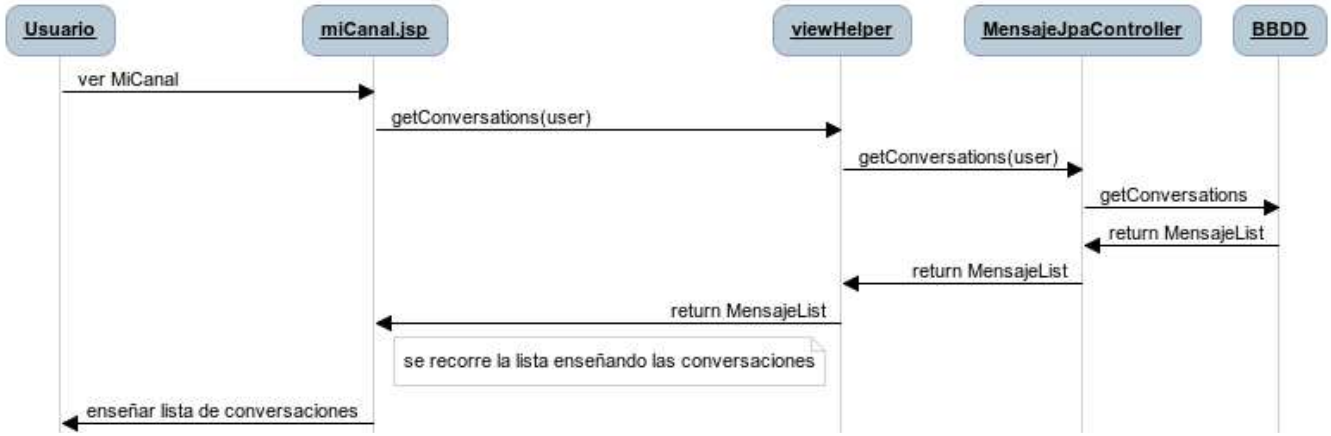
El usuario busca un viaje mediante el formulario que el sistema le presenta, cuando el cliente envía el formulario, el sistema busca en la base de datos un viaje que coincida con ese origen y destino, una vez el sistema devuelve la información a myTrips.jsp se hace una petición a GoogleMaps para pintar el mapa de todos los viajes resultado de la búsqueda.

Crear un viaje



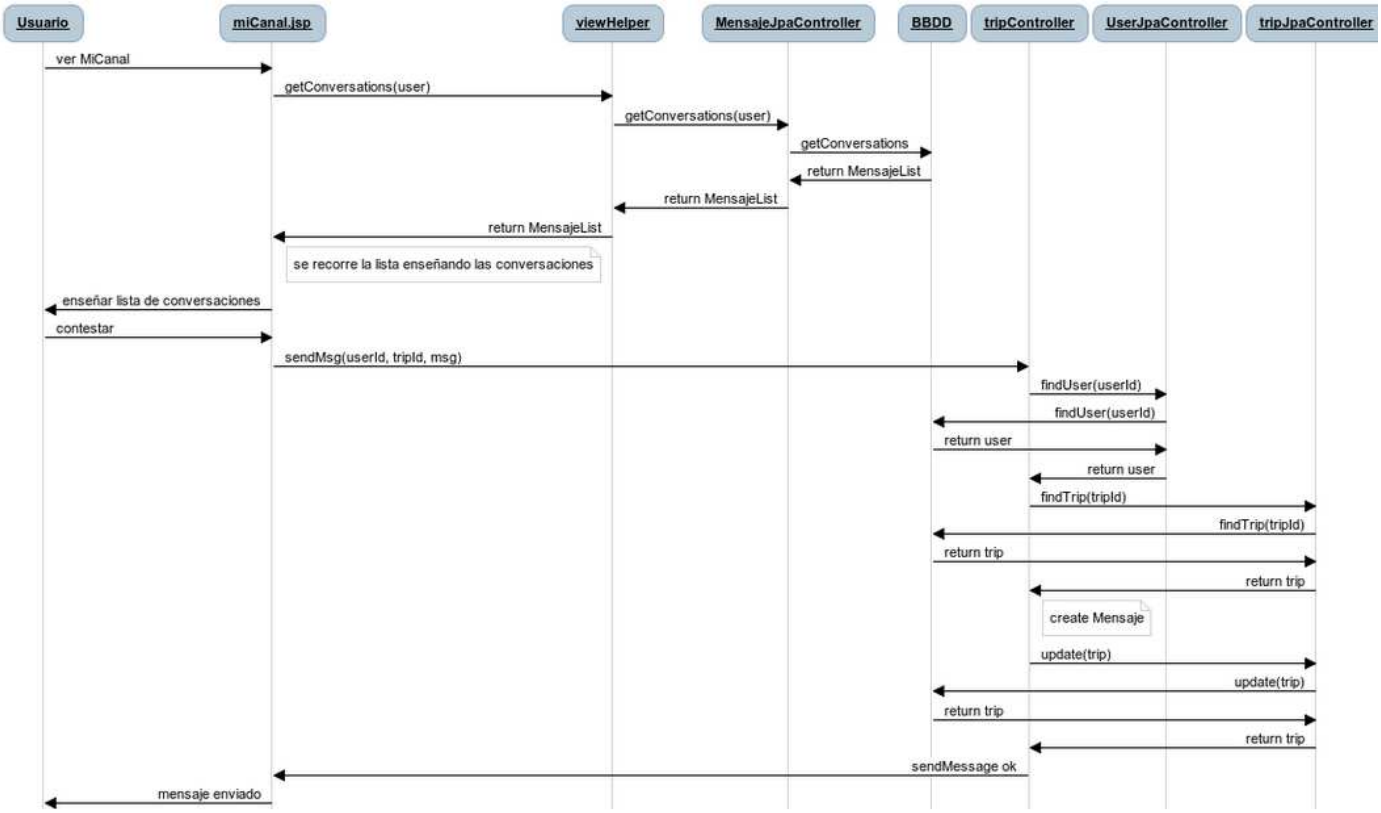
El usuario a través de formulario completa todos los datos que son necesarios para un viaje y lo envía al servidor, en este caso al controlador tripController, que creará el viaje en la base de datos.

Mi canal



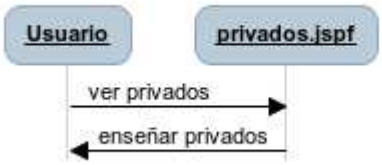
El usuario quiere ver si canal personal, que son todas las conversaciones donde él ha hablado, el ayudante de la vista coge las conversaciones del usuario en cuestión haciendo una consulta a la base de datos, y una vez termina le devuelve a la vista la lista de conversaciones que necesita.

Contestar en una de tus conversaciones



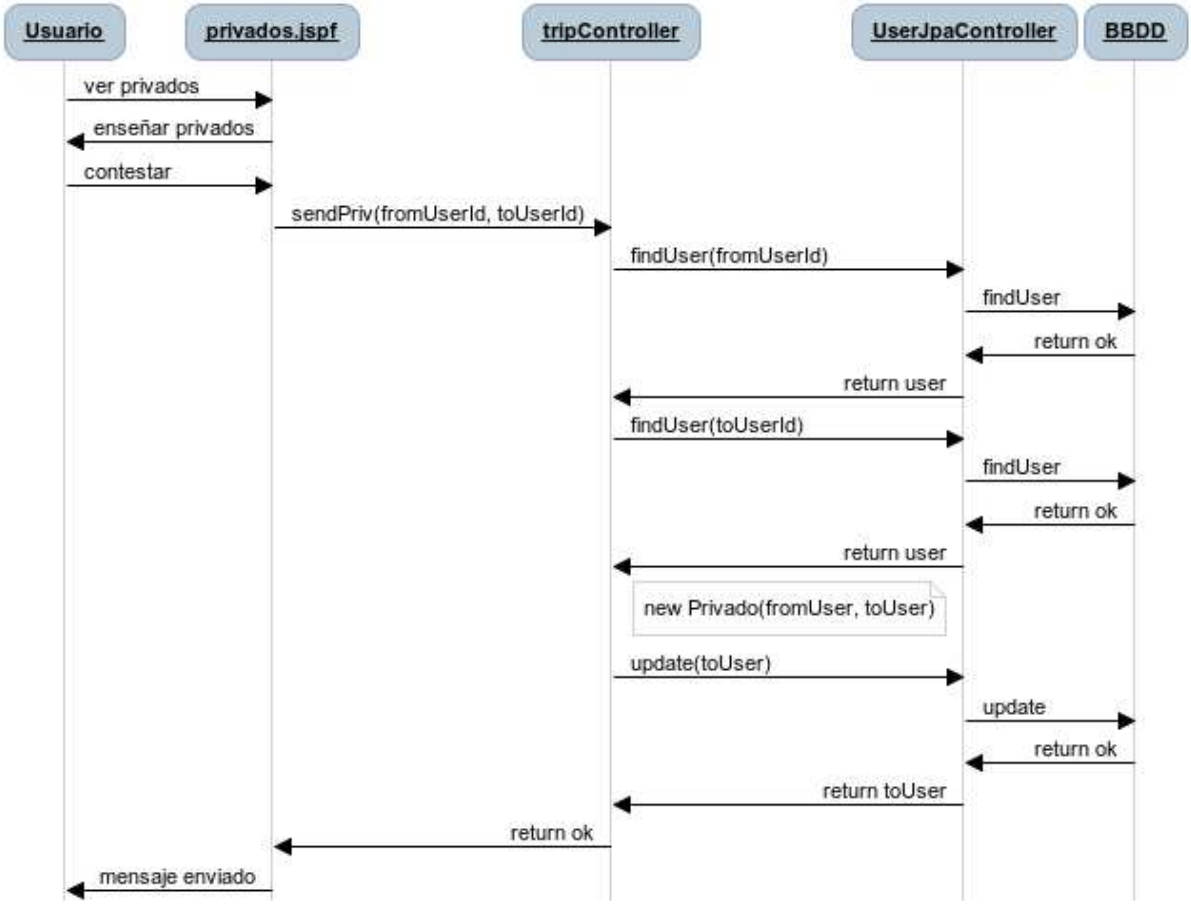
El usuario quiere contestar en alguna de las conversaciones de su canal, el usuario llegará al canal y le dará clic en el botón de contestar, entonces el sistema le presenta una ventana donde escribe su contestación, ese mensaje llega al tripController que verifica si el usuario es correcto y si el viaje es correcto, y entonces crea el mensaje en la conversación del viaje adecuado.

Ver mensajes privados



El usuario quiere ver los mensajes privados que tiene, directamente haciendo clic en ese apartado, la vista privados.jspf se los enseña. Hay que decir que esta vista ha recibido previamente del servidor todos los privados del usuario y los ha cargado en la página pero están ocultos hasta que el usuario desea verlos.

Contestar un mensaje privado



El usuario quiere contestar a uno de sus mensajes privados, y hace clic en el botón de contestar esto hace que le llegue al controlador, tripController la petición de crear un nuevo privado del usuario hacia otro usuario distinto, el sistema comprueba que los usuarios son correctos y entonces crear el privado, una vez hecho esto actualiza el usuario destino, para que pueda ver su nuevo mensaje privado.

Estructura de base de datos

Como expliqué anteriormente en este proyecto los objetos modelo son también las tablas de la base de datos, y por tanto la estructura de base de datos es exactamente la misma.

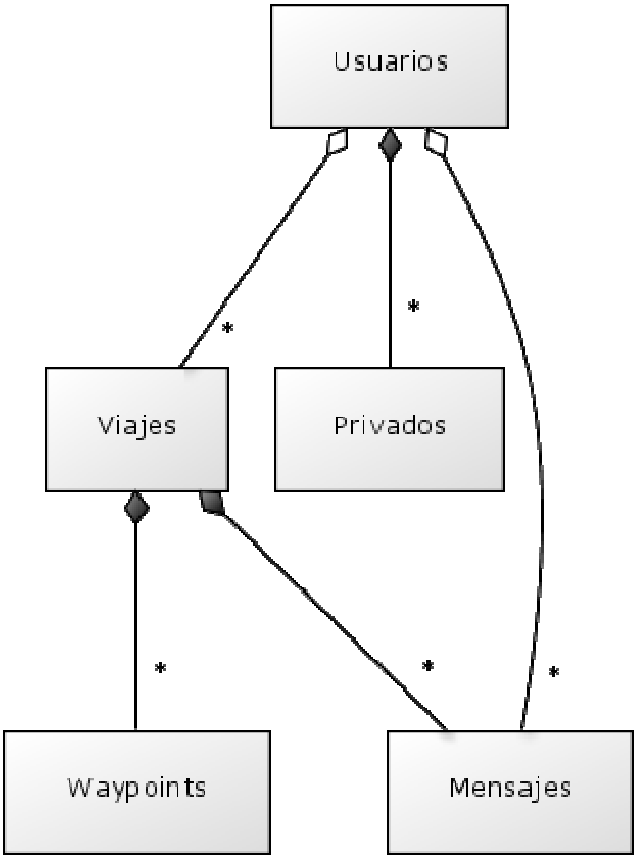


Tabla Usuarios

esta tabla es la encargada de guardar todos los datos del usuario.

Atributos

user_id: será la clave primaria, de tipo int y auto incremental

security_code: se guardará el Hash MD5 para la validación del usuario, cuando se necesite activar uno nuevo.

login: Varchar que guarda el nombre con el que el usuario accederá a la aplicación

password: clave que necesita introducir el usuario al intentar acceder a la aplicación.

nombre: nombre del usuario dentro de la aplicación.

apellido1 y apellido2: apellidos del usuario, no son imprescindibles.

email: Este campo guarda el email, del usuario donde se le enviará el correo de activación de su usuario, además en un futuro se le podrían hacer llegar promociones.
active: sirve para comprobar si este usuario está activo en la aplicación.

Tabla Privados

Esta tabla representa a un mensaje desde un usuario a otro usuario.
Para crear la relación entre los dos usuarios el que envía el mensaje y el que lo va a recibir se utilizan dos claves foráneas en los campos from_id y to_id de la tabla, que harán referencia a los user_id de la tabla usuarios del usuario que envía y del usuario que lo recibe.

Atributos
privado_id: clave primaria, de tipo int, auto incremental.
from_id: clave foránea a la tabla usuarios, representa al usuario que envió el mensaje privado.
to_id: clave foránea a la tabla usuarios, que representa al usuario que debe recibir el mensaje privado.
descripcion: el mensaje propiamente dicho.
hora: es la representación en texto de la hora a la que se envió el mensaje.

Tabla Mensajes

Esta tabla es la representación de los mensajes que los diferentes usuarios irán dejando en una conversación dentro de un viaje. Tiene dos claves foráneas una hacia la tabla viaje y la otra a la tabla usuarios.

Atributos
mensaje_id: clave primaria, de tipo int, y auto incremental
from_id: clave foránea a la tabla usuarios, es el usuario que escribo este mensaje.
viaje_id: clave foránea a la tabla viaje, es el viaje dónde se ha escrito este mensaje.
descripcion: el mensaje propiamente dicho.
hora: es la representación en texto de la hora a la que se envió el mensaje.

Tabla Viajes

Esta tabla contiene la información necesaria para el control de un viaje (origen, destino, etc.). Además tiene una clave foránea a la tabla usuario, para controlar quien lo creó.

Atributos

viaje_id: clave primaria, de tipo int, auto incremental.
user_id: clave foránea a la tabla usuarios, es el usuario que creó el viaje, y por tanto su propietario.
origen: es la dirección del origen de este viaje.
destino: es la dirección del destino de este viaje.
hora_salida: es la hora a la que el usuario, normalmente comienza este viaje.
hora_llegada: es la hora a la que el usuario, normalmente llega al destino.
descripcion: esta campo se utiliza para que el propietario pueda hacer algún comentario sobre la ruta o lo que crea adecuado.

Tabla Waypoints

Esta tabla representa a un punto intermedio en un viaje y contiene la dirección de este punto intermedio. Además de una clave foránea a la tabla viaje para saber que viaje lo creó.

Atributos

waypoint_id: clave primaria, de tipo int, auto incremental
viaje_id: clave foránea a la tabla viaje, es el viaje donde pertenece este punto intermedio.
ruta: dirección de este punto intermedio.

Dificultades y Resultados

El primer gran reto fue la configuración de SPRING tuve que leer mucho sobre los ficheros de configuración del SPRING y sobre todas las posibilidades de configuración que tiene. También aprendiendo todas las anotaciones que se utilizan para configurar los controladores SPRING y que respondieran correctamente a las URLs que tenían configuradas, además de aprender a hacer las llamadas asíncronas (Ajax) al servidor y como devolver los datos hacia la vista que había hecho la petición para que pudiera enseñar los resultados. Resultó difícil conseguir hacer funcionar JPA, ya que no sabía que era un persistence.xml al inicio de este proyecto, muchas veces no funcionó como esperaba y no veía los resultados reflejados en la base de datos.

Como resultado del proyecto he podido conseguir lo que esperaba al inicio, además he podido probar la aplicación en diferentes navegadores, como son Firefox, Chrome e Internet Explorer 9, en todos ellos se obtiene una experiencia de uso igual, al ser una interfaz gráfica simple y corta no da demasiados problemas, el único problema que tuve fue con las barras laterales que en Internet Explorer se continuaban viendo mientras en que los otros navegadores no. La aplicación no se ha podido probar desde un servidor de verdad, ya que los servidores de aplicaciones no son tan baratos como cualquier servidor. Pero se hicieron pruebas desde ordenadores en la misma red, accediendo a la aplicación (haciendo de servidor mi portátil), enviándose mensajes, consultando viajes etc.

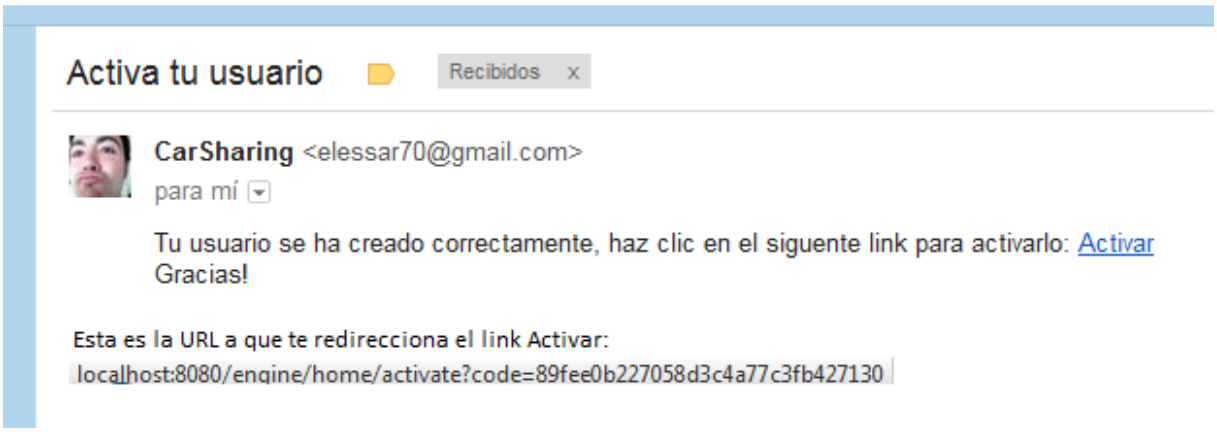
Capturas

A continuación muestro algunas capturas de la aplicación.

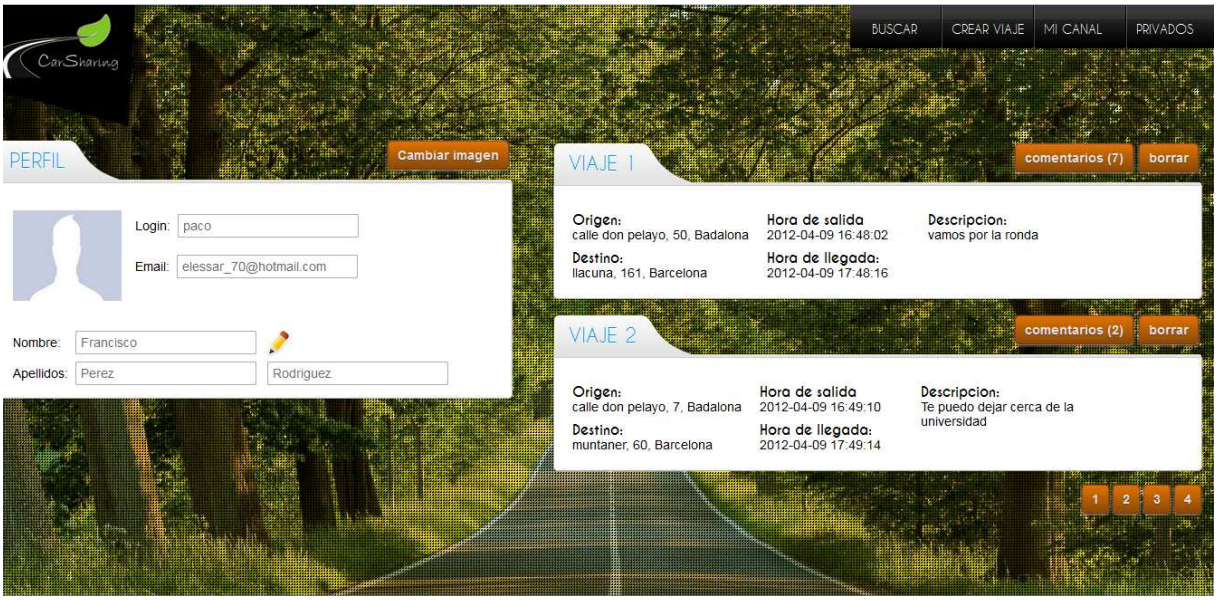
Login

Registro

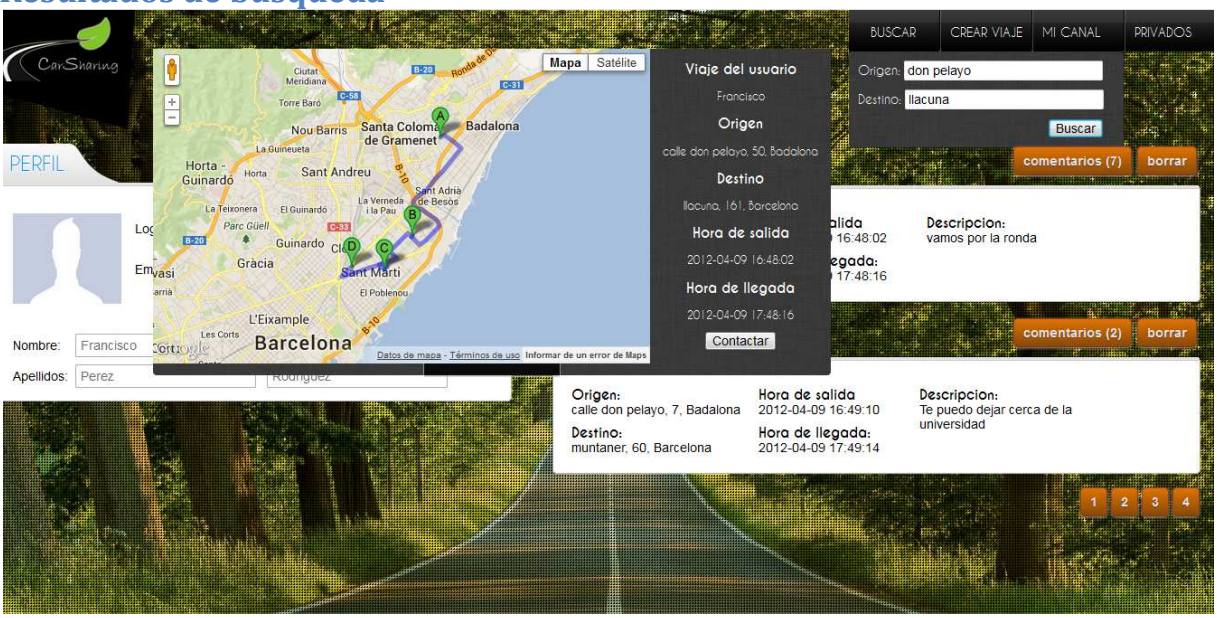
Email de activación recibido



Home



Resultados de búsqueda



Formulario de creación de viaje

Conclusiones

He conseguido una aplicación web que trabaja sobre un servidor de aplicaciones Glassfish, usando Spring MVC para la gestión de los controladores web, además se utiliza JSPs para crear las diferentes vistas de la aplicación, y por último JPA para gestionar las conexiones con la base de datos.

Los siguientes pasos de la aplicación serían internacionalizar la aplicación mediante el uso del componente SPRING que permite utilizar ficheros .properties para gestionar los textos que se muestran en la aplicación ya que estos textos una vez hecha esta configuración cambiarían en función del *Locale* del cliente, es decir si SPRING detecta que el navegador que utiliza el cliente es Alemán automáticamente configuraría los textos en alemán, ya he investigado sobre este tema y los ficheros .properties tiene un nombre de este estilo messages_en.properties (en el caso del texto en inglés, se sigue la ISO_639-1 para la parte que continua después del _ en los nombres de los ficheros .properties, como podrían ser _es, _de, _tr, etc.), además continuar enriqueciendo el buscador de viajes. Crear una aplicación móvil llamada por ejemplo ShareInstant que sería capaz de utilizar los datos del GPS de tu móvil y recomendarte los viajes que tienes más cercanos, además de poder gestionar el resto de apartados de la aplicación web, desde el móvil. Para ello se tendría que crear una API para que la aplicación móvil pudiera comunicarse con el servidor central y poder llevar a cabo todas sus funciones. También se podría integrar la conexión con Facebook y Google+ para en vez de crear usuarios, y usar los que ya tienen en esas redes, para así poder enviar los mensajes a través de esas redes.

Anexo: Ficheros de configuración

Web.xml

Las aplicaciones web Java utilizan un archivo de descriptor de implementación para determinar la forma en que las URL se asignan a los servlets y las direcciones URL que necesitan autenticación, entre otra información. Este archivo se llama web.xml y se encuentra en el WAR de la aplicación, concretamente, en el directorio WEB-INF/. web.xml forma parte del estándar de Servlet para las aplicaciones web.

```
<?xml version="1.0" encoding="UTF-8"?><web-app version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
```

Aquí se indica donde está el fichero de configuración del contexto de SPRING

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/web-application-context.xml</param-value>
</context-param>
```

Se coloca un listener que escucha al despliegue de la aplicación y entonces cargará el contexto SPRING

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Se añaden filtros para la correcta codificación de los parámetros recibidos en los controladores

```
<filter>
  <filter-name>encoding-filter</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encoding-filter</filter-name>
  <url-pattern>/*</url-pattern>
```

```
  </filter-mapping>
  <filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>UrlRewriteFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

Se configure la clase SPRING que hará de Servlet Dispatcher, y se le da un nombre, además se configura la URL que escuchará.

```
  <servlet>
    <servlet-name>Dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Dispatcher</servlet-name>
    <url-pattern>/web/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Configuración de SPRING

SPRING necesita configurar un fichero que yo llamé web-application-context.xml, es MUY importante tener correctamente importados los xmlns ya que si no la aplicación no funcionará como me pasó a mí y me costó mucho encontrar que tenía un fallo en la url.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
```

Esta propiedad declara que será necesario el soporte para las anotaciones de controlador SPRING, como pueden ser @Controller, @PeticiónMapping, entre otras.

```
  <mvc:annotation-driven />
```

Esta propiedad declara que será necesario el soporte para las anotaciones generales de SPRING, como pueden ser @Required, @Autowired, entre otras.

```
<context:annotation-config />
```

Estas propiedades que vienen a continuación trabajan juntas para conseguir internacionalizar la aplicación.

messageSource controla la ubicación de los ficheros de propiedades, que son los que tienen todo el texto de la aplicación.

```
<bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
   <property name="basename" value="/WEB-INF/messages/messages" />
   <property name="defaultEncoding" value="UTF-8"/>
</bean>
```

localeChangeInterceptor es el encargado de cambiar el locale del navegador si se diera el caso que no tenemos ese idioma en los ficheros de properties.

```
<bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
   <property name="paramName" value="locale" />
</bean>
```

localeResolver es quien identifica el locale que está utilizando el navegador

```
<bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
   <property name="defaultLocale" value="es"/>
</bean>
```

Este HadlerMapping que he implementado sirve para controlar la petición HTTP y que se lance el localeChangeInterceptor, para que se pueda obtener el locale del navegador y conseguir ofrecerle al cliente los textos en el idioma que está usando en su navegador.

```
<bean id="handlerMapping"
class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
   <property name="interceptors">
      <ref bean="localeChangeInterceptor" />
   </property>
</bean>
```

Con esta propiedad hacemos que SPRING escanee los paquetes en busca de todas las clases anotadas con @Controller y los dejará preparados para que puedan recibir peticiones.

```
<context:component-scan base-package="web.controllers" />
```

```
  <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"/>
  <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />
```

Esta propiedad es la encarga de encontrar nuestras vistas

```
  <bean name ="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/pages/" />
    <property name="suffix" value=".jsp" />
  </bean>
```

Estas propiedades son las que llevan a cabo las peticiones a la base de datos
El valor más importante es el nombre de la unidad de persistencia, ese nombre será el que debamos poner en nuestro persistence.xml ¡sino no funcionara!

```
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/engine"/>
    <property name="username" value="root"/>
    <property name="password" value=""/>
  </bean>

  <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="enginePU" />
  </bean>
</beans>
```

Configuración JPA

La configuración JPA se hace dentro del fichero persistence.xml que explique en el apartado de base de datos y tiene este formato. Lo más destacable es el nombre de la unidad de persistencia como acabo de comentar anteriormente, y la URL del servidor de base de datos, así como la configuración del correcto usuario sino no se podrán ejecutar correctamente las consultas.

```
 <persistence-unit name="enginePU" transaction-type="RESOURCE_LOCAL">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <properties>
   <property name="javax.persistence.jdbc.user" value="root"/>
   <property name="javax.persistence.jdbc.password" value=""/>
   <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
   <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/engine"/>
   <property name="dialect" value="org.hibernate.dialect.MySQLDialect"/>
   <property name="eclipselink.logging.level" value="INFO"/>
   <property name="eclipselink.cache.size.default" value="5000"/>
   <property name="eclipselink.jdbc.connections.initial" value="8"/>
   <property name="eclipselink.jdbc.connections.min" value="8"/>
   <property name="eclipselink.jdbc.connections.max" value="8"/>
  </properties>
 </persistence-unit>
```

Create Tables

```
CREATE TABLE `usuarios` (
	`user_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
	`security_code` varchar(40) NOT NULL DEFAULT '',
	`login` varchar(25) NOT NULL,
	`password` varchar(40) NOT NULL,
	`nombre` varchar(20) DEFAULT NULL,
	`apellido1` varchar(20) DEFAULT NULL,
	`apellido2` varchar(20) DEFAULT NULL,
	`email` varchar(100) DEFAULT NULL,
	`active` tinyint(1) NOT NULL DEFAULT '0',
	PRIMARY KEY (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1

CREATE TABLE `viaje` (
	`viaje_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
	`user_id` int(10) unsigned DEFAULT NULL,
	`origen` varchar(100) DEFAULT NULL,
	`destino` varchar(100) DEFAULT NULL,
	`hora_salida` varchar(100) DEFAULT NULL,
	`hora_llegada` varchar(100) DEFAULT NULL,
	`descripcion` varchar(250) DEFAULT NULL,
	PRIMARY KEY (`viaje_id`),
	KEY `user_id` (`user_id`),
	CONSTRAINT `viaje_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `usuarios` (`user_id`) ON
	DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=latin1

CREATE TABLE `mensajes` (
	`mensaje_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
	`from_id` int(10) unsigned NOT NULL,
	`viaje_id` int(10) unsigned NOT NULL,
	`descripcion` varchar(250) DEFAULT NULL,
	`hora` varchar(100) NOT NULL DEFAULT '',
	PRIMARY KEY (`mensaje_id`),
	KEY `from_id` (`from_id`),
	KEY `viaje_id` (`viaje_id`),
	CONSTRAINT `mensajes_ibfk_1` FOREIGN KEY (`from_id`) REFERENCES `usuarios` (`user_id`)
	ON DELETE CASCADE,
	CONSTRAINT `mensajes_ibfk_2` FOREIGN KEY (`viaje_id`) REFERENCES `viaje` (`viaje_id`) ON
	DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=latin1
```

```
CREATE TABLE `waypoints` (
	`waypoint_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
	`viaje_id` int(10) unsigned NOT NULL,
	`ruta` varchar(250) NOT NULL,
	PRIMARY KEY (`waypoint_id`),
	KEY `viaje_id` (`viaje_id`),
	CONSTRAINT `waypoints_ibfk_1` FOREIGN KEY (`viaje_id`) REFERENCES `viaje` (`viaje_id`)
	ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1

CREATE TABLE `privados` (
	`privado_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
	`from_id` int(10) unsigned NOT NULL,
	`to_id` int(10) unsigned NOT NULL,
	`descripcion` varchar(250) NOT NULL,
	`hora` varchar(100) NOT NULL,
	PRIMARY KEY (`privado_id`),
	KEY `from_id` (`from_id`,`to_id`)
	CONSTRAINT `privados_ibfk_1` FOREIGN KEY (`from_id`) REFERENCES `usuarios` (`user_id`)
	ON DELETE CASCADE
	CONSTRAINT `privados_ibfk_2` FOREIGN KEY (`to_id`) REFERENCES `usuarios` (`user_id`) ON
	DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1
```

Requisitos básicos

Cito a continuación los requerimientos básicos que tiene la aplicación:

1. Necesitarás un **servidor Glassfish 3.1** o superior, se puede utilizar el que integra por defecto el entorno de desarrollo Netbeans.
2. Es necesario disponer de un **servidor MySQL** que controle nuestra base de datos.
3. **Librerías SPRING MVC 3** o superior
4. **Librerías Jackson 1.9** o superior (necesaria para manejar los datos JSON)
5. **Librerías JSTL 1.1** o superior (necesarias para las JSPs)

Bibliografía

http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29

http://es.wikipedia.org/wiki/Spring_Framework

http://www.springsource.org/

http://www.davidmarco.es/tutoriales/spring-mvc-sbs/

http://es.wikipedia.org/wiki/JPA

http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html

https://developers.google.com/appengine/docs/java/config/webxml?hl=es

http://es.wikipedia.org/wiki/JavaServer_Pages

http://www.objectdb.com/api/java/jpa/annotations