CarSharing MIJE

Implementación y diseño

Índice de contenido

1 Especificaciones

El proyecto completo CarSharing del grupo MIJE (Merce, Ivan, Jorge y Emilio) se encuentra disponible en :

https://github.com/scaballe/MIJE-PDS2

Sin embargo para llegar al producto final se han ido superando diferentes etapas y se ha decidido ir creando carpetas dentro de este espacio para guardar el histórico de las mismas y que sirvan como referencia. Así pues la última versión se encuentra en :

https://github.com/scaballe/MIJE-PDS2/tree/master/carsharing2/carsharing2

2 Preparación entorno

2.1 Herramientas

Para obtener la versión y poder generar el producto correcto se deben de cumplir los siguientes requisitos :

- Java JDK 1.7 o superior
- Eclipse 4.0 o superior
- Maven 3.0 o superior
- Cliente Git
- Jboss AS 7.1.1
- PostgreSQL

Para el correcto despliegue las herramientas deben estar instaladas y configuradas de acuerdo con sus respectivas especificaciones, en particular la variable de entorno JBOSS_HOME debe apuntar al directorio de instalación del mismo.

2.2 PostgreSQL

El motor de base de datos debe de haber sido configurado siguiendo las instrucciones proporcionadas por el profesorado en el documento del laboratorio «Tutorial_instalación._Laboratorio_PDS_v2.0.pdf»

En concreto debe existir una base de datos «postgre» y un usuario con permisos para crear, borrar, etc las tablas.

Así mismo para la ejecución del entorno de test será necesario una base de datos adicional «mijetest» y un usuario «mije» / «mije» con permisos sobre ella.

2.3 Preparación JBoss

El servidor de aplicaciones Jboss debe de haber sido configurado siguiendo las instrucciones proporcionadas por el profesorado en el documento del laboratorio «Tutorial_instalación._Laboratorio_PDS_v2.0.pdf».

A parte de la configuración del driver postgresql se debe configurar un JNDI para el dataSource «java:jboss/postgresDS» que apunte a la base de datos anterior.

En concreto la configuración debe quedar como se muestra a continuación:

```
<datasource jta="false" jndi-name="java:jboss/postgresDS" pool-name="postgresDS" enabled="true" use-java-context="true" use-ccm="false">
        <connection-url>jdbc:postgresql://localhost:5432/postgres</connection-url>
        <driver-class>org.postgresql.Driver</driver-class>
        <driver>postgresql</driver>
            <security>
                <user-name>postgres</user-name>
                <password>postgres</password>
            </security>
</datasource>
<driver name="postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
</driver>
```

2.4 Preparación Maven

Para generar el producto no hace falta disponer de eclipse, sino que basta con tener el sistema de dependencias MAVEN instalado y configurado. Este se encuentra instalado por defecto en Eclipse por lo que si se va a usar este IDE se puede utilizar el que viene embebido en el mismo.

Se ha decidido utilizar Maven porque es un sistema que permite la gestión de dependencias, generación de producto y versionado del mismo de una forma mucho más simple y completa que Ant.

Una vez instalado y configurado sólo hay que ejecutar «mvn install» desde el directorio donde se encuentra el fichero pom.xml para compilar, instalar y desplegar el aplicativo en el servidor.

Así mismo si se quiere utilizar Eclipse primero se deberá importar el proyecto maven (Import / Existing maven project y seleccionar la carpeta que contiene el fichero pom.xml) y después sobre el proyecto con el botón derecho seleccionar «Run as » « Maven Install »

Cualquiera de los dos métodos descargará previamente las dependencias necesarias por lo que puede tardar unos minutos en terminar dependiendo de la conexión a Internet que se esté usando. Una vez descargadas las dependencias el proceso compilará, ejecutará los test, generará los artefactos y los copiará a la carpeta de despliegue de Jboss. Si dicho servidor se encuentra arrancado iniciará el proceso de despliegue automáticamente.

3 JTA

Para el diseño de las entidades JTA se han tomado las siguientes decisiones :

- Utilizar el mecanismo de herencia y agrupar en una única tabla « user » las entidades de driver y passenger (así mismo se podrían incluir en esta la entidade de administrador, etc).

- Establecer relaciones OneToMany entre Driver y Car así como entre Driver y Trip, de tal forma que si tenemos un Driver podamos acceder a los cars y trips que tenga en el sistema. También podremos añadir nuevos elementos a la colección mediante métodos addXXXX oportunamente creados

4 EJB

Se ha creado un nuevo EJB, UtilBean, que no se incluye en el diseño inicial. Su objetivo es centralizar aquellas funcionalidades de apoyo no contempladas por el diseño y que pueden ser útiles a todos los módulos.

Entre otras funcionalidades incluye :

- cargar un modelo de datos u otro en la base de datos para disponer de datos de pruebas. Obviamente su sentido sería para un entorno de desarrollo y/o integración, no para producción.

- ofrecer una lista de CityJPA útliles para la vista.

5 Security

Para permitir el acceso a determinadas partes de la aplicación por usuarios autentificados se ha optado por crear dos layouts (private y public). Mientras que el layout público renderiza la página sin chequear si el usuario está identificado, el layout private sí lo hace. En un desarrollo más completo dicha funcionalidad debería recaer sobre alguna librería más completa.

6 Bootstrap

Se ha creado un Bean, BootStrapApplicationListener, encargado de añadir a la base de datos los objetos que fueran necesarios para su inicialización, como son por ejemplo algunas ciudades de ejemplo y que es ejecutado por el servidor al inicializar la aplicación automáticamente.