
# Trabajo Final de Grado / JEE

## Gestión de un servicio de ofertas hoteleras desarrollado con AngularJs y Spring

Sergi Martín Sandoval

13 de enero de 2015

*Agraeixo als meus pares el seu generós esforç i sacrifici per aconseguir que jo gaudís d'aquelles oportunitats que ells només van poder veure passar de lluny.*

*I a la Silvia ... que des de fa uns anys m'acompanya, sempre amb un somriure, en aquesta i moltes d'altres empreses.*

# *Índice*

# Motivación y Objetivos del Proyecto

Una de las principales motivaciones de este proyecto era adquirir conocimiento de tecnologías que están cobrando mucho protagonismo en el desarrollo de aplicaciones web. La más actual de todas ellas sea posiblemente el Framework AngularJs, pero sin olvidar las versiones más actualizadas del ya muy consolidado Spring Framework, o la integración del Framework Hibernate con este último, para gestionar la capa de persistencia.

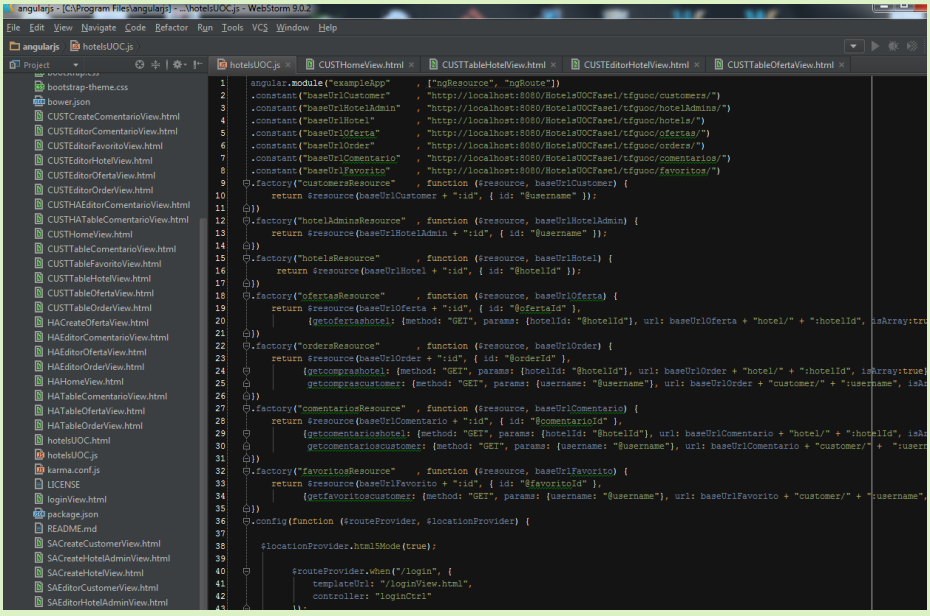
Mi conocimiento de estas tecnologías era nulo antes de empezar este proyecto y tampoco mi ámbito profesional está directamente relacionado con el desarrollo de aplicaciones web, por lo cual ha sido un estimulante desafío para mí superar la curva de aprendizaje necesaria para desarrollar un producto generado únicamente con estas tecnologías.

Considero que el objetivo fundamental del presente proyecto era además del ya mencionado proceso de aprendizaje, crear una aplicación que sirviera de base para en un futuro desarrollar partiendo de ella, ideas de negocio que dieran lugar a aplicaciones profesionales.

# Entorno de desarrollo

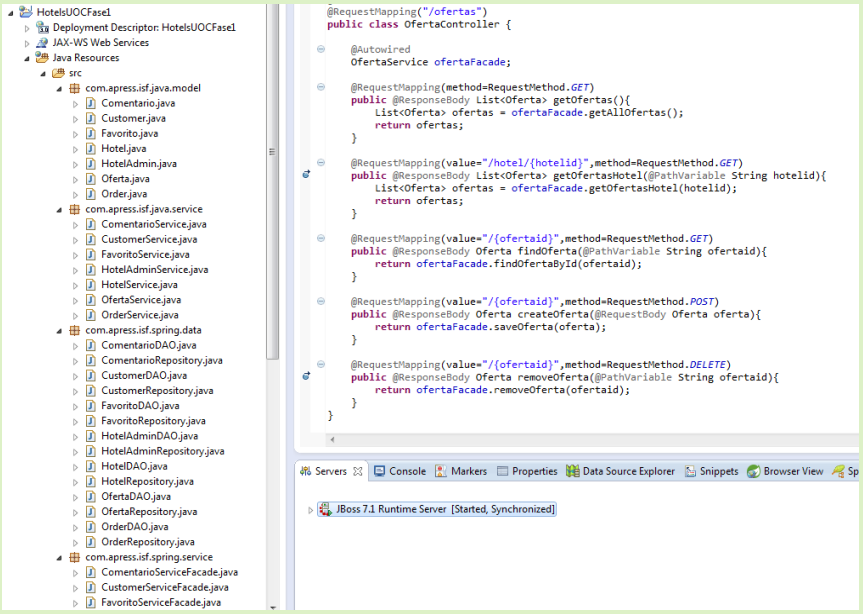
### WebStorm

Se ha utilizado este IDE para el desarrollo AngularJS. Se ha instalado Node.js en el puerto 5000. La experiencia con esta herramienta ha sido muy satisfactoria.



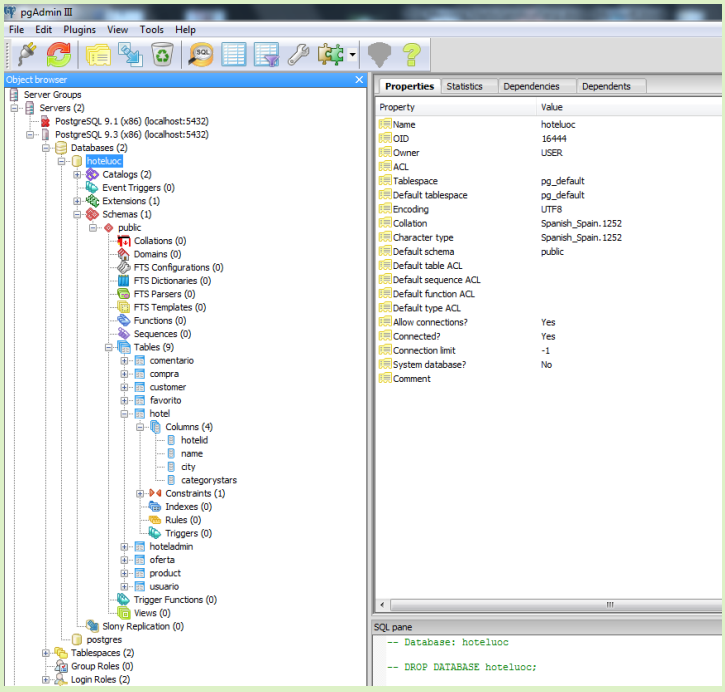
## Eclipse

Para el desarrollo Java se ha utilizado Eclipse configurado para desplegar las aplicaciones en el servidor de aplicaciones JBoss escuchando por el puerto 8080 y consultando el Sistema Gestor de Bases de Datos PostgreSQL, instalado en el puerto 5432.




## PostGreSql

Como se ha comentado en el párrafo anterior se ha trabajado con PostgreSQL para dotar de persistencia a la aplicación.
(port 5432)



# *Análisis Funcional*

A continuación se presentan siete diagramas:

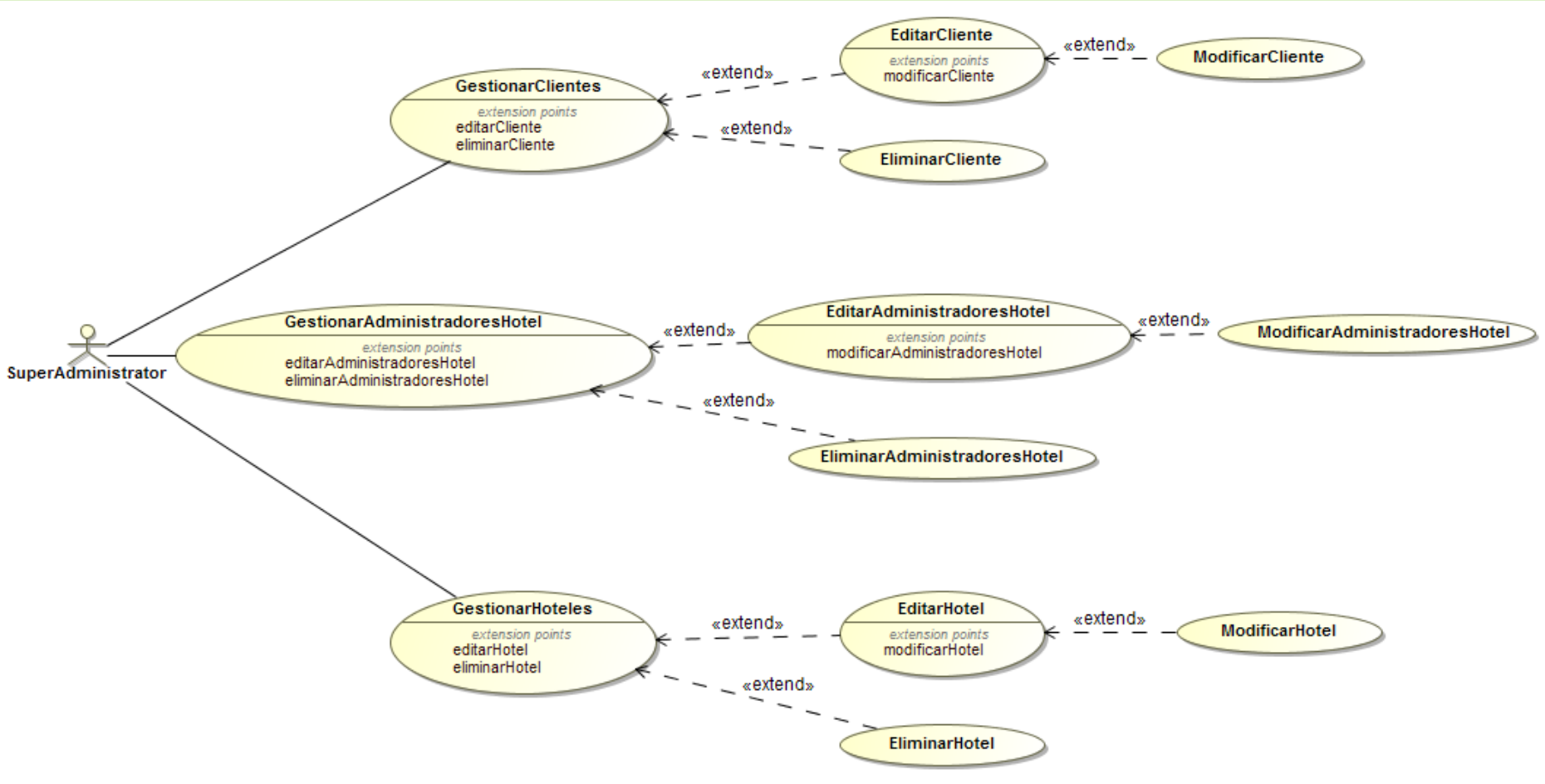
Los tres primeros diagramas de Casos de uso *SuperAdministrator*, *HotelAdministrator* y *Customer* representan de manera muy fidedigna todos los casos de uso que la aplicación implementa en el preciso momento de la Entrega Final.

El cuarto diagrama, es un diagrama de clases que fotografía el punto de vista de la información, el estado del diseño de la base de datos.

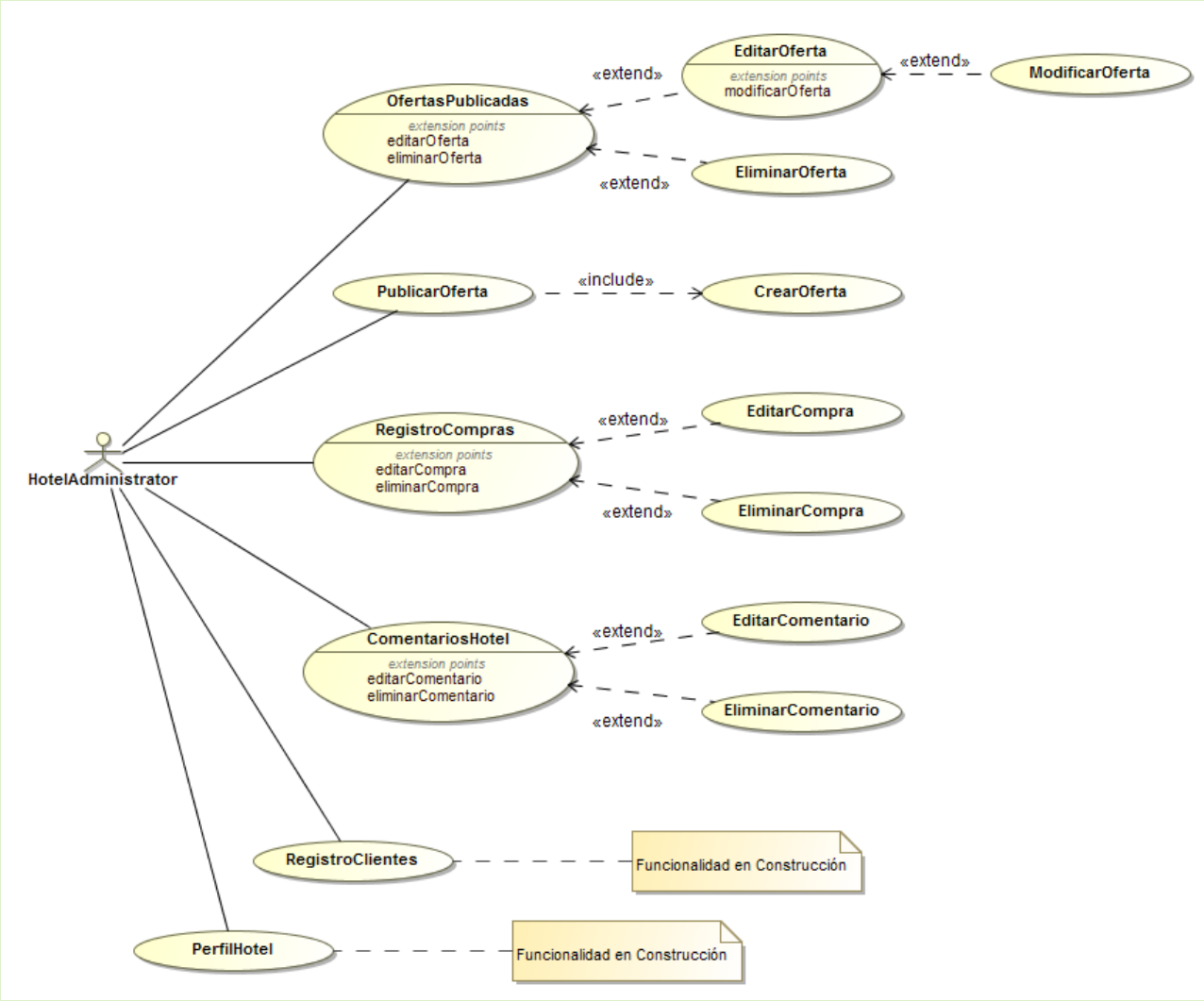
Los últimos tres diagramas representan una posible manera de ampliar la aplicación incluyendo elementos tales como vales de descuento que el hotel puede utilizar para fidelizar a sus clientes habituales e incentivar el consumo de los nuevos, notificaciones para enriquecer la experiencia en el uso de la aplicación tanto de clientes como de administradores de hoteles o detalladas puntuaciones de los clientes acerca de sus pernoctaciones para que el resto de clientes encuentre en ello un eficaz herramienta de decisión y los gerentes de los hoteles puedan reorientar sus estrategias de negocio convenientemente.

El último de los diagramas representa el aspecto del punto de vista de la información para esta posible ampliación.

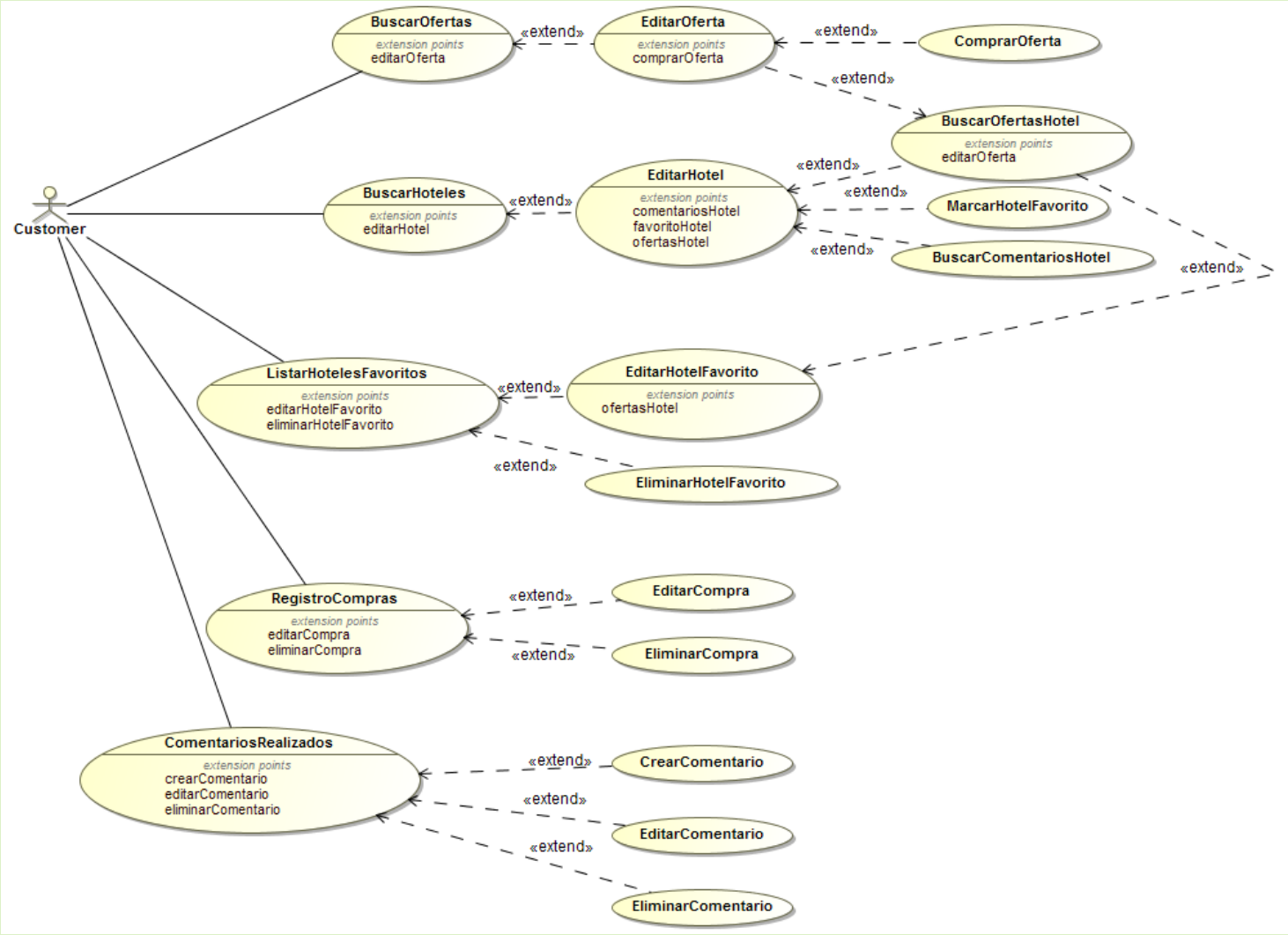
## *Diagrama Casos de uso SuperAdministrator*



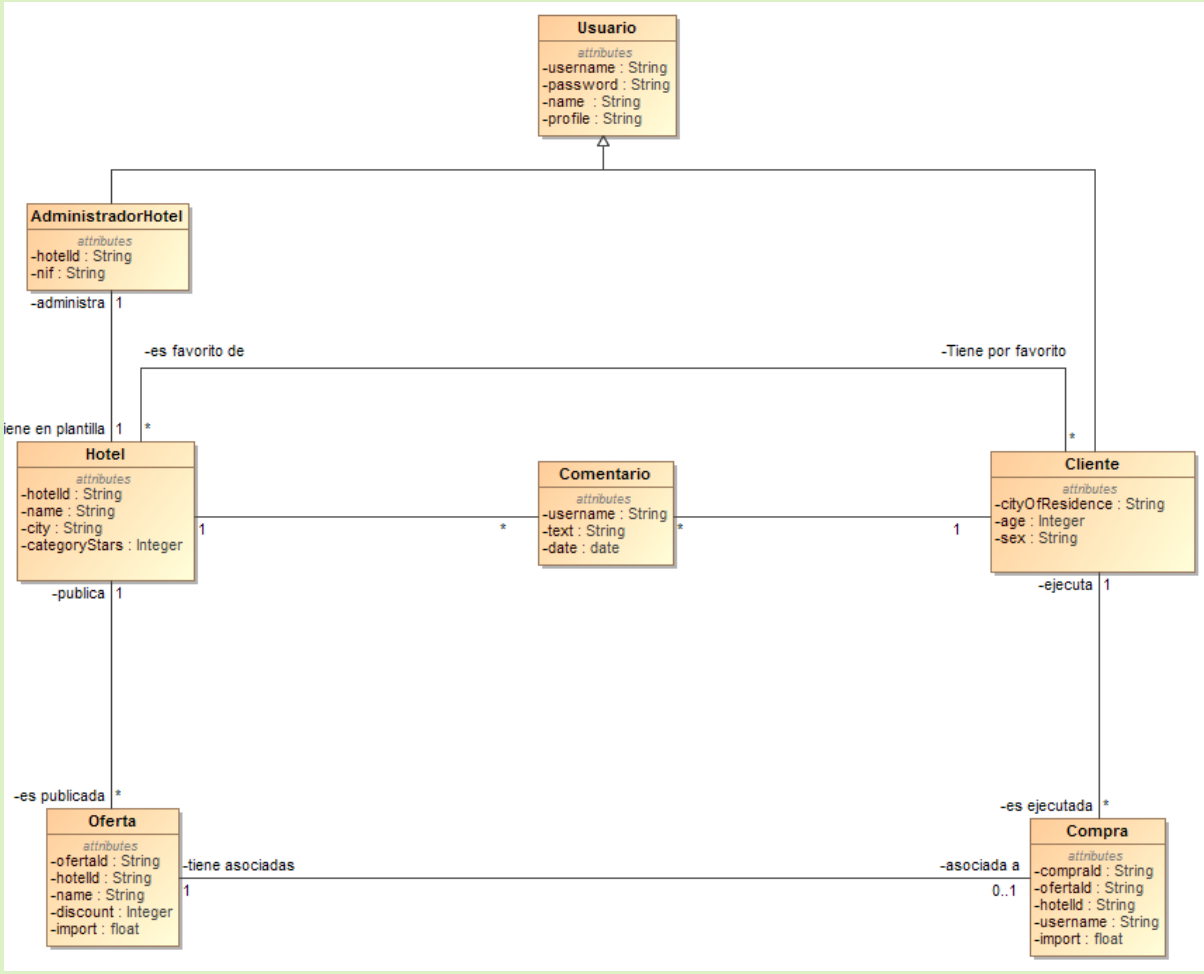
## *Diagrama Casos de uso HotelAdministrator*



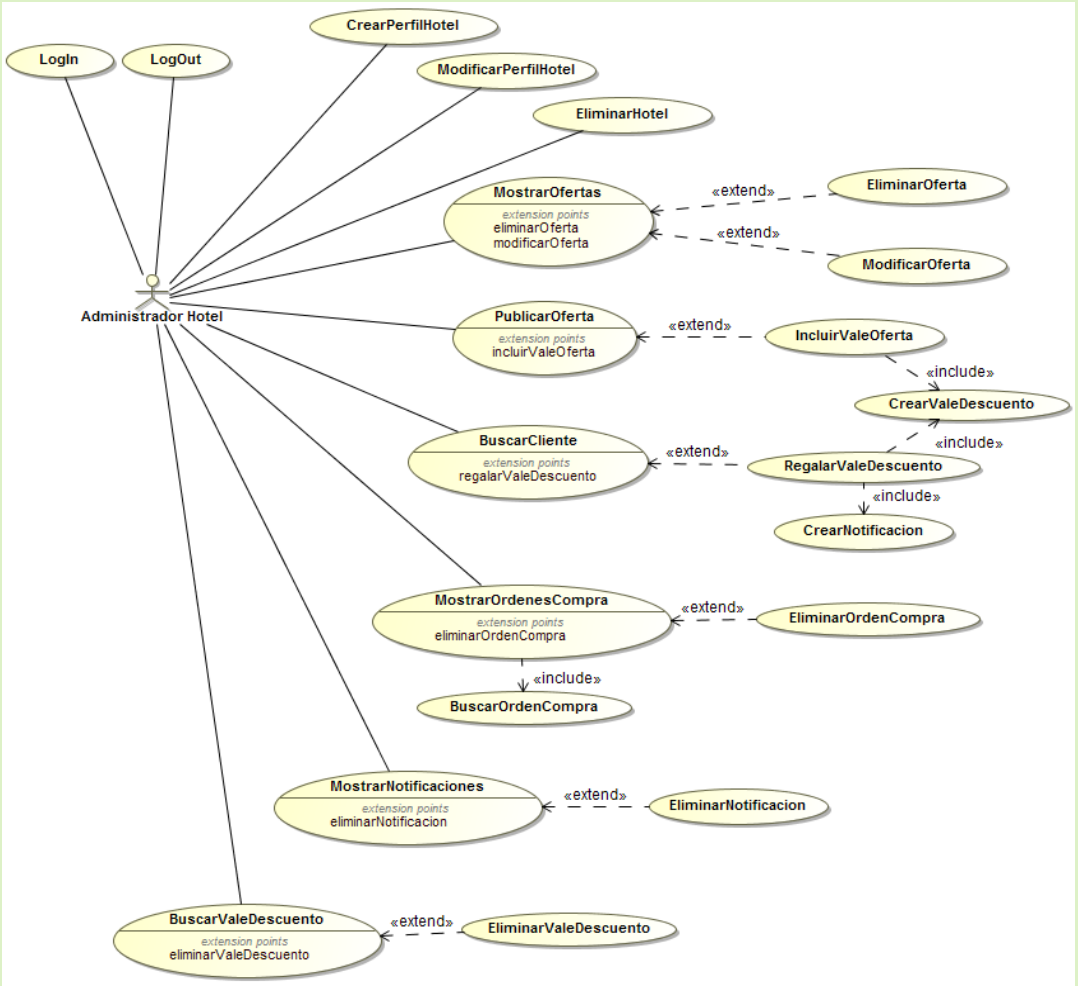
## *Diagrama Casos de uso Customer*



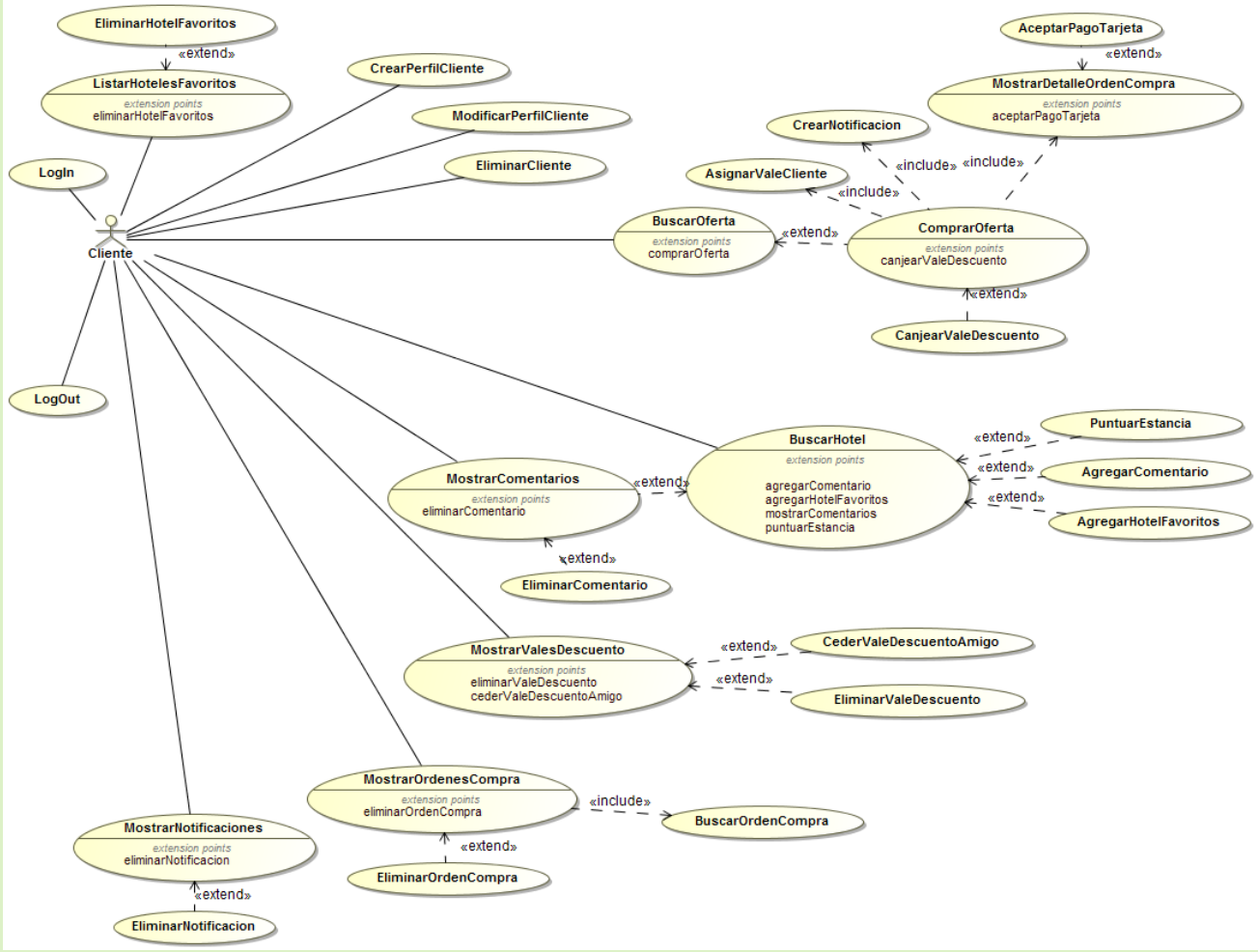
## *Puno de vista de la Información*



## *Diagrama Casos de uso HotelAdministrator (posible propuesta de ampliación)*



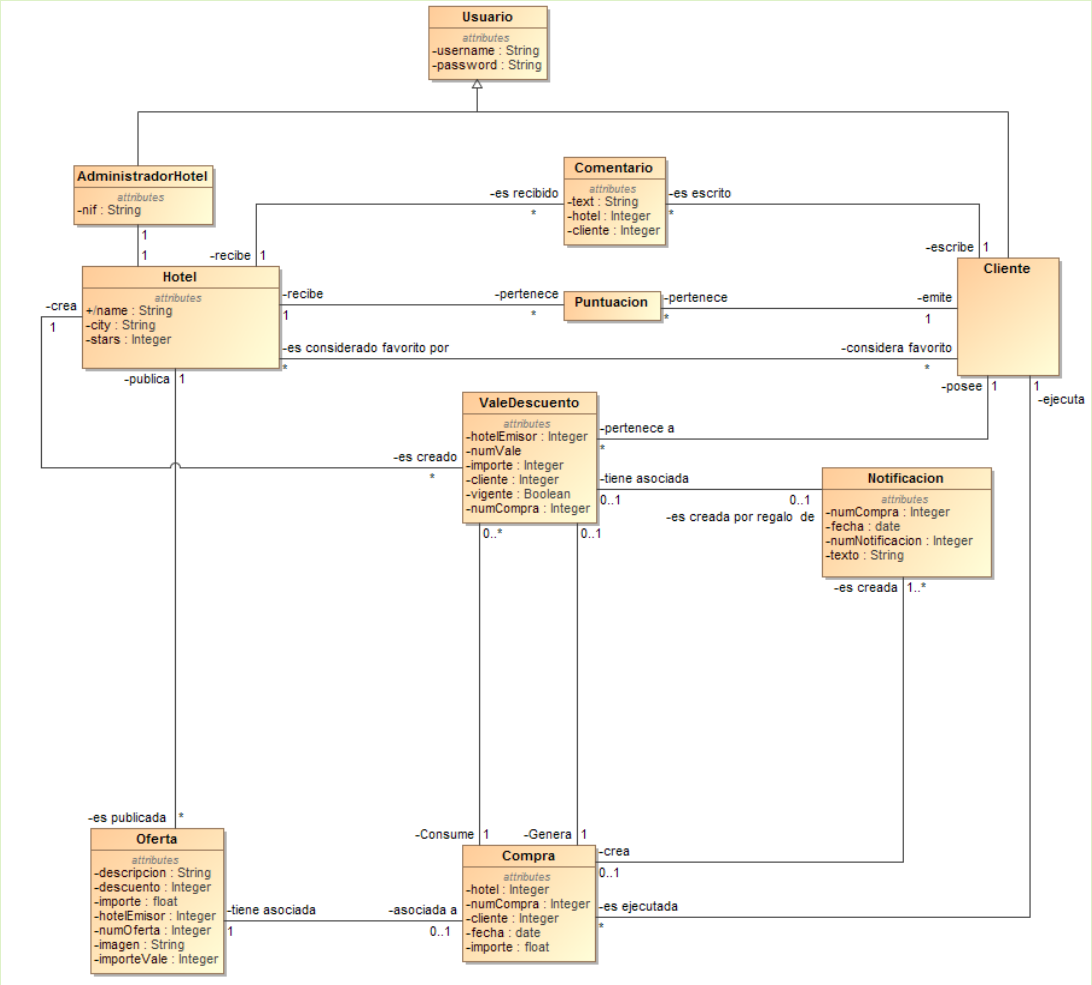
## *Diagrama Casos de uso Customer (posible propuesta de ampliación)*






Sergi Martín Sandoval 13 de enero de 2015

## *Puno de vista de la Información (posible propuesta de ampliación)*



# *Manual de Usuario*

Esta sección pretende ser un manual de usuario exhaustivo, en el que se muestren todas y cada una de las funcionalidades que la aplicación pone a disposición de los usuarios. Se inicia la sección con el proceso de login y registro y seguidamente se enumeran todas las funcionalidades que el sistema habilita para cada uno de los tres posibles perfiles: *SuperAdministrator*, *HotelAdministrator* y *Customer*.

**Nota 1.** *En la parte superior de todas las vistas mostradas, se adjunta una etiqueta que hace referencia al nombre del fichero .html correspondiente a la vista en el cliente AngularJs implementado. Estas etiquetas serán útiles en otras secciones de esta memoria.*

**Nota 2.** *La siguiente imagen constituye la parte superior de todas y cada una de las vistas, es posible que en algún screenshot no se muestre por problemas de espacio, pero aún así, es seguro que la vista la muestra durante la ejecución de la aplicación.*

**[All Profiles | hotelsUOC.html]**


### *Login(LOG)*

Esta es la primera pantalla que muestra la aplicación:

**[All Profiles | loginView.html]**


La pantalla de login ofrece dos posibilidades:

La primera de ellas es entrar las credenciales si el usuario ya se ha registrado correctamente en el sistema anteriormente . Si las credenciales son correctas la aplicación mostrará el home correspondiente al perfil del usuario, caso contrario,

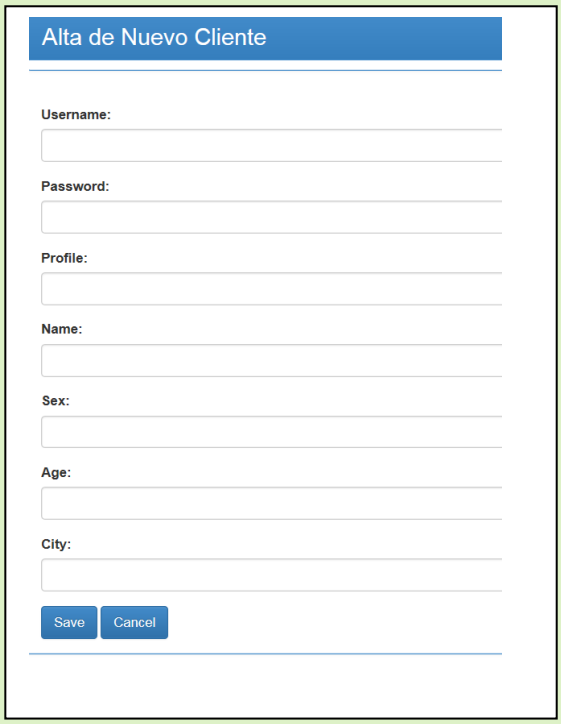
volverá a mostrar la pantalla de login con los campos Username y Password resteados.

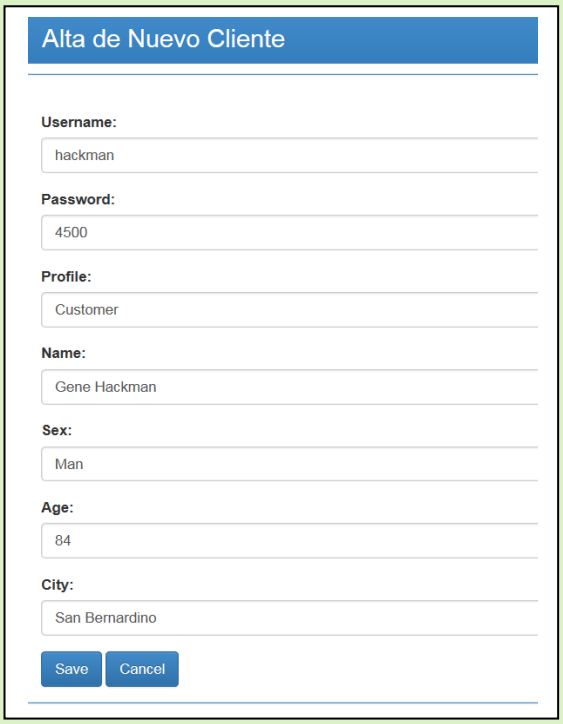
La segunda posibilidad es registrarse en el sistema, bien como un nuevo customer, bien como un nuevo HotelAdministrator.

- **Register New Customer**

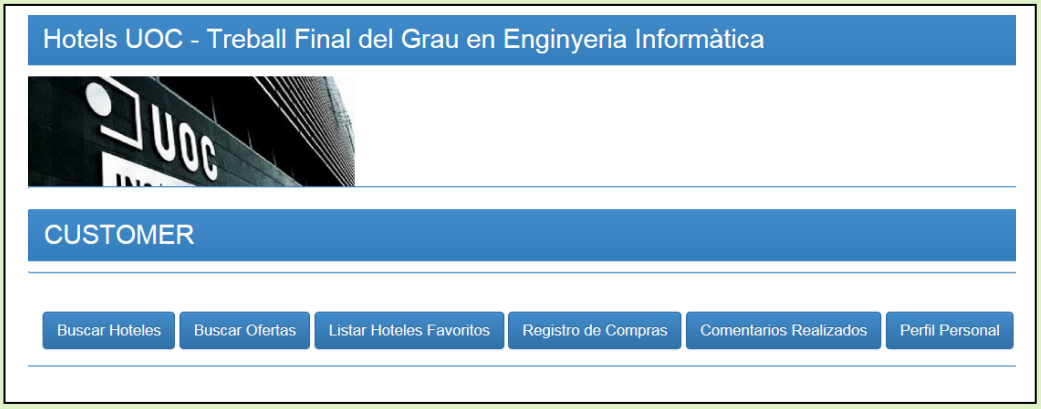
Pulsando el botón *Register New Customer,* la aplicación nos permite dar de alta un nuevo cliente. Para ello muestra una vista con todos los campos que constituyen un nuevo perfil *Customer* en blanco, para que sea el usuario quien los rellene con los datos que desee.

**[SACreateCustomerView.html]** **[SACreateCustomerView.html]**



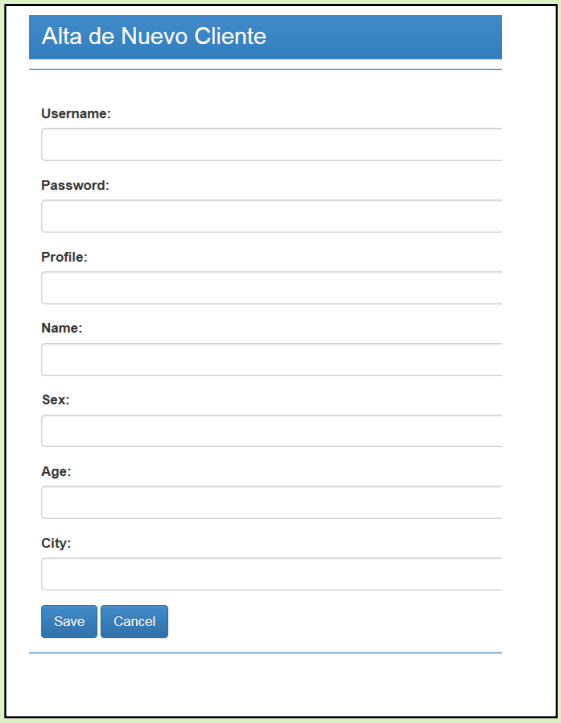

Una vez finalizado el registro se muestra la vista correspondiente al home del perfil *customer*. **[CUSTHomeView.html]**



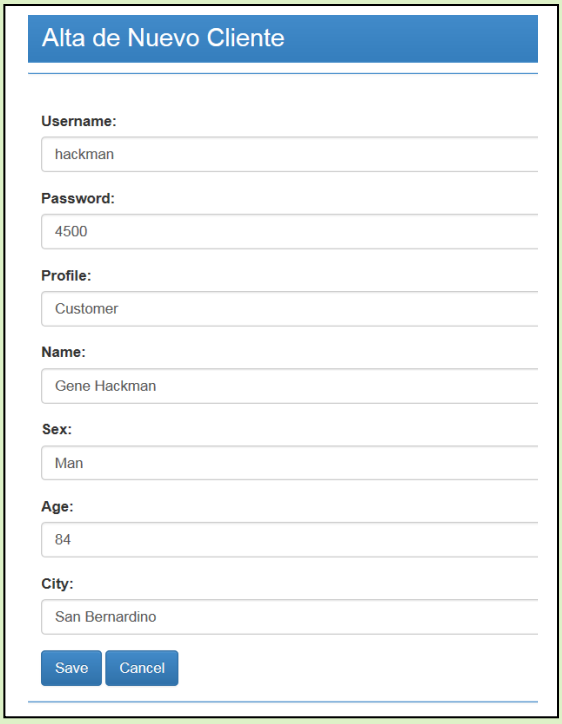
- **Register New HotelAdministrator**

Pulsando el botón *Register New HotelAdministrator,* la aplicación nos permite dar de alta un nuevo administrador de hotel. Para ello muestra una vista con todos los campos que constituyen un nuevo perfil *HotelAdministrator* en blanco, para que sea el usuario quien los rellene con los datos que desee.

**[SACreateHotelAdminView.html]**




**[SACreateHotelAdminView.html]**




Una vez finalizado el registro se muestra la vista correspondiente al home del perfil *HotelAdministrator*.

**[HAHomeView.html]**

### *Perfil SuperAdministrator (SA)*

El perfil SuperAdministrator asume la responsabilidad de gestionar los clientes, administradores de hotel y hoteles. Esta gestión se basa en la creación, modificación y eliminación de aquellas entidades que así lo requieran.

Una vez realizado exitosamente el login en el sistema, la aplicación muestra la vista correspondiente al *home* SA:

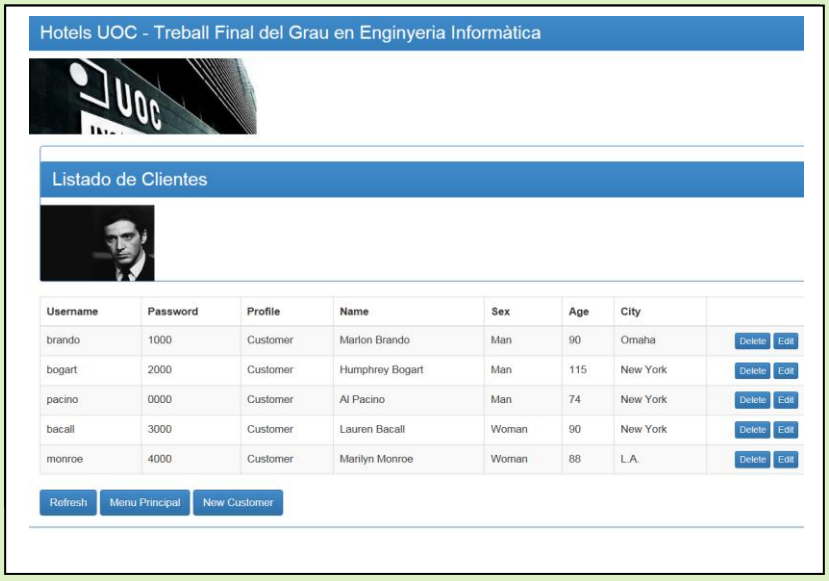
**[SAHomeView.html]**


Esta es la descripción de las funcionalidades que desarrolla el perfil SuperAdministrator:

- **Customer Management**

Al pulsar el botón *Customer Management* se despliega una lista de todos los clientes almacenados en el sistema.

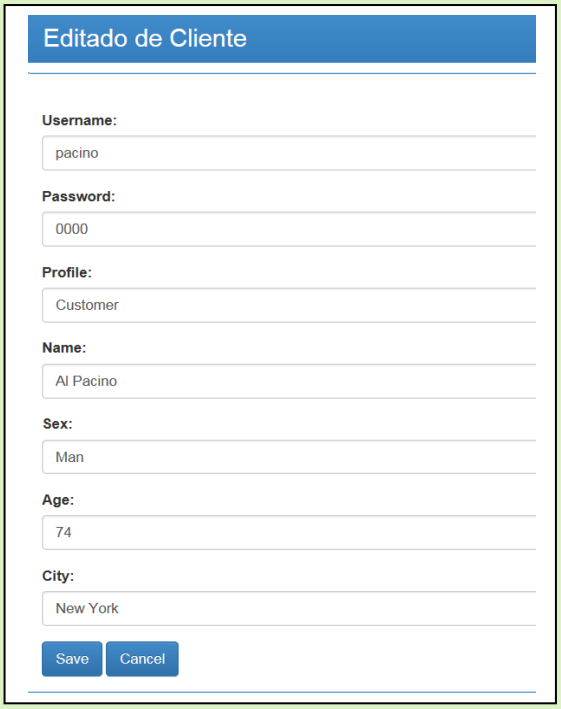
**[SATableCustomerView.html]**



Des de la vista del listado de clientes del sistema es posible:

1. Editar cualquiera de los clientes (botón *Edit* en la fila del cliente que deseemos editar)

**[SAEditorCustomerView.html]**




Una vez editado el cliente podemos realizar en él cuantas modificaciones queramos y salvarlas pulsando el botón *Save* (botón *Cancel* para abortar la acción y retornar al listado de clientes).

2. Eliminar cualquiera de los clientes (botón *Delete* en la fila del cliente que deseemos eliminar)

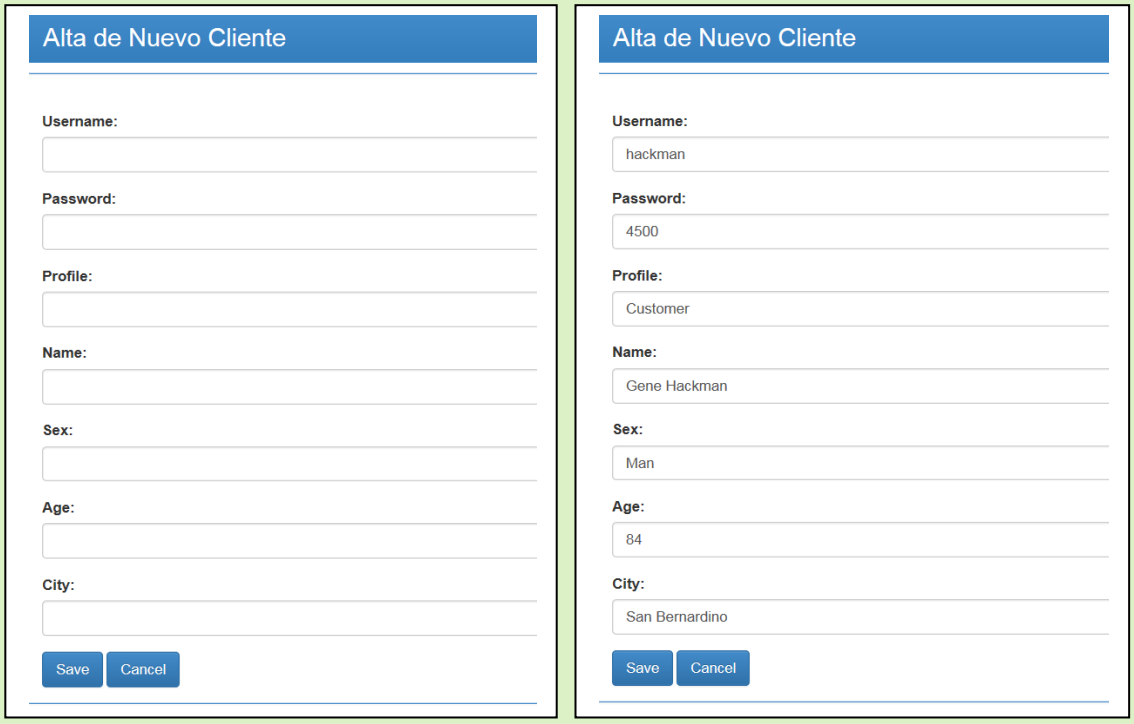
Aspecto de la lista de clientes tras eliminar el cliente con username = 'monroe' :

**[SATableCustomerView.html]**

3. Crear un nuevo cliente (botón *New Customer )*

Pulsando el botón *New Customer,* la aplicación nos permite dar de alta un nuevo cliente. Para ello muestra una vista con todos los campos que constituyen un nuevo perfil *Customer* en blanco, para que sea el SuperAdministrator quien los rellene con los datos que desee.

**[SACreateCustomerView.html]** **[SACreateCustomerView.html]**




Para confirmar la creación de un nuevo cliente se pulsa el botón Save (botón Cancel para abortar la acción y retornar al listado de clientes).

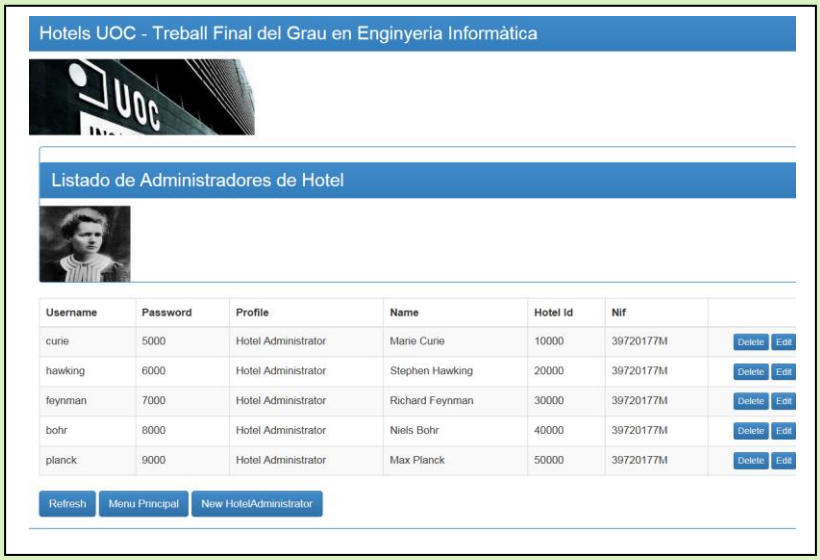
Aspecto de la lista de clientes tras crear el cliente de usrename = 'hackman':

**[SATableCustomerView.html]**

- **HotelAdministrator Management**

Al pulsar el botón *HotelAdministrator Management* se despliega una lista de todos los administradores de hotel almacenados en el sistema.

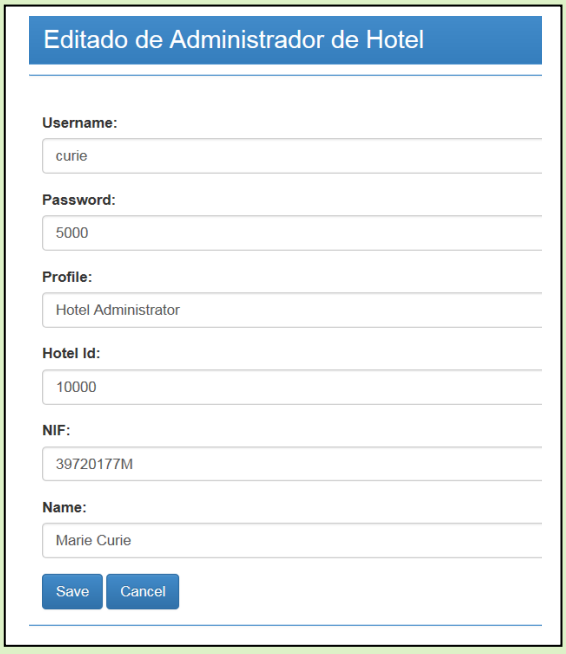
**[SATableHotelAdminView.html]**




Des de esta vista del listado de administradores de hotel del sistema es posible:

1. Editar cualquiera de los administradores de hotel (botón *Edit* en la fila del HotelAdministrator que deseemos editar)

**[SAEditorHotelAdminView.html]**




Una vez editado el administrador de hotel podemos realizar en él cuantas modificaciones queramos y salvarlas pulsando el botón *Save* (botón *Cancel* para abortar la acción y retornar al listado de administradores de hotel ).

2. Eliminar cualquiera de los administradores de hotel (botón *Delete* en la fila del HotelAdministrator que deseemos eliminar).

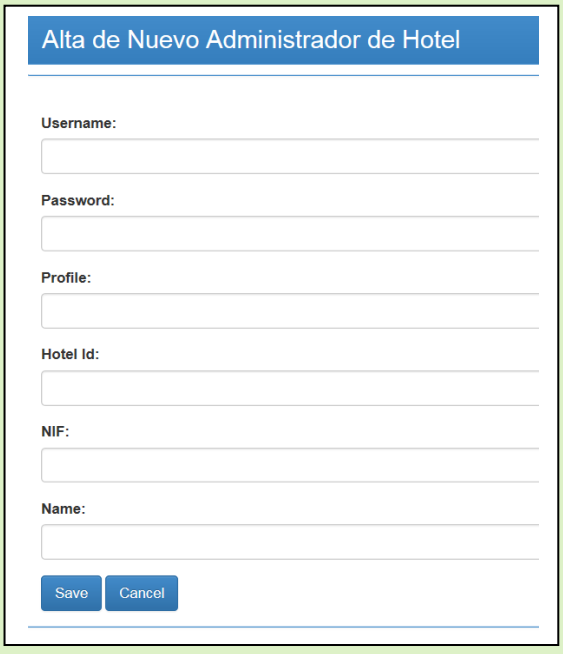
Aspecto de las lista de administradores de hotel tras eliminar el HotelAdministrator con *username* = 'planck' :

**[SATableHotelAdminView.html]**

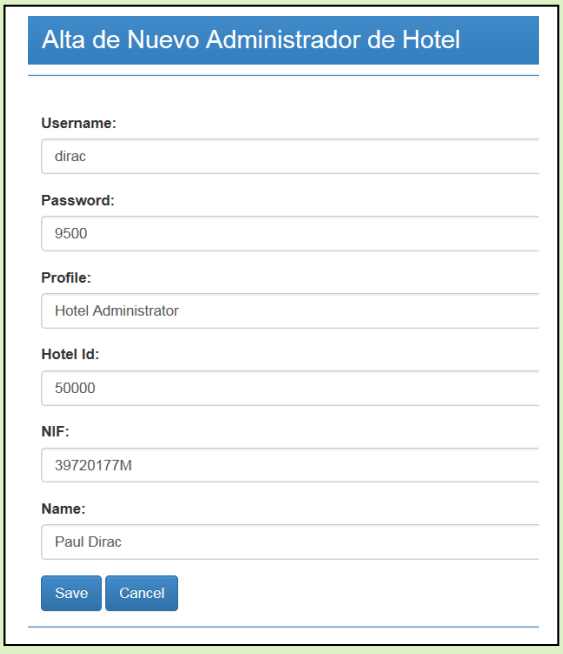

3. Crear un nuevo cliente (botón *New HotelAdministrator )*

Pulsando el botón *HotelAdministrator Management,* la aplicación nos permite dar de alta un nuevo administrador de hotel. Para ello muestra una vista con todos los campos que constituyen un nuevo perfil *HotelAdministrator* en blanco, para que sea el SuperAdministrator quien los rellene con los datos que desee.

**[SACreateHotelAdminView.html]**



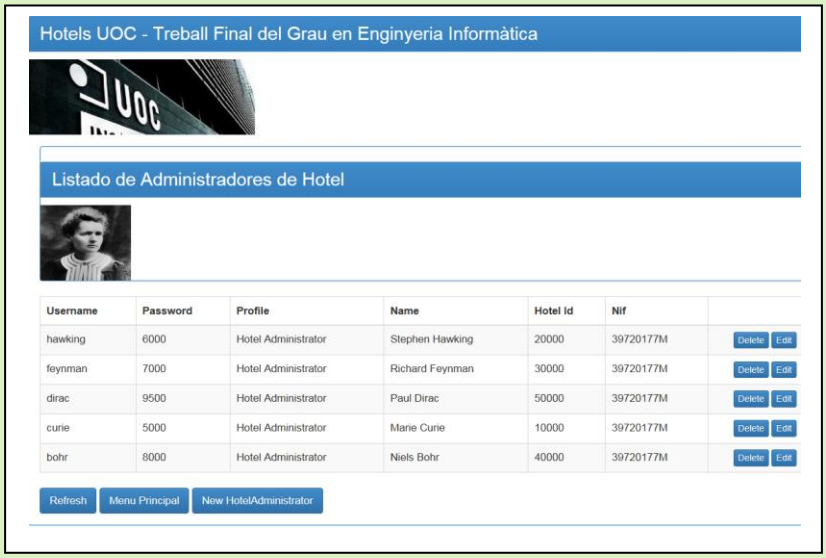

**[SACreateHotelAdminView.html]**




Para confirmar la creación de un nuevo administrador de hotel se pulsa el botón Save (botón Cancel para abortar la acción y retornar al listado de administradores de hotel).

Aspecto de la lista de administradores de hotel tras crear el **HotelAdministrator** de *usrename* = 'dirac'

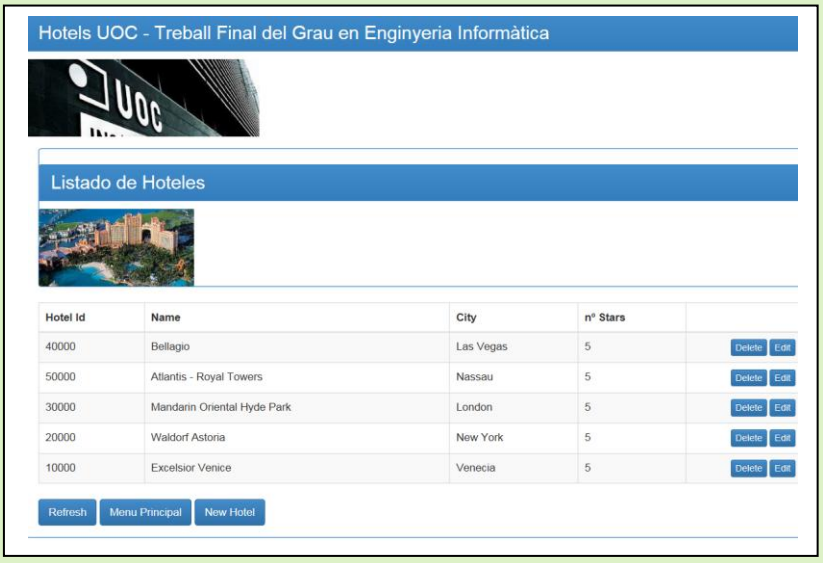
**[SATableHotelAdminView.html]**




- **Hotel Management**

Al pulsar el botón *Hotel Management* se despliega una lista de todos los hoteles almacenados en el sistema.

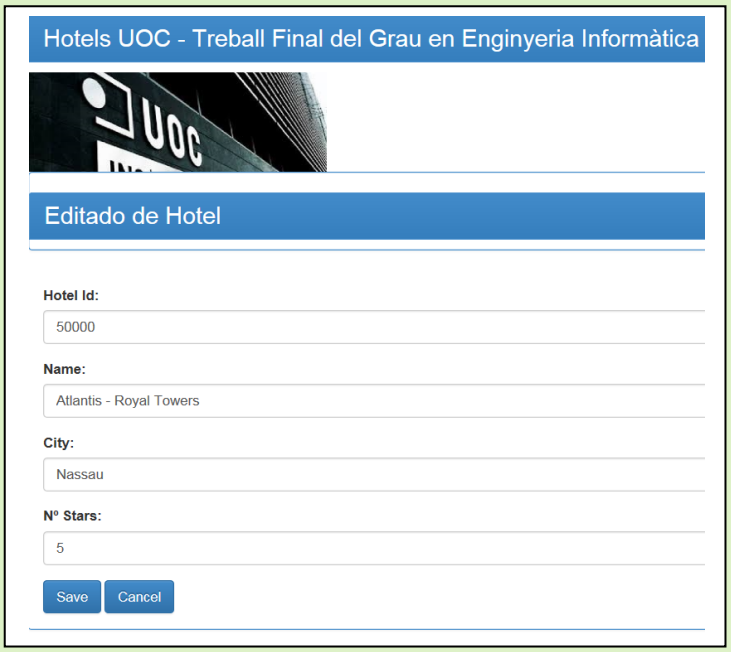
**[SATableHotelView.html]**




Des de esta vista del listado de hoteles del sistema es posible:

1. Editar cualquiera de los hoteles (botón *Edit* en la fila del hotel que deseemos editar)

**[SAEditorHotelView.html]**




Una vez editado el hotel podemos realizar en él cuantas modificaciones queramos y salvarlas pulsando el botón *Save* (botón *Cancel* para abortar la acción y retornar al listado de hoteles).

2. Eliminar cualquiera de los hoteles (botón *Delete* en la fila del hotel que deseemos eliminar)

Aspecto de las lista de hoteles tras eliminar el hotel con hotelId = '40000' :

**[SATableHotelView.html]**


3. Crear un nuevo hotel (botón *New Hotel )*

Pulsando el botón *New Hotel,* la aplicación nos permite dar de alta un nuevo hotel. Para ello muestra una vista con todos los campos que constituyen un nuevo hotel en blanco, para que sea el SuperAdministrator quien los rellene con los datos que desee.

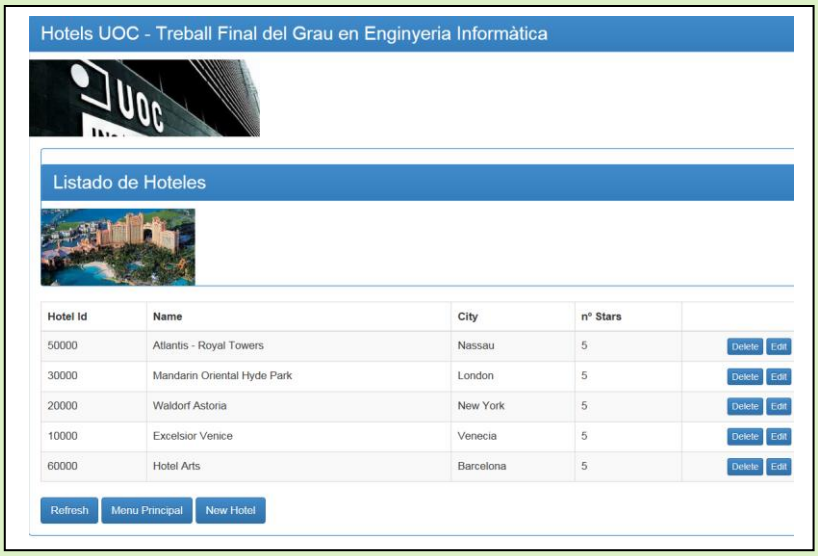
**[SACreateHotelView.html]**


**[SACreateHotelView.html]**


Para confirmar la creación de un nuevo hotel se pulsa el botón Save (botón Cancel para abortar la acción y retornar al listado de hoteles).

Aspecto de la lista de clientes tras crear el cliente de usrename = user2

**[SATableHotelView.html]**



### *Perfil HotelAdministrator (HA)*

El perfil *HotelAdministrator* asume la responsabilidad de gestionar las ofertas, las órdenes de compra y los comentarios referidos al hotel que administra.

Una vez realizado exitosamente el login en el sistema, la aplicación muestra la vista correspondiente al *home* HA:

**[HAHomeView.html]**

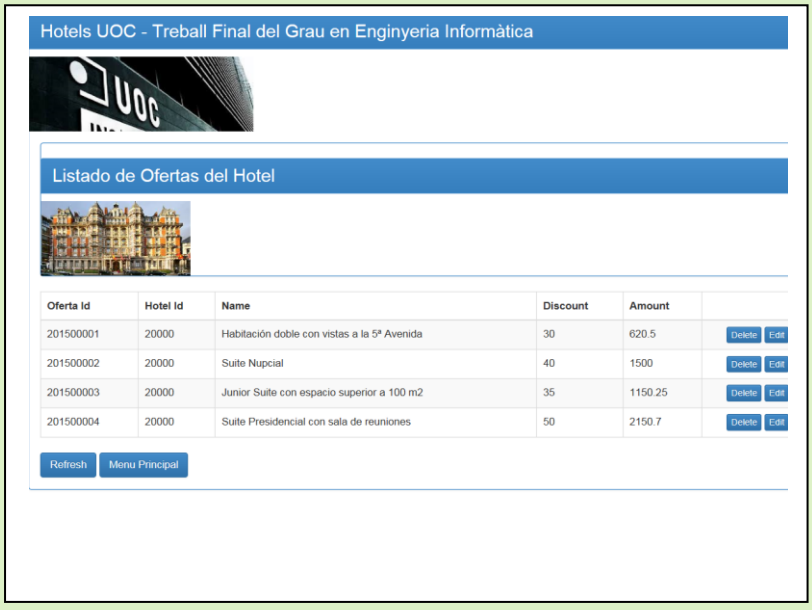

Esta es la descripción de las funcionalidades que desarrolla el perfil *HotelAdministrator*:

- **Ofertas Publicadas**

Pulsando el botón *Ofertas Publicadas* del *home* del perfil *HA*, se muestra una vista con todas las ofertas publicadas del hotel administrado por el perfil *HotelAdministrator.* '

Este *HotelAdministrator* administra el hotel correspondiente al *hotelId=20000.*

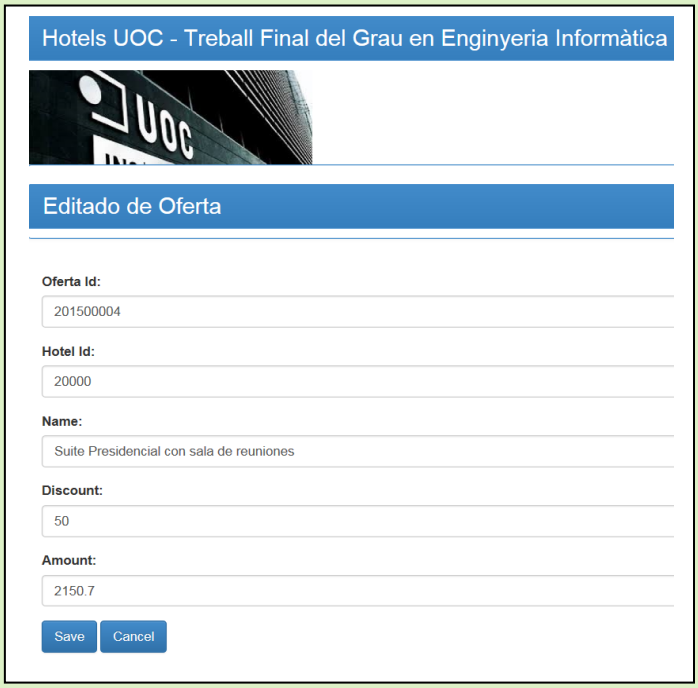
**[HATableOfertaView.html]**



Des de esta vista del listado de ofertas del hotel es posible:

1. Editar cualquiera de las ofertas (botón *Edit* en la fila de la oferta que deseemos editar)

**[ HAEditorOfertaView.html]**




Una vez editada la oferta es posible realizar en ella cuantas modificaciones queramos y salvarlas pulsando el botón *Save* (botón *Cancel* para abortar la acción y retornar al listado de ofertas).

2. Eliminar cualquiera de las ofertas (botón *Delete* en la fila de la oferta que deseemos editar)

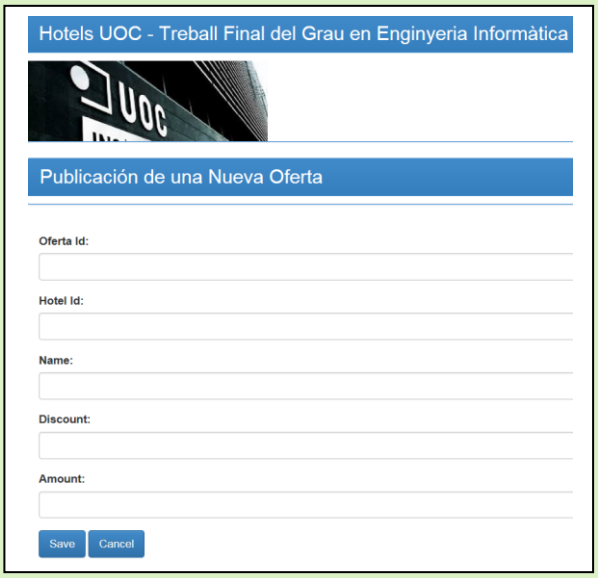
Aspecto de las lista de ofertas tras eliminar la oferta con ofertaId= '201500002' :

**[HATableOfertaView.html]**

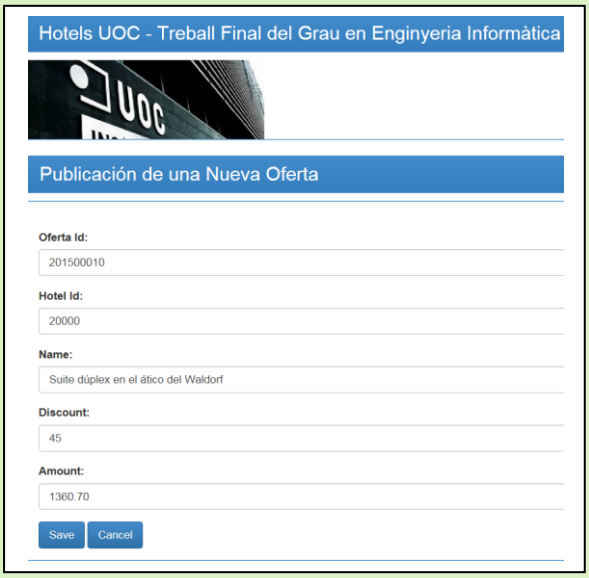
- **Publicar Oferta**

Pulsando el botón *Publicar Oferta* en el *home* del perfil *HA,* la aplicación nos permite dar de alta una nueva oferta. Para ello muestra una vista con todos los campos que constituyen una oferta en blanco, para que sea el *HotelAdministrator* del hotel quien los rellene con los datos de la oferta que desea publicar.

**[HACreateOfertaView.html]**




**[HACreateOfertaView.html]**




Para confirmar la creación de una nueva oferta se pulsa el botón Save (botón Cancel para abortar la acción y retornar al listado de ofertas del hotel).

Aspecto de la lista de ofertas tras confirmar la creación de la oferta correspondiente a *ofertaId*= '201500010 ' :

**[HATableOfertaView.html]**

- **Registro de Compras**

Pulsando el botón *Registro de Compras* en el home del perfil HA, la aplicación muestra un listado de todas las compras correspondientes a las ofertas del hotel que los clientes han comprado.

**[HATableOrderView.html]**


Des de esta vista del listado de compras del hotel es posible:

1. Editar cualquiera de las compras (botón *Edit* en la fila de la compra que deseemos editar)

**[HAEditorOrderView.html]**


Una vez editada una compra ***no*** se ofrece la posibilidad de ser modificada (botón *Cancel* para retornar al listado de compras).

2. Eliminar cualquiera de las compras (botón *Delete* en la fila de la compra que deseemos eliminar)

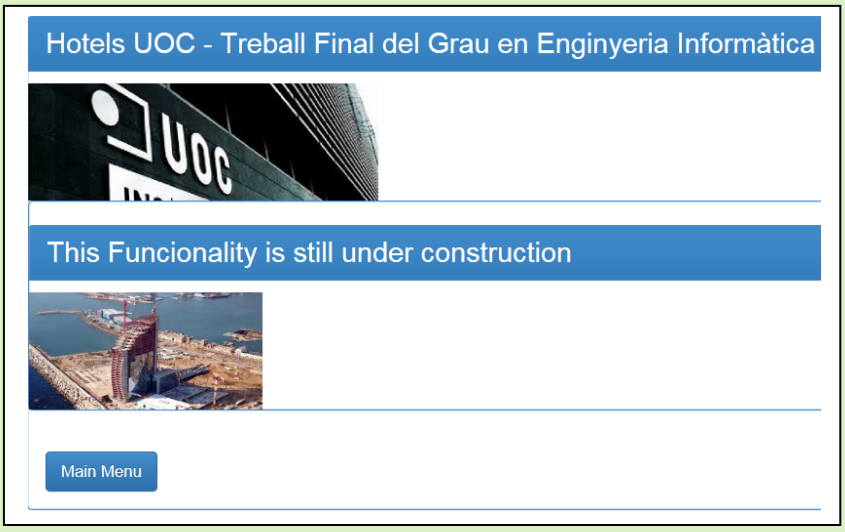
Aspecto de la lista de compras tras eliminar la compra con *orderId*= '20150000320000' :
**[HATableOrderView.html]**


Se trata de un borrado lógico. La compra ya no aparecerá más en la lista del administrador del hotel, pero no será borrada del sistema y sí seguirá apareciendo en la lista de compras realizadas por el cliente hasta que él mismo decida eliminarla de su lista de compras.

- **Registro de Clientes**

Esta es una funcionalidad que en una primera ampliación de la aplicación sería interesante implementar. Es importante conocer qué clientes nos compran, que consumen, y con qué frecuencia y cantidad lo hacen para poner en marcha mecanismos de fidelización de clientes y marketing personalizado. Al final de la memoria se propone una posible primera ampliación de la aplicación.

**[ underConstructionView.html]**



- **Comentarios Hotel**

Pulsando el botón *Comentarios Hotel* en el *home* del perfil *HA*, la aplicación muestra un listado de todos los comentarios que los clientes han realizado sobre el hotel y que el administrador del hotel no ha borrado.

**[HATableComentarioView.html]**


Des esta vista del listado de comentarios sobre el hotel es posible:

1. Editar cualquiera de los comentarios (botón *Edit* en la fila del comentario que deseemos editar)

**[HAEditorComentarioView.html]**


Una vez editado un comentario ***no*** se ofrece la posibilidad de ser modificado (botón Cancel para retornar al listado de comentarios).

2. Eliminar cualquiera de los comentarios (botón *Delete* en la fila del comentario que deseemos eliminar)

Aspecto de la lista de comentarios tras eliminar el comentario con *comentarioId*= '300001' :

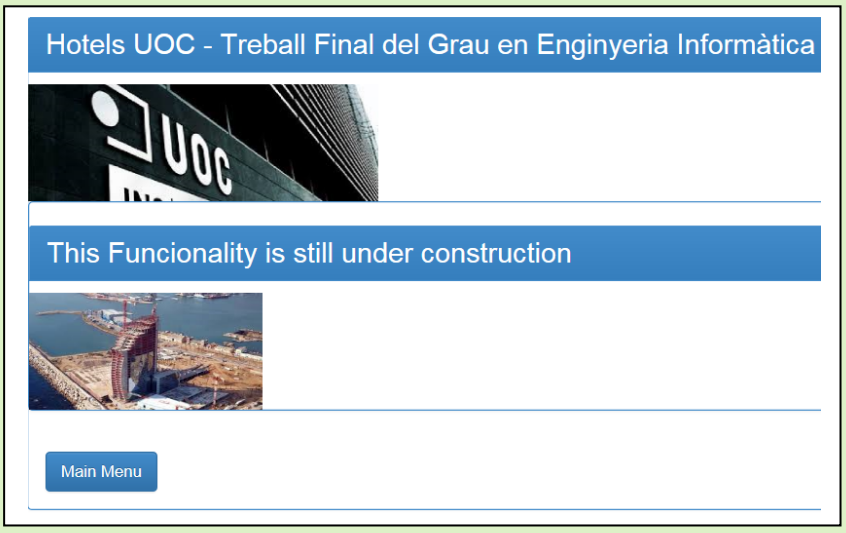
**[HATableComentarioView.html]**


Se trata de un borrado lógico. El comentario ya no aparecerá más en la lista de comentarios del hotel, y ningún cliente podrá ya ver ese comentario asociado al hotel, pero no será borrado del sistema y sí seguirá apareciendo en la lista de comentarios realizados del cliente que lo creó.

- **Perfil Hotel**

  Sobre esta funcionalidad comentar aspectos similares a los ya comentados en la funcionalidad Registro de Clientes. Esta funcionalidad Perfil Hotel, cobrará importancia cuando en una futura ampliación se enriquezca gráficamente la aplicación, se permita a los administradores de hotel crear perfiles complejos que contengan fotos, especialidades del hotel, factores diferenciales, etc.

**[ underConstructionView.html]**



### *Perfil Customer (CUST)*

El perfil *Customer* puede buscar hoteles y ofertas en la aplicación, comprar cuántas ofertas desee, ver los comentarios que otros clientes han hecho sobre cierto hotel o realizar comentarios sobre los hoteles en los que se ha alojado. También puede construir su propia lista de hoteles favoritos, que puede ir modificando tal y como lo desee.

Una vez realizado exitosamente el login en el sistema, la aplicación muestra la vista correspondiente al *home* del perfil *CUST*:

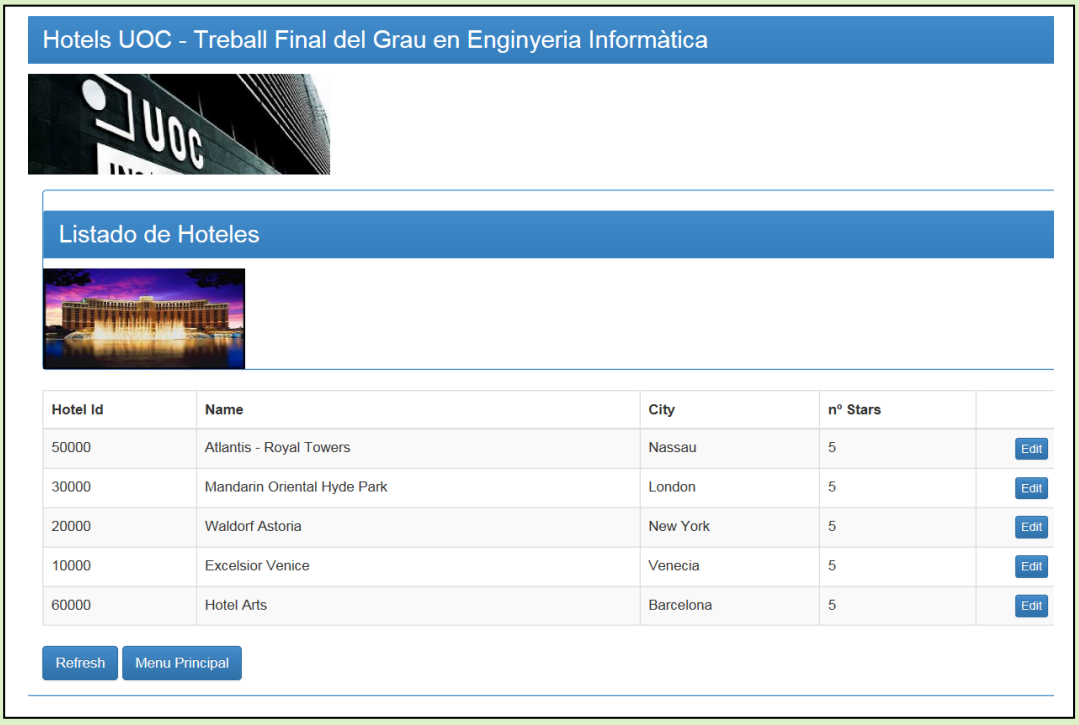
**[CUSTHomeView.html]**


Esta es la descripción de las funcionalidades que desarrolla el perfil *Customer* :

- **Buscar Hoteles**

Pulsando el botón *Buscar Hoteles* en el *home* del perfil *CUST*, se muestra una vista con todos los hoteles registrados en el sistema.

**[CUSTTableHotelView.html]**



Des de esta vista del listado de hoteles es posible editar cualquiera de los hoteles (botón *Edit* en la fila del hotel que deseemos editar).

**[CUSTEditorHotelView.html]**

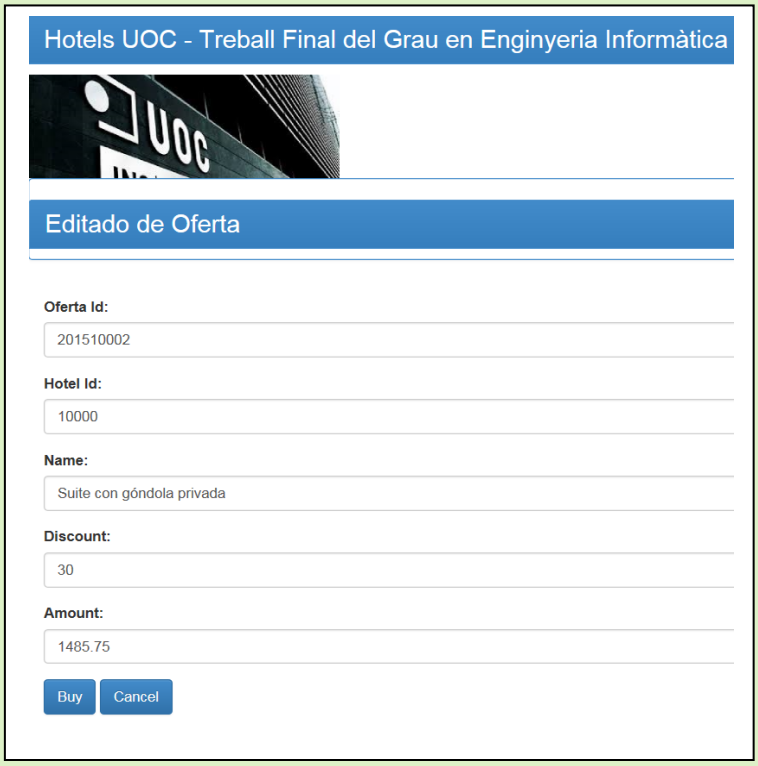

Des de esta vista del editado de un hotel es posible:

1. Mostrar un listado de todas las ofertas que el hotel tiene registradas en el sistema.

**[CUSTTableOfertaView.html]**


Desde el listado de las ofertas del hotel editado podemos editar cualquiera de ellas (botón Edit en la fila de la oferta que deseemos editar)

**[CUSTEditorOfertaView.html]**




Y una vez editada una de las ofertas del listado de ofertas del hotel previamente editado, la aplicación permite proceder a su compra (se explica esta funcionalidad en el siguiente apartado, **Buscar Ofertas**)

2. Añadir el hotel editado a la lista de favoritos del cliente (botón Add Favoritos en la vista de editado del hotel.

**[CUSTTableFavoritoView.html]**

3. Mostrar un listado de todos los comentarios realizados sobre el hotel no eliminados por el administrador del hotel (botón Comments en la vista del listado del hotel).

   Tras retornar al menú principal y pulsar el botón Listar Hoteles Favoritos, comprobamos que el hotel de *hotelId*='10000' ha pasado a formar parte de la lista de hoteles favoritos del cliente.

**[CUSTHATableComentarioView.html]**

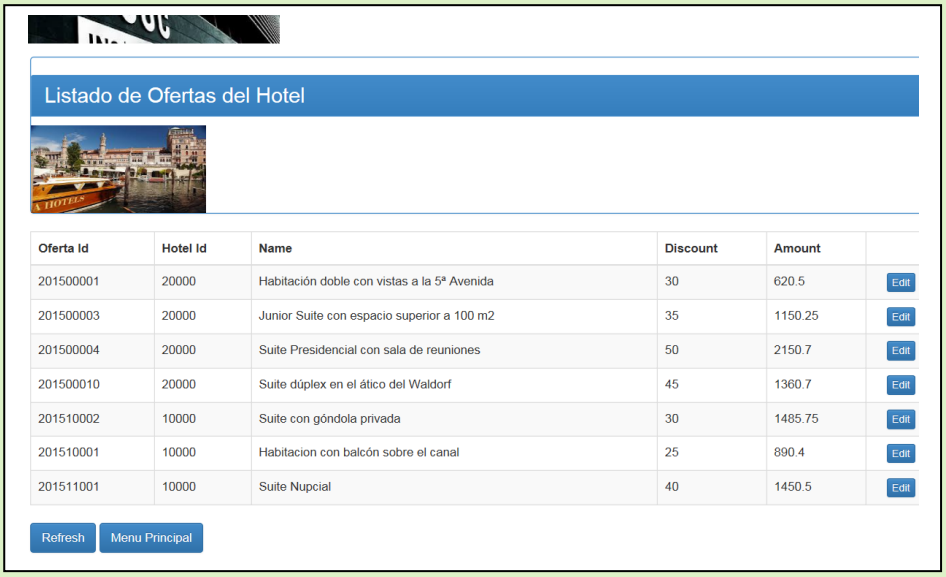

Des de esta vista del listado de los comentarios pertenecientes al hotel previamente editado, se puede editar cualquiera de los comentarios de la lista (botón Edit en la fila del comentario que deseemos editar).

**[CUSTHAEditorComentarioView.html]**

- **Buscar Ofertas**

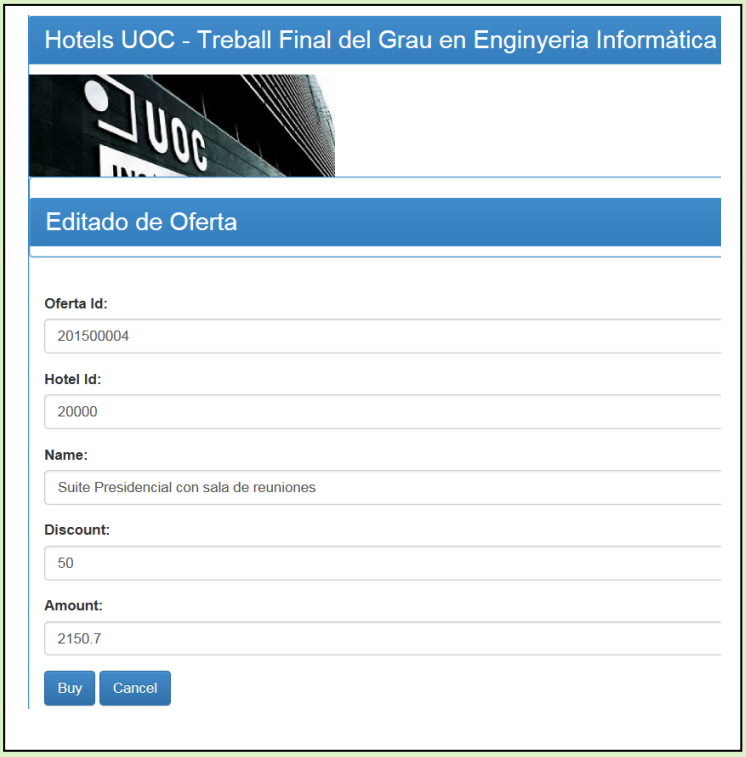
Pulsando el botón *Buscar Ofertas* en el *home* del perfil *CUST*, se muestra una lista con todas las ofertas publicadas en el sistema por cualquiera de los hoteles que en él están registrados.

**[CUSTTableOfertaView.html]**




Desde la vista del listado de todas las ofertas publicadas en el sistema puede editarse cualquiera de ellas (botón Edit en la fila de la oferta que deseemos editar).

**[CUSTEditorOfertaView.html]**



Y tal y como se ha comentado en el apartado anterior, una vez publicada un oferta , la aplicación permite proceder a su compra (botón Buy en la vista de editado de la oferta).

Tras retornar al menú principal y pulsar el botón Registro de Compras, comprobamos que la compra realizada (*orderId*= '20150000420000') ya pude ser visualizada en la lista de compras del cliente.

**[CUSTTableOrderView.html]**

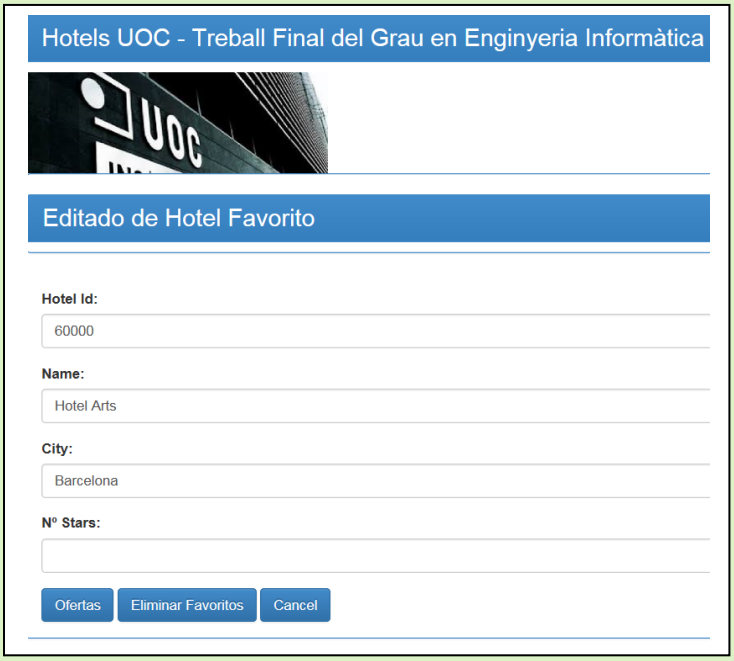

- **Listar Hoteles Favoritos**

Pulsando el botón Listar Hoteles Favoritos en el *home* del perfil *CUST*, se muestra una lista con todos los hoteles que el cliente ha marcado como favoritos:

**[CUSTTableFavoritoView.html]**

Desde esta lista de hoteles favoritos se puede editar cualquiera de los hoteles.

**[CUSTEditorFavoritoView.html]**




Y a su vez, des de la vista de edición de un hotel marcado como favorito se puede:

1. Eliminar el hotel de la lista de favoritos (botón Eliminar Favoritos).

Pulsamos el botón Eliminar Favoritos, retornamos al menú principal y pulsamos el botón Listar Hoteles Favoritos en el *home* del perfil *CUST para comprobar* que el hotel ha sido eliminado de la lista de hoteles favoritos y ya no aparece en ella.

**[CUSTTableFavoritoView.html]**

2. Mostrar un listado de las ofertas que el hotel marcado como favorito tiene publicadas en el sistema.

Des de la vista correspondiente a este listado se puede editar cualquiera de las ofertas, y una vez editada, proceder a su compra siguiendo el proceso expuesto en el apartado **Buscar Ofertas**.

- **Registro de Compras**

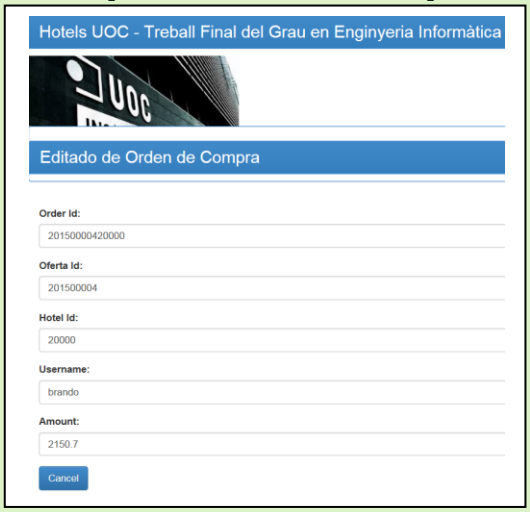
Pulsando el botón *Registro de Compras* en el *home* del perfil *CUST*, se muestra una lista con todas las órdenes de compra realizadas por el cliente que no han sido eliminadas de la lista por éste.

**[CUSTTableOrderView.html]**


Des de esta vista del listado de compras realizadas por el cliente es posible:

1. Editar cualquiera de las compras (botón *Edit* en la fila de la compra que deseemos editar).

**[CUSTEditorOrderView.html]**



Una vez editada una compra ***no*** se ofrece la posibilidad de ser modificada (botón *Cancel* para retornar al listado de compras).

2. Eliminar cualquiera de las compras (botón *Delete* en la fila de la compra que deseemos eliminar)

Aspecto de la lista de compras tras eliminar la compra con *orderId*= '20150000420000' :

**[CUSTTableOrderView.html]**


Se trata de un borrado lógico. La compra ya no aparecerá más en la lista de compras del cliente, pero no será borrada del sistema y sí seguirá apareciendo en la lista del registro de compras del hotel al cual pertenecía dicha compra.

- **Comentarios Realizados**

Pulsando el botón *Comentarios Realizados* en el *home* del perfil *CUST*, se muestra una lista con todos los comentarios realizados por el cliente que no han sido eliminados de la lista por éste.

**[CUSTTableComentarioView.html]**

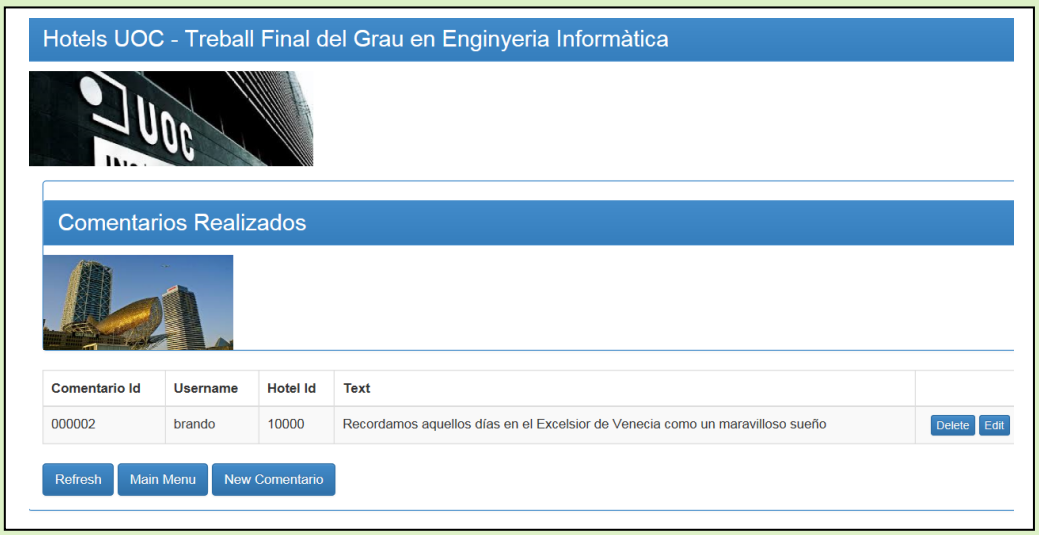
Des de la vista del listado de comentarios realizados por el cliente es posible:

1. Editar cualquiera de los comentarios (botón *Edit* en la fila del comentario que deseemos editar).

**[CUSTEditorComentarioView.html]**


2. Eliminar cualquiera de los comentarios (botón *Delete* en la fila del comentario que deseemos eliminar).

**[CUSTTableComentarioView.html]**

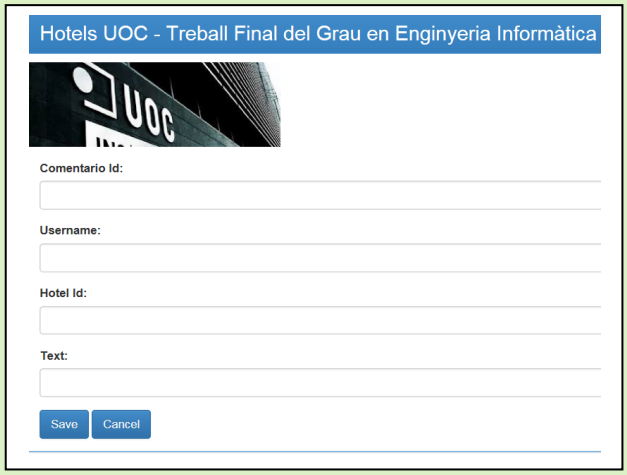



Se trata de un borrado lógico. El comentario ya no aparecerá más en la lista de comentarios realizados por el cliente, pero no será borrado del sistema y sí seguirá apareciendo en la lista de comentarios asociados al hotel si el administrado del hotel no lo ha eliminado, pudiendo el resto de clientes seguir viendo el comentario asociado al hotel.

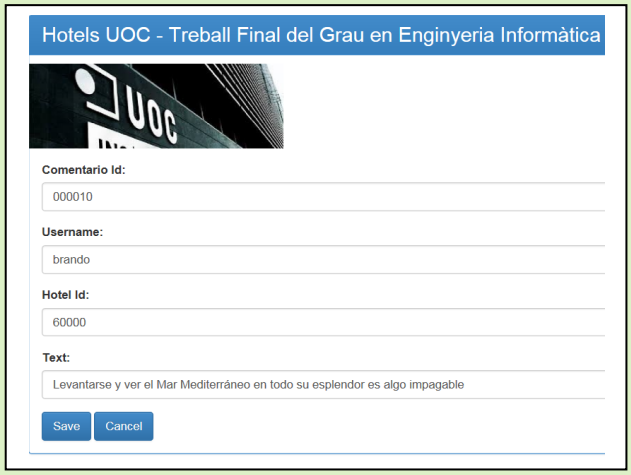
3. Crear un nuevo comentario asociándolo al hotel que se desee.

Pulsando el botón *New Comentario* , la aplicación nos permite dar de alta un nuevo comentario. Para ello muestra una vista con todos los campos que constituyen un comentario en blanco, para que sea el administrador del hotel quien los rellene con los datos del comentario que desea publicar.

**[CUSTCreateComentarioView.html]**



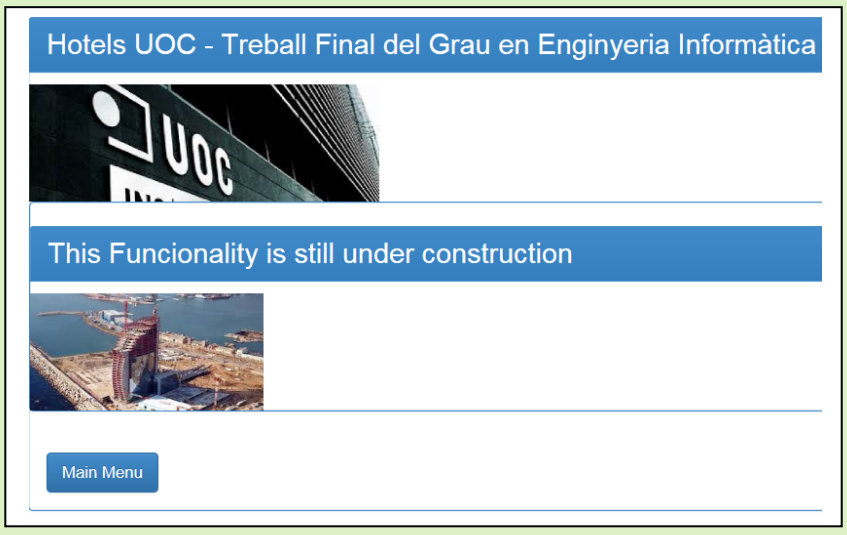

**[CUSTCreateComentarioView.html]**




- **Perfil Personal**

Sobre esta funcionalidad comentar aspectos similares a los ya comentados en la funcionalidad Perfil Hotel. Esta funcionalidad Perfil Personal cobrará importancia cuando en una futura ampliación se enriquezca gráficamente la aplicación y se permita a los clientes crear perfiles complejos que contengan fotos, preferencias, etc.

**[underConstructionView.html]**



# Curva de aprendizaje

La curva de aprendizaje ha sido una de las tareas en la que más tiempo se ha invertido por ser clave en la consecución de los objetivos del proyecto. Se ha tratado un amplio abanico de bibliografía para abordar tres temas principales: AngularJs Framework, Spring Framework y Hibernate Framework. Esta sección está destinada a exponer de manera clara y sencilla los principios básicos en los que se fundamentan cada uno de estos tres marcos de trabajo.


En este apartado se pretende dar una idea de la estructura y de las principales características que una aplicación desarrollada mediante AngularJS presenta. Los componentes fundamentales de la aplicación son los módulos, servicios, controladores, directivas y vistas. Se intentará analizar conceptualmente y contextualizar cada uno de estos elementos, utilizando ejemplos procedentes del código desarrollado durante el proyecto(*).

**Módulos**

Los módulos son los componentes de nivel superior en las aplicaciones AngularJS y realizan tres funciones principales en ellas:

- Asociar una aplicación AngularJS con una región concreta de un documento HTML.
- Actuar como puerta de entrada a características clave del Framework AngularJS.
- Ayudar a organizar el código en una aplicación AngularJS.

La primera línea de código del Anexo 1 refleja la creación de un módulo:

```
angular.module("exampleApp", ["ngResource", "ngRoute"])
```

El primero de los parámetros se refiere al nombre que se asigna al módulo, mientras que el segundo de ellos es una lista de las dependencias del mismo. Se hablará de los módulos *ngResource* y *ngRoute* a lo largo de este apartado. El método *module* admite un tercer parámetro de configuración equivalente a invocar el método *config* sobre el módulo (se habla de este método seguidamente).

*(*) Se incluyen dos anexos en este apartado que serán referenciados con frecuencia en esta sección:*

*El **Anexo 1** muestra el código cliente desarrollado en AngularJS, sin incluir por motivos de espacio, las porciones de código de cada uno de los controladores de los que únicamente se muestra la signatura.*

*El **Anexo 2** muestra en miniatura todas las vistas de la aplicación.*

Sobre un objeto *module* se definen una docena de métodos de los cuales se destacan cuatro, todo ellos utilizados en el cliente Angular del proyecto:

- **`config(callback)`**
  Registra una función que será usada para configurar el módulo en el momento en que sea cargado.

- **`factory(name, provider)`**
  Crea un servicio.

- **`controller(name, constructor)`**
  Crea un controlador.

- **`directive(name, factory)`**
  Crea una directiva que extiende el vocabulario HTML estándar.

### Servicios

Se utilizan para encapsular funcionalidades que serán reutilizadas en otros componentes de la aplicación y que no encajan de manera clara en el modelo MVC, se trata más bien de funcionalidades transversales cuyo ámbito de influencia va más allá de un único componente.

El propio AngularJS define una serie de servicios que pone a disposición de los desarrolladores para abordar los problemas más paradigmáticos en el desarrollo web. Algunos ejemplos son el servicio *$http* que crea y gestiona peticiones Ajax, el servicio *$resource* que proporciona soporte para trabajar con servicios web REST y los servicios *$rootScope*, *$route* y *$routeParams* que también serán explicados en esta sección.

### Controladores

Los controladores actúan como enlace entre el modelo y las vistas, proporcionando datos y métodos a las vistas y definiendo la lógica de negocio necesaria para convertir las acciones del usuario en cambios en el modelo .

Los controladores son creados por mediación del método *controller* mencionado anteriormente y solicitan el servicio *$scope* que es usado para proporcionar a la vista su propio ámbito, definiendo los datos y la lógica de negocio que la vista puede utilizar. Aunque habitualmente es tratado como tal, estrictamente hablando *$scope* no es un servicio sino un objeto proporcionado por el servicio *$rootscope*. Los *$scope* están organizados jerárquicamente, a cada nuevo controlador se le asigna un objeto *$scope* que es hijo del *$scope* raíz. Esta organización jerárquica permite la comunicación entre distintos controladores a través de sus respectivos *$scope*.

### Directivas

Las directivas constituyen una muy importante característica de AngularJs, pues permiten la extensión del lenguaje HTML contribuyendo a la construcción de complejas aplicaciones web que proporcionan una rica experiencia al usuario. En un primera clasificación podríamos dividir las directivas en "data binding directives" y "template directives" (se puede traducir como *directivas de enlazado de datos* y *directivas de plantilla)*. Del primer tipo destaca la directiva *ng-model* y del segundo las directivas *ng-include* y *ng-repeat*. Se verán ejemplos de su uso en las vistas parciales del cliente AngularJS implementado.

### Vistas

En este punto cabe remarcar que las aplicaciones web generadas con AngularJS son de página única, es decir, en realidad existe un único documento *html* que es con el que el usuario interactúa. Aun así es posible utilizar la denominada técnica de vistas parciales. La directiva *ng_include* mencionada anteriormente, recupera el fragmento de código HTML que le indicamos del servidor, lo compila para procesar las directivas que pueda contener, y lo añade al documento *html* principal. Cada uno de estos fragmentos es lo que conocemos como vistas parciales. Analizaremos todas las vistas parciales del cliente AngularJS implementado (todas ellas representadas en el Anexo 2) y veremos cómo podemos vincularlas con los controladores y URLs que deseemos utilizando el servicio *$route*.

Una vez hecha una primera aproximación a los componentes que constituyen una aplicación AngularJs, pasaremos a analizar cómo han sido utilizados en el cliente AngularJs implementado.

## Análisis del Cliente AngularJS implementado

En el anexo 1 podemos observar que el código del cliente Angular implementado se estructura en cinco grandes bloques:

- Definición del módulo: Se define un único módulo que presenta dependencias con los módulos *ngResource* y *ngRoute*.

- Definición de los valores constantes: Se definen siete valores constantes (*baseUrlHotel, baseUrlOferta, baseUrlCustomer, baseUrlHotelAdmin, baseUrlOrder, baseUrlComentario, baseUrlFavorito).*

- Definición de los servicios: Se definen siete servicios (*hotelsResource*, *ofertasResource*, *customersResource*, *hotelAdminsResource*, *ordersResource*, *comentariosResource*, *favoritosResource*). Cada uno de estos servicios acaba realizando una llamada al servicio $resource propio de AngularJS e incluido en el módulo opcional *ngResource*. Este hecho genera la necesidad de explicar en qué consiste este servicio.

- Definición de las rutas: Se aplica el método *config* al objeto *module* que registra una función que será usada para configurar el módulo en el momento en que sea cargado. Esta función presenta dependencias con los servicios $routeProvider y $locationProvider, así pues, será necesario explicar los servicios $route y $location para entender cómo se definen las rutas, como se asignan los controladores a las vistas y como navegamos de una a otra en la aplicación.

Antes de profundizar en el análisis se añaden a continuación una selección de elementos del cliente Angular que se utilizarán como ejemplo para mejorar el entendimiento de todo lo que se pretende explicar (esta selección de elementos será

referenciada como "*subconjunto-ejemplo del cliente AngularJS*"). Este es el inventario de los componentes que constituyen el *subconjunto-ejemplo del cliente AngularJS*:

- Definición del módulo *exampleApp*
- Definición de la constante *baseUrlOferta*
- Definición del servicio ofertas*Resource*
- Definición de la ruta "*/homeCustomer* "
- Definición de la ruta "*/listOferta*"
- Definición de la ruta "*/CUSTEditOferta/:id*"
- Definición del controlador *homeCustomerCtrl*
- Definición del controlador *tableOfertaHotelCtrl*
- Definición del controlador *editOfertaCtrl*

- *hotelsUOC.html*
  Código html del documento base en el cual se incluyen todas las vistas parciales.

- *homeCustomerView.html (source)*
  Código *html* de la vista inicial del perfil *customer*.

- *homeCustomerView.html (view)*
  Aspecto que muestra en el navegador la vista inicial del perfil *customer*.

- *CUSTTableOfertaView.html (source)*
  Código *html* de la vista del listado de ofertas del perfil *customer*.

- *CUSTTableOfertaView.html (view)*
  Aspecto que muestra en el navegador la vista del listado de ofertas del perfil *customer.*

- *CUSTEditorOfertaView.html (source)*
  Código *html* de la vista de edición de una oferta del perfil *customer*.

- *CUSTEditorOfertaView.html (view)*
  Aspecto que muestra en el navegador la vista de edición de una oferta del perfil *customer.*

```
angular.module("exampleApp"     , ["ngResource", "ngRoute"])
.constant("baseUrlOferta"                                                    ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/ofertas/")
.factory("ofertasResource"      , function ($resource, baseUrlOferta) {
    return $resource(baseUrlOferta + ":id", { id: "@ofertaId" },
           {getofertashotel: {method: "GET", params: {hotelId: "@hotelId"}, url:
baseUrlOferta + "hotel/" + ":hotelId", isArray:true}} );
})
.config(function ($routeProvider, $locationProvider) {

        $routeProvider.when("/homeCustomer", {
            templateUrl: "/CUSTHomeView.html",
            controller: "homeCustomerCtrl"
        });
        $routeProvider.when("/listOferta", {
```

```
            templateUrl: "/CUSTTableOfertaView.html",
            controller: "tableOfertaCtrl",
            resolve: {
                dataOferta: function (ofertasResource) {
                    return ofertasResource.query();
                }
            }
        });

        $routeProvider.when("/CUSTEditOferta/:id", {
            templateUrl: "/CUSTEditorOfertaView.html",
            controller: "editOfertaCtrl"
        });

})
.controller("homeCustomerCtrl", function ($scope, $location) {

        $scope.hoteles = function () {
            $location.path("/CUSTListHotel");
        };
        $scope.ofertas = function () {
            $location.path("/listOferta");
        };
        $scope.favoritos = function () {
            $location.path("/CUSTListFavorito");
        };
        $scope.compras = function () {
            $location.path("/listOrderCustomer");
        };
        $scope.comentarios = function () {
            $location.path("/listComentarioCustomer");
        };
        $scope.perfil = function () {
            $location.path("/CUSTUnderConstruction");
        }
})
.controller("tableOfertaHotelCtrl", function ($scope, $location, $route,
ofertasResource) {

        if ($location.path().indexOf("/HAListOfertaHotel") == 0) {
            $scope.data.ofertas =
ofertasResource.getofertashotel($scope.data.CurrentHotelAdmin);
        }

        $scope.deleteOferta = function (oferta) {
            oferta.$delete().then(function () {
                $scope.data.ofertas.splice($scope.data.ofertas.indexOf(oferta), 1);
            });
            $route.reload();
        };

        $scope.refreshOfertas = function () {
            $route.reload();
        }
})
```

```
.controller("editOfertaCtrl", function ($scope, $routeParams, $location,
ofertasResource, ordersResource) {

        $scope.currentOferta = null;

        if ($location.path().indexOf("/CUSTEditOferta/") == 0 ||
$location.path().indexOf("/HAEditOferta/") == 0){
            var id = $routeParams["id"];
            for (var i = 0; i < $scope.data.ofertas.length; i++) {
                if ($scope.data.ofertas[i].ofertaId == id) {
                    $scope.currentOferta = $scope.data.ofertas[i];
                    break;
                }
            }
        }

        $scope.createOrder = function () {

            function order(ofertaId,hotelId, usuario, importe) {
                this.orderId = ofertaId + hotelId;
                this.ofertaId = ofertaId;
                this.hotelId = hotelId;
                this.username = usuario;
                this.amount = importe;
                this.deleteByHotel=false;
                this.deleteByCustomer=false;
            }
            var order = new
order($scope.currentOferta.ofertaId,$scope.currentOferta.hotelId,$scope.data.username
,$scope.currentOferta.amount);

            ordersResource.save(order);
            $location.path("/listOrderCustomer");
        };

        $scope.createOferta = function (oferta) {
            ofertasResource.save(oferta);
            $location.path("/homeHotelAdmin");
        };

        $scope.updateOferta = function (oferta) {
            ofertasResource.save(oferta);
            $location.path("/HAListOfertaHotel");
        };

        $scope.saveEditCreateOferta = function (oferta) {
            $scope.createOferta(oferta);
            $scope.currentOferta = {};
        };

        $scope.saveEditOferta = function (oferta) {
            $scope.updateOferta(oferta);
            $scope.currentOferta = {};
        };

        $scope.cancelEditCreateOferta = function () {
```

```
            $location.path("/homeHotelAdmin");
        };

        $scope.cancelEditOferta = function () {
            if ($location.path().indexOf("/CUSTEditOferta/") == 0){
                $location.path("/listOferta");
            }

            if ($location.path().indexOf("/HAEditOferta/") == 0){
                $location.path("/HAListOfertaHotel");
            }
        };
})
```

***hotelsUOC.html***

```
<!DOCTYPE html>
<html ng-app="exampleApp">
<head>
    <title>Hotels UOC - Treball Final del Grau en Enginyeria
Informàtica</title>
    <script src="angular.js"></script>
    <script src="angular-resource.js"></script>
    <script src="angular-route.js"></script>
    <link href="bootstrap.css" rel="stylesheet" />
    <link href="bootstrap-theme.css" rel="stylesheet" />
    <script src="hotelsUOC.js"></script>
</head>
<body ng-controller="defaultCtrl">
    <div class="panel panel-primary">
        <h3 class="panel-heading">Hotels UOC - Treball Final del Grau en
Enginyeria Informàtica</h3>
        <img src="images/uoc.jpg" style="width:300px;height:100px">
        <div ng-view></div>
    </div>
</body>
</html>
```

***homeCustomerView.html (source)***

```
<div class="panel panel-primary">
    <h3 class="panel-heading">CUSTOMER</h3>
</div>
<div class="panel-body">
    <div>
        <button class="btn btn-primary" ng-click="hoteles()">Buscar
Hoteles</button>
        <button class="btn btn-primary" ng-click="ofertas()">Buscar
Ofertas</button>
        <button class="btn btn-primary" ng-click="favoritos()">Listar Hoteles
Favoritos</button>
        <button class="btn btn-primary" ng-click="compras()">Registro de
Compras</button>
        <button class="btn btn-primary" ng-click="comentarios()">Comentarios
Realizados</button>
        <button class="btn btn-primary" ng-click="perfil()">Perfil
Personal</button>
    </div>
</div>
```

***homeCustomerView.html (view)***


***CUSTTableOfertaView.html (source)***

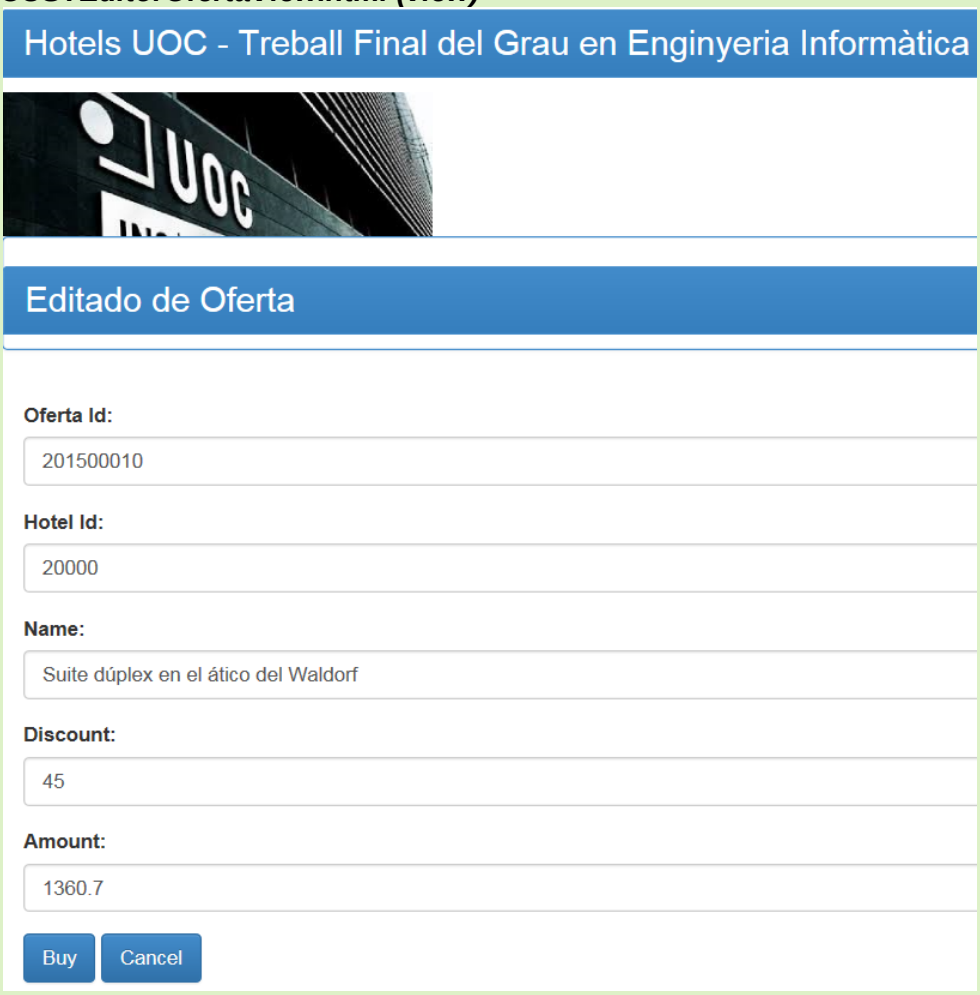
```
<div class="panel-body">
    <table class="table table-striped table-bordered">
        <thead>
            <tr>
               <th>Oferta Id</th>
               <th>Hotel Id</th>
                <th>Name</th>
                <th>Discount</th>
                <th>Amount</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
        <div class="panel panel-primary">
            <h3 class="panel-heading">Listado de Ofertas del Hotel</h3>
            <img src="images/excelsior.jpg" style="width:200px;height:100px">
        </div>
            <tr ng-repeat="item in data.ofertas">
                <td>{{item.ofertaId}}</td>
                <td>{{item.hotelId}}</td>
                <td>{{item.name}}</td>
                     <td>{{item.discount}}</td>
                <td>{{item.amount}}</td>
                <td class="text-center">
                    <a href="/CUSTEditOferta/{{item.ofertaId}}" class="btn
btn-xs btn-primary">Edit</a>
                </td>
            </tr>
        </tbody>
    </table>
    <div>
        <button class="btn btn-primary" ng-
click="refreshOfertas()">Refresh</button>
        <a href="/homeCustomer" class="btn btn-primary">Menu Principal</a>
    </div>
</div>
```

***CUSTTableOfertaView.html (view)***


***CUSTEditorOfertaView.html (source)***

```
<div class="panel panel-primary">
    <h3 class="panel-heading">Editado de Oferta</h3>
</div>
<div class="panel-body">
    <div class="form-group">
        <label>Oferta Id:</label>
        <input class="form-control" ng-model="currentOferta.ofertaId" />
    </div>
    <div class="form-group">
        <label>Hotel Id:</label>
        <input class="form-control" ng-model="currentOferta.hotelId" />
    </div>
    <div class="form-group">
        <label>Name:</label>
        <input class="form-control" ng-model="currentOferta.name" />
    </div>
    <div class="form-group">
        <label>Discount:</label>
        <input class="form-control" ng-model="currentOferta.discount" />
    </div>
    <div class="form-group">
        <label>Amount:</label>
        <input class="form-control" ng-model="currentOferta.amount" />
    </div>
    <button class="btn btn-primary" ng-click="createOrder()">Buy</button>
    <button class="btn btn-primary" ng-
click="cancelEditOferta()">Cancel</button>
</div>
```

***CUSTEditorOfertaView.html (view)***




Como se ha comentado al inicio de este análisis, para entender la definición de los servicios y la definición de las rutas de la aplicación es necesario explicar previamente los servicios *$resource* , *$location* y *$route.*

*$resource*

AngularJS ofrece este servicio definido dentro del módulo *ngResource* para dar soporte a las aplicaciones en su interacción con servicios web REST, liberando a los desarrolladores de tratar los detalles de las peticiones Ajax y los formatos URL.

La llamada al servicio $resource retorna un objeto de tipo *Access Object* sobre el cual, por defecto, están definidos los métodos que se muestran en la *Tabla [T01].*

*Tabla [T01]*

| Nombre | Http | Url | Descripción |
|---|---|---|---|
| `delete(object)` | `DELETE` | `/baseUrl/objectId` | Elimina el objeto con el ID especificado |
| `get(id)` | `GET` | `/baseUrl/objectId` | Recupera el objeto con el ID especificado |
| `query()` | `GET` | `/baseUrl` | Recupera todos los objetos en un array |
| `remove(object)` | `DELETE` | `/baseUrl/objectId` | Elimina el objeto con el ID especificado |
| `save(object)` | `POST` | `/baseUrl/objectId` | Guarda las modificaciones en el objeto con el ID especificado. Si no existiera lo crea y le asigna el ID especificado. |

El servicio admite una reconfiguración de los métodos definidos por defecto y/o una ampliación de los mismos mediante la definición de nuevos.

Analicemos la definición del servicio customersResource:

```
.factory("ofertasResource"      , function ($resource, baseUrlOferta) {
    return $resource(baseUrlOferta + ":id", { id: "@ofertaId" },
          {getofertashotel: {method: "GET", params: {hotelId: "@hotelId"}, url:
baseUrlOferta + "hotel/" + ":hotelId", isArray:true}} );
})
```

Se aplica el método factory sobre el objeto módulo para generar un nuevo servicio de nombre *ofertasResource* y función constructora *function ($resource, baseUrlOferta){}*. Es esta función constructora la que realiza una llamada al servicio *$resource* que admite tres parámetros (cada uno de ellos subrayado por separado) . El primer parámetro hace referencia a la *url* en la que reside el servicio web, el segundo indica de dónde tomar los valores de los segmentos no estáticos de la url que han sido marcados con ":" en el primer parámetro, y el tercer parámetro permite reconfigurar el servicio *$resource*. Es en este tercer parámetro dónde se crea el nuevo método *getofertas que* realiza una petición *http GET* al servidor, tienen un segmento variable en su *url* (*hotelId*) y recibe por respuesta un *array* de resultados. Teniendo en cuenta que la constante *baseUrlOferta* ha sido definida como `"http://localhost:8080/HotelsUOCFase1/tfguoc/ofertas/")`, el servicio realiza una petición *http GET* a los recursos representados por la URL:

```
"http://localhost:8080/HotelsUOCFase1/tfguoc/ofertas/ hotel /hotelId"
```

Se adjunta la porción de código del método *handler (o manejador)* de la petición, pertenecientes al controlador Oferta*Controller* y residente en la capa servidor (en el apartado Spring se explicará con detalle qué significa cada una de las anotaciones).

```
@Controller
@RequestMapping("/ofertas")
public class OfertaController {

      @Autowired
      OfertaService ofertaFacade;

                                    [ ... ]

      @RequestMapping(value="/hotel/{hotelid}",method=RequestMethod.GET)
      public @ResponseBody List<Oferta> getOfertasHotel(@PathVariable String
hotelid){
            List<Oferta> ofertas = ofertaFacade.getOfertasHotel(hotelid);

            return ofertas;
      }
                                    [ ... ]
}
```

*$location*

Una de las principales características del servicio *$location es que p*ermite acceder a la URL actual del navegador para consultarla o modificarla, simplificando mucho la navegación a través de las diferentes vistas de nuestra aplicación.

Son varios los métodos definidos en este servicio pero el más explotado ha sido el método *path () / path(String url).* Si no se pasa parámetro alguno, el método *path* retorna la *url* actual, caso de pasar un *String*, el método lo interpretará como la nueva *url* a cargar en el navegador.

En los controladores (ver ejemplo en los controladores *homeCustomerCtrl y editOfertaCtrl* del *subconjunto-ejemplo del cliente AngularJS)* se utiliza el método path repetidamente para redireccionar a una nueva ruta cada vez que finaliza la ejecución de un método invocado desde alguna vista.

*$route*

El servicio $route, que permite una navegación óptima y sencilla a través de las vistas de la aplicación, está definido en el módulo opcional de AngularJS ngRoute. Las funcionalidades proporcionadas por este servicio giran alrededor de un conjunto de mapeos entre las urls y los nombres de los ficheros de las vistas de la aplicación, denominadas rutas url. Estos mapeos se definen a través del servicio proveedor del servicio $route, $routeProvider.

Observando las rutas "*/listOferta*" y "*/CUSTEditOferta/:id*" incluidas en el *subconjunto-ejemplo del cliente AngularJS,* se observa como la definición se produce aplicando el método when de $routeProvider que a su vez ofrece un conjunto de propiedades de configuración tales como *templateUrl*, *controller* y *resolve* que se comentan a continuación:

- La propiedad de configuración *templateUrl* especifica la *url* del fichero de la vista que se muestra o "displaya" cada vez que la ruta coincide con la demandada por el navegador.

- La propiedad de configuración *controller* especifica el nombre del controlador asociado a la vista que la ruta muestra o "displaya". Una nueva instancia del controlador será creada cada vez que la vista sea mostrada (este hecho es importante pues simplifica el trabajo del desarrollador, librándole de trabajar con eventos que informen de los cambios de las rutas).

- La propiedad de configuración *resolve* permite especificar dependencias, que serán inyectadas al controlador especificado mediante la propiedad de configuración *controller*, cada vez que éste sea instanciado.

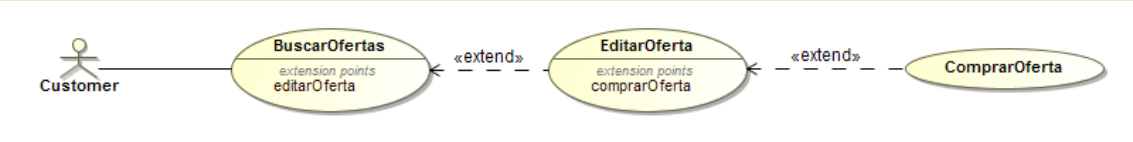
Cabe remarcar que las rutas pueden tener parámetros, por ejemplo la ruta "*/CUSTEditOferta/:id*" presenta un parámetro de nombre *id* en el segundo segmento. Esto significa que todas las rutas que tengan dos segmentos y el primero se corresponda con */CUSTEditOferta/* serán consideradas de este tipo. La posibilidad de incluir parámetros en las rutas y que éstos puedan ser accedidos por el servicio $routeParams dota de gran flexibilidad a la navegabilidad de la aplicación.

Puede apreciarse como se utiliza el servicio *$routeParams* en el controlador *editOfertaCtrl* para recuperar el parámetro *id* de la ruta y conocer así qué oferta debe mostrar la vista encargada de editarla (CUSTEditorOfertaView.html).

**En el anexo dos todas las rutas**

**Análisis técnico del caso de uso Comprar*Oferta***

Como resumen final del apartado "curva de aprendizaje/AngularJS", se realiza en este párrafo un análisis técnico detallado del caso de uso Comprar*Oferta*, utilizando para ello toda la información contenida en el *subconjunto-ejemplo del cliente AngularJS*.




Veamos qué secuencia de procesos se desencadenan cuando pulsamos el botón buscar ofertas en el *home* del perfil *Customer*, luego pulsamos el botón edit para editar cualquiera de las ofertas y finalmente pulsamos el botón Buy para comprar la oferta que hemos editado.

La definición de la ruta "*/homeCustomer*" determina que cada vez que se carga la ruta en el navegador se muestra la vista *CUSTHomeView.html* que tiene asociado el controlador homeCustomerCtrl.

Fijándonos en el código *html* de la vista *CUSTHomeView.html* observamos que se ha asociado mediante la directiva *ng-click* la ejecución del método hoteles() (perteneciente al controlador homeCustomerCtrl) al evento de pulsar el botón "Buscar Hoteles".

En el fragmento de código perteneciente a dicho controlador podemos constatar que lo

único que realiza el método $scope.hoteles es recargar la ruta "*/listOferta*" en el navegador.

La definición de la ruta "*/listOferta* " determina que cada vez que se carga la ruta en el navegador se muestra la vista *CUSTTableOfertaView.html* que tiene asociado el controlador *tableOfertaCtrl*.

En el código *html* de la vista *CUSTTableOfertaView.html* observamos que, el evento de pulsar el botón *Edit* de cualquiera de las filas de la lista de ofertas que muestra la vista, nos redirige a la ruta */CUSTEditOfert/{{item.ofertaId}}.*

La definición de la ruta "*/CUSTEditOferta/:id* " determina que, cada vez que se carga la ruta en el navegador, se muestra la vista CUSTEditorOfertaView.html cuyo controlador asociado es *editOfertaCtrl*.

Finalmente, pulsando el botón *Buy* de dicha vista ejecutamos el método *$scope.createOrder* del controlador *editOfertaCtrl*.

Y es en este fragmento de código en el que:

- Se crea un nuevo objeto *order* partiendo de los campos de la oferta editada *($scope.currentOferta)* y del *username* del usuario que realiza la compra, almacenado en el proceso de login (*$scope.data.username*).

- Se invoca el método *save* del servicio *ordersResource* para dotar de persistencia al objeto

- Se carga una nueva ruta en el navegador ( la ruta correspondiente al listado de órdenes de compra).

ANGULARJS by Google

**(Anexo 1)**

```
angular.module("exampleApp"      , ["ngResource", "ngRoute"])
.constant("baseUrlCustomer"                                                    ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/customers/")
.constant("baseUrlHotelAdmin"                                                  ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/hotelAdmins/")
.constant("baseUrlHotel"                                                       ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/hotels/")
.constant("baseUrlOferta"                                                      ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/ofertas/")
.constant("baseUrlOrder"                                                       ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/orders/")
.constant("baseUrlComentario"                                                  ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/comentarios/")
.constant("baseUrlFavorito"                                                    ,
"http://localhost:8080/HotelsUOCFase1/tfguoc/favoritos/")
.factory("customersResource"     , function ($resource, baseUrlCustomer) {
    return $resource(baseUrlCustomer + ":id", { id: "@username" });
})
.factory("hotelAdminsResource"   , function ($resource, baseUrlHotelAdmin) {
    return $resource(baseUrlHotelAdmin + ":id", { id: "@username" });
})
.factory("hotelsResource"        , function ($resource, baseUrlHotel) {
     return $resource(baseUrlHotel + ":id", { id: "@hotelId" });
})
.factory("ofertasResource"       , function ($resource, baseUrlOferta) {
    return $resource(baseUrlOferta + ":id", { id: "@ofertaId" },
           {getofertashotel:  {method:  "GET",  params:  {hotelId:  "@hotelId"},  url:
baseUrlOferta + "hotel/" + ":hotelId", isArray:true}} );
})
.factory("ordersResource"        , function ($resource, baseUrlOrder) {
    return $resource(baseUrlOrder + ":id", { id: "@orderId" },
          {getcomprashotel:  {method:  "GET",  params:  {hotelId:  "@hotelId"},  url:
baseUrlOrder + "hotel/" + ":hotelId", isArray:true},
           getcomprascustomer: {method: "GET", params: {username: "@username"}, url:
baseUrlOrder + "customer/" + ":username", isArray:true}});
})
.factory("comentariosResource"   , function ($resource, baseUrlComentario) {
    return $resource(baseUrlComentario + ":id", { id: "@comentarioId" },
          {getcomentarioshotel: {method: "GET", params: {hotelId: "@hotelId"}, url:
baseUrlComentario + "hotel/" + ":hotelId", isArray:true},
           getcomentarioscustomer: {method: "GET", params: {username: "@username"},
url: baseUrlComentario + "customer/" +  ":username", isArray:true}});
})
.factory("favoritosResource"     , function ($resource, baseUrlFavorito) {
    return $resource(baseUrlFavorito + ":id", { id: "@favoritoId" },
          {getfavoritoscustomer:  {method:  "GET",  params:  {username:  "@username"},
url: baseUrlFavorito + "customer/" + ":username", isArray:true}});
})
.config(function ($routeProvider, $locationProvider) {

  $locationProvider.html5Mode(true);
```

```
$routeProvider.when("/login", {
    templateUrl: "/loginView.html",
    controller: "loginCtrl"
});

$routeProvider.when("/loginCreateCustomer", {
    templateUrl: "/SACreateCustomerView.html",
    controller: "editCustomerCtrl"
});

$routeProvider.when("/loginCreateHotelAdmin", {
    templateUrl: "/SACreateHotelAdminView.html",
    controller: "editHotelAdminCtrl"
});
$routeProvider.when("/homeSuperAdmin", {
    templateUrl: "/SAHomeView.html",
    controller: "homeSuperAdminCtrl"
});

$routeProvider.when("/listCustomer", {
    templateUrl: "/SATableCustomerView.html",
    controller: "tableCustomerCtrl",
    resolve: {
        dataCustomer: function (customersResource) {
            return customersResource.query();
        }
    }
});

$routeProvider.when("/editCustomer/:id", {
    templateUrl: "/SAEditorCustomerView.html",
    controller: "editCustomerCtrl"
});

$routeProvider.when("/createCustomer", {
    templateUrl: "/SACreateCustomerView.html",
    controller: "editCustomerCtrl"
});

$routeProvider.when("/listHotelAdmin", {
    templateUrl: "/SATableHotelAdminView.html",
    controller: "tableHotelAdminCtrl",
    resolve: {
        dataHotelAdmin: function (hotelAdminsResource) {
            return hotelAdminsResource.query();
        }
    }
});

$routeProvider.when("/editHotelAdmin/:id", {
    templateUrl: "/SAEditorHotelAdminView.html",
    controller: "editHotelAdminCtrl"
});

$routeProvider.when("/createHotelAdmin", {
```

```
    templateUrl: "/SACreateHotelAdminView.html",
    controller: "editHotelAdminCtrl"
});

$routeProvider.when("/SAListHotel", {
    templateUrl: "/SATableHotelView.html",
    controller: "tableHotelCtrl",
    resolve: {
        dataHotel: function (hotelsResource) {
            return hotelsResource.query();
        }
    }
});

$routeProvider.when("/SAEditHotel/:id", {
    templateUrl: "/SAEditorHotelView.html",
    controller: "editHotelCtrl"
});

$routeProvider.when("/createHotel", {
    templateUrl: "/SACreateHotelView.html",
    controller: "editHotelCtrl"
});

$routeProvider.when("/SAUnderConstruction", {
    templateUrl: "/underConstructionView.html",
    controller: "underConstructionCtrl"
});

$routeProvider.when("/homeCustomer", {
    templateUrl: "/CUSTHomeView.html",
    controller: "homeCustomerCtrl"
});

$routeProvider.when("/CUSTListHotel", {
    templateUrl: "/CUSTTableHotelView.html",
    controller: "tableHotelCtrl",
    resolve: {
        dataHotel: function (hotelsResource) {
            return hotelsResource.query();
        }
    }
});

$routeProvider.when("/CUSTEditHotel/:id", {
    templateUrl: "/CUSTEditorHotelView.html",
    controller: "editHotelCtrl"
});

$routeProvider.when("/CUSTEditHotelFavorito/:id", {
    templateUrl: "/CUSTEditorFavoritoView.html",
    controller: "editHotelCtrl"
});

$routeProvider.when("/CUSTListFavorito", {
    templateUrl: "/CUSTTableFavoritoView.html",
```

```
    controller: "tableFavoritoCtrl"
});

$routeProvider.when("/listOferta", {
    templateUrl: "/CUSTTableOfertaView.html",
    controller: "tableOfertaCtrl",
    resolve: {
        dataOferta: function (ofertasResource) {
            return ofertasResource.query();
        }
    }
});

$routeProvider.when("/CUSTEditOferta/:id", {
    templateUrl: "/CUSTEditorOfertaView.html",
    controller: "editOfertaCtrl"
});

$routeProvider.when("/CUSTListOfertaHotel", {
    templateUrl: "/CUSTTableOfertaView.html",
    controller: "tableOfertaHotelCtrl"
});

$routeProvider.when("/listOrderCustomer", {
    templateUrl: "/CUSTTableOrderView.html",
    controller: "tableOrderCtrl"
});

$routeProvider.when("/editOrderCustomer/:id", {
    templateUrl: "/CUSTEditorOrderView.html",
    controller: "editOrderCtrl"
});

$routeProvider.when("/listComentarioCustomer", {
    templateUrl: "/CUSTTableComentarioView.html",
    controller: "tableComentarioCtrl"
});

$routeProvider.when("/CUSTlistComentarioHotel", {
    templateUrl: "/CUSTHATableComentarioView.html",
    controller: "tableComentarioCtrl"
});

$routeProvider.when("/CUSTEditComentario/:id", {
    templateUrl: "/CUSTEditorComentarioView.html",
    controller: "editComentarioCtrl"
});

$routeProvider.when("/CUSTHAEditComentario/:id", {
    templateUrl: "/CUSTHAEditorComentarioView.html",
    controller: "editComentarioCtrl"
});

$routeProvider.when("/HAEditComentario/:id", {
    templateUrl: "/HAEditorComentarioView.html",
    controller: "editComentarioCtrl"
```

```
        });

        $routeProvider.when("/createComentario", {
            templateUrl: "/CUSTCreateComentarioView.html",
            controller: "editComentarioCtrl"
        });

        $routeProvider.when("/CUSTUnderConstruction", {
            templateUrl: "/underConstructionView.html",
            controller: "underConstructionCtrl"
        });

        $routeProvider.when("/homeHotelAdmin", {
            templateUrl: "/HAHomeView.html",
            controller: "homeHotelAdminCtrl"
        });

        $routeProvider.when("/HAListOfertaHotel", {
            templateUrl: "/HATableOfertaView.html",
            controller: "tableOfertaHotelCtrl"
        });

        $routeProvider.when("/HAEditOferta/:id", {
            templateUrl: "/HAEditorOfertaView.html",
            controller: "editOfertaCtrl"
        });

        $routeProvider.when("/createOferta", {
            templateUrl: "/HACreateOfertaView.html",
            controller: "editOfertaCtrl"
        });

        $routeProvider.when("/listOrderHotel", {
            templateUrl: "/HATableOrderView.html",
            controller: "tableOrderCtrl"
        });

        $routeProvider.when("/editOrderHotel/:id", {
            templateUrl: "/HAEditorOrderView.html",
            controller: "editOrderCtrl"
        });

        $routeProvider.when("/listComentarioHotel", {
            templateUrl: "/HATableComentarioView.html",
            controller: "tableComentarioCtrl"
        });

        $routeProvider.when("/HAUnderConstruction", {
            templateUrl: "/underConstructionView.html",
            controller: "underConstructionCtrl"
        });
        $routeProvider.otherwise({
            templateUrl: "/loginView.html",
            controller: "loginCtrl"
        });
})
```

```
.controller("defaultCtrl", function ($scope, $route) { [...] })

.controller("loginCtrl", function ($scope, $location, customersResource, hotelAdminsResource, hotelsResource) { [...] })

.controller("homeSuperAdminCtrl", function ($scope, $location) { [...] })

.controller("tableCustomerCtrl", function ($scope, $location, $route, dataCustomer) { [...] })

.controller("editCustomerCtrl", function ($scope, $routeParams, $location, customersResource) { [...] })

.controller("tableHotelAdminCtrl", function ($scope, $location, $route, dataHotelAdmin) { [...] })

.controller("homeHotelAdminCtrl", function ($scope, $location) { [...] })

.controller("homeCustomerCtrl", function ($scope, $location) { [...] })

.controller("tableHotelCtrl", function ($scope, $location, $route, dataHotel) { [...] })

.controller("editHotelCtrl", function ($scope, $routeParams, $location, hotelsResource, ofertasResource, favoritosResource, comentariosResource) { [...] })

.controller("tableOfertaCtrl", function ($scope, $location, $route, dataOferta) { [...] })

.controller("tableOfertaHotelCtrl", function ($scope, $location, $route, ofertasResource) { [...] })

.controller("editOfertaCtrl", function ($scope, $routeParams, $location, ofertasResource, ordersResource) { [...] })

.controller("tableOrderCtrl", function ($scope, $location, $route, ordersResource) { [...] })

.controller("editOrderCtrl", function ($scope, $routeParams, $location, ordersResource) { [...] })

.controller("tableComentarioCtrl", function ($scope, $location, $route, comentariosResource) { [...] })

.controller("editComentarioCtrl", function ($scope, $routeParams, $location, comentariosResource) { [...] })

.controller("underConstructionCtrl", function ($scope, $location) { [...] });
```


**(Anexo 2)**

## Tabla de rutas URL

*Establece una relación entre rutas URL, Vistas y controladores*

| Ruta URL | Controlador | Fichero Html de la vista |
|---|---|---|
| login | loginCtrl | loginView.html |
| loginCreateCustomer | editCustomerCtrl | SACreateCustomerView.html |
| loginCreateHotelAdmin | editHotelAdminCtrl | SACreateHotelAdminView.html |
| homeSuperAdmin | homeSuperAdminCtrl | SAHomeView.html |
| listCustomer | tableCustomerCtrl | SATableCustomerView.html |
| editCustomer/:id | editCustomerCtrl | SAEditorCustomerView.html |
| createCustomer | editCustomerCtrl | SACreateCustomerView.html |
| editCustomerCtrl | tableHotelAdminCtrl | SATableHotelAdminView.html |
| editHotelAdmin/:id | editHotelAdminCtrl | SAEditorHotelAdminView.html |
| createHotelAdmin | editHotelAdminCtrl | SACreateHotelAdminView.html |
| SAListHotel | tableHotelCtrl | SATableHotelView.html |
| SAEditHotel/:id | editHotelCtrl | SAEditorHotelView.html |
| createHotel | editHotelCtrl | SACreateHotelView.html |
| SAUnderConstruction | underConstructionCtrl | underConstructionView.html |
| homeCustomer | homeCustomerCtrl | CUSTHomeView.html |
| CUSTListHotel | tableHotelCtrl | CUSTTableHotelView.html |
| CUSTEditHotel/:id | editHotelCtrl | CUSTEditorHotelView.html |
| CUSTEditHotelFavorito/:id | editHotelCtrl | CUSTEditorFavoritoView.html |
| CUSTListFavorito | tableFavoritoCtrl | CUSTTableFavoritoView.html |
| listOferta | tableOfertaCtrl | CUSTTableOfertaView.htm |
| CUSTEditOferta/:id | editOfertaCtrl | CUSTEditorOfertaView.html |
| CUSTListOfertaHotel | tableOfertaHotelCtrl | CUSTTableOfertaView.html |
| listOrderCustomer | tableOrderCtrl | CUSTTableOrderView.html |
| editOrderCustomer/:id | editOrderCtrl | CUSTEditorOrderView.html |
| listComentarioCustomer | tableComentarioCtrl | CUSTTableComentarioView.html |
| CUSTlistComentarioHotel | tableComentarioCtrl | CUSTHATableComentarioView.html |
| CUSTEditComentario/:id | editComentarioCtrl | CUSTEditorComentarioView.html |
| CUSTHAEditComentario/:id | editComentarioCtrl | CUSTHAEditorComentarioView.html |
| HAEditComentario/:id | editComentarioCtrl | HAEditorComentarioView.html |
| createComentario | editComentarioCtrl | CUSTCreateComentarioView.html |
| CUSTUnderConstruction | underConstructionCtrl | underConstructionView.html |
| homeHotelAdmin | homeHotelAdminCtrl | HAHomeView.html |
| HAListOfertaHotel | tableOfertaHotelCtrl | HATableOfertaView.html |
| HAEditOferta/:id | editOfertaCtrl | HAEditorOfertaView.html |
| createOferta | editOfertaCtrl | HACreateOfertaView.html |
| listOrderHotel | tableOrderCtrl | HATableOrderView.html", |
| editOrderHotel/:id | editOrderCtrl | HAEditorOrderView.html |
| listComentarioHotel | tableComentarioCtrl | HATableComentarioView.html |
| HAUnderConstruction | underConstructionCtrl | underConstructionView.html |


Spring se basa en unas pocas ideas fundamentales, todas ellas focalizadas a la consecución del principal objetivo de este Framework: Simplificar el desarrollo en Java. Las cuatro estrategias clave que emplea Spring Framework para conseguirlo son:

- Desarrollo ligero y poco invasivo utilizando POJOs.
- Bajo acoplamiento mediante la utilización de la inyección de dependencias (dependency injection) y el diseño orientado a interfaces.
- Programación declarativa a través de aspectos.
- Reducción de la duplicidad de código por mediación de la utilización de plantillas (templates).

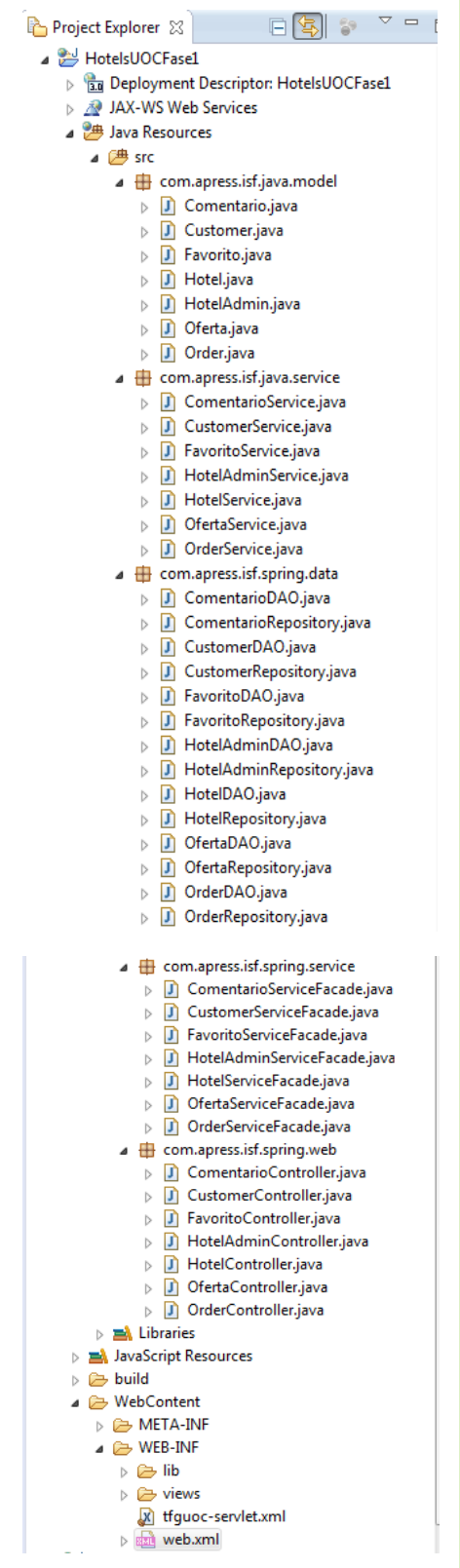
Spring es un Framework basado en el concepto Contenedor. Los Contenedores utilizan inyección de dependencias para gestionar los componentes que constituyen la aplicación, esto incluye crear asociaciones entre ellos, facilitar su reutilización, facilitar el ciclo de testing y aumentar la claridad y comprensión del código.

Es necesario configurar los contenedores de Spring ( Spring Framework pone varios tipos de contenedores a disposición de los desarrolladores) para indicar qué beans deben contener y como enlazarlos entre sí.

La configuración de Spring se define en uno o varios ficheros XML. Se han desarrollado técnicas basadas en anotaciones para aligerar el contenido de dichos ficheros y dotar de mayor eficiencia a la aplicaciones.

En la aplicación del presente Trabajo Final de Grado se han utilizado dos ficheros XML para configurar Spring: tfguoc-servlet.xml y web.xml cuyo contenido se muestra en la página 67. Se analizarán las características más destacadas de estos ficheros y las anotaciones en el código de las clases java en las páginas 68,69 y 70.

Se tratará también a lo largo de esta sección otras aspectos importantess tales como el soporte que Spring ofrece para integrar Hibernate en la capa de persistencia y las herramientas que ofrece para construir servicios web REST.




### *Estructura y arquitectura de la capa servidor de la aplicación*

A la izquierda de este texto se muestra la estructura de toda la capa servidor de la aplicación en el proyecto de Eclipse *HotelsUOCFase1*.

Se pueden apreciar cuatro packages diferentes que contienen un total de 28 clases y 14 interfaces java. Por su parte, la carpeta WEB-INF contiene los ficheros de configuración *tfguoc-servlet.xml* y *web.xml.*

En la página 66 se muestra un diagrama de componentes que explica la arquitectura de la capa servidor de la aplicación:

Hay siete controladores relacionados cada uno de ellos con una sola entidad o fuente de datos de la aplicación, a saber: *Comentario, Customer, Favorito, Hotel, HotelAdmin, Oferta* y *Order*. Cada controlador utiliza los métodos de una interface que ofrece servicios. Esta interface es implementada por una clase siguiendo el patrón fachada, que a su vez , utiliza una interface DAO (Data Access Object). Y esta última interface es implementada por una clase que accede a la base de datos y gestiona la fuente de datos asociada al controlador.

Cada tipo de clase (controladoras, de servicio y de acceso a datos) está anotada con diferentes anotaciones que serán comentadas a pie de diagrama

Finalmente, el diagrama de la página 67 es un zoom de una parte del diagrama de componentes de la página 66.

***Diagrama de Componentes (1): Arquitectura de la capa servidor de la aplicación***

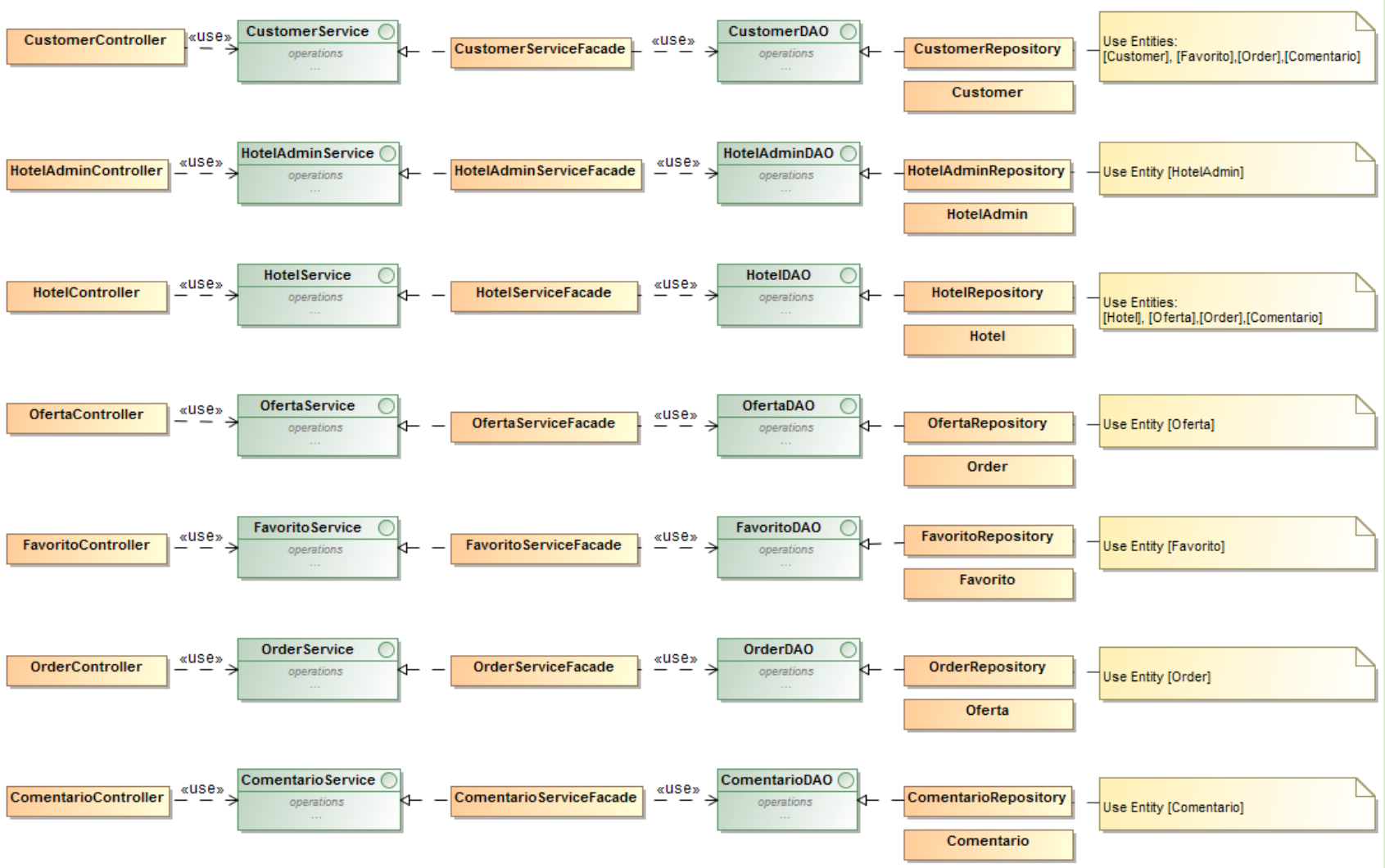
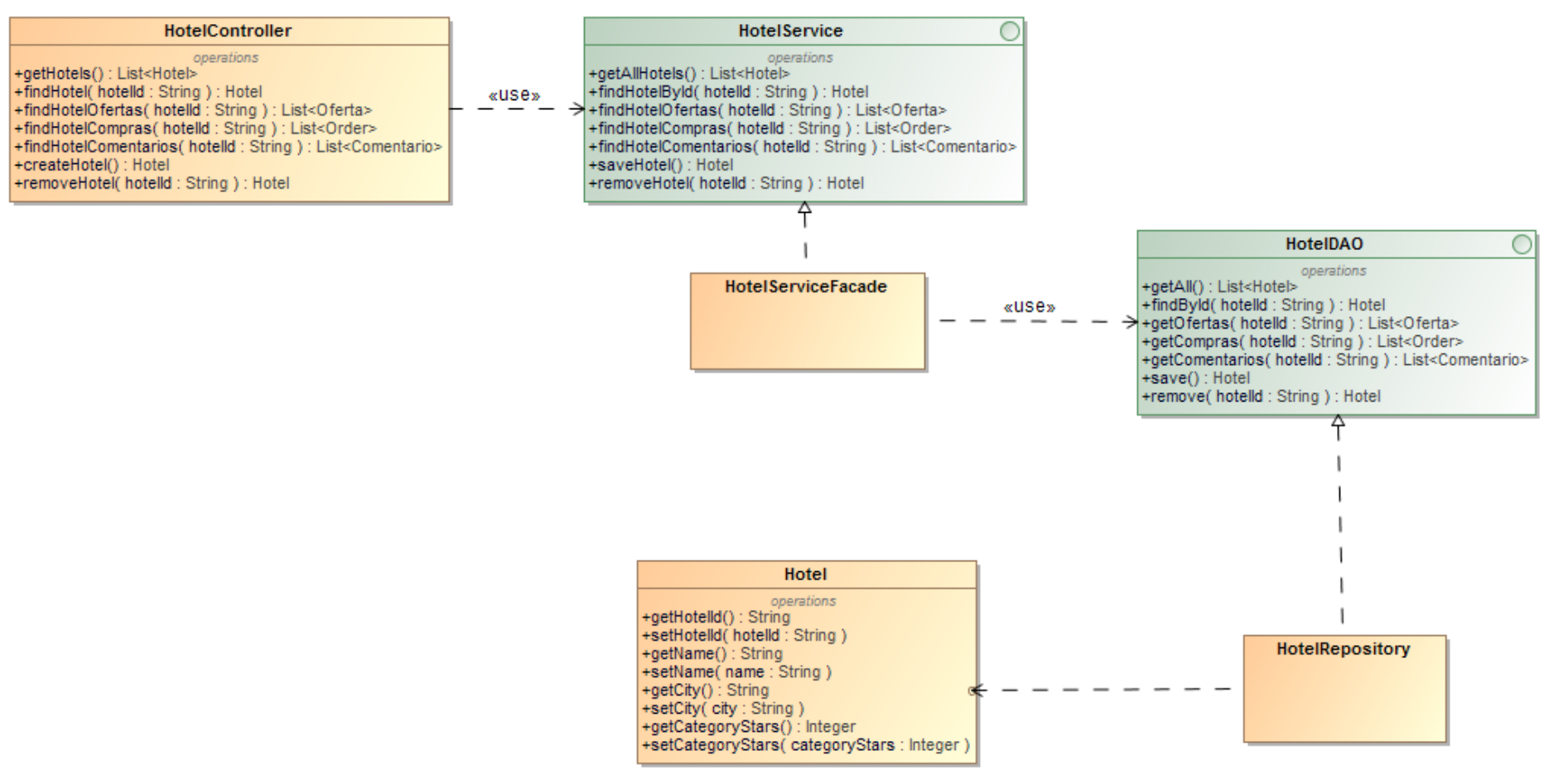


***Diagrama de componentes (2): Ampliación de la Arquitectura de la capa servidor de la aplicación (clase HotelController y todos los componente relacionados)***



### *Aspectos a destacar del fichero de configuración [ tfguoc-servlet.xml ]*

Localización de todos los espacios de nombres y esquemas.

```
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd
  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd">
```

Búsqueda automática de clases anotadas como @Component, @Controller y @Repository en el package 'com.apress.isf' y sus subpackages (esto es, todos los packages de la aplicación). Toda las clases encontradas tras el escaneo serán dadas de alta como beans en el contenedor.

```
<context:component-scan base-package="com.apress.isf"/>
```

Configuración del bean de tipo *DriverManagerDataSource* encargado de gestionar la conexión con la base de datos. Se le pasan como parámetros el driver de la base de datos, la localización de la misma y las credenciales.

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="org.postgresql.Driver"/>
      <property name="url" value="jdbc:postgresql://localhost:5432/hoteluoc"/>
      <property name="username" value="USER"/>
      <property name="password" value="PASSWORD"/>
</bean>
```

Configuración del bean tipo *InternalResourceViewResolver* , uno de los resolutores de vistas que ofrece Spring, le indicamos en que carpeta encontrar las vistas y la extensión de las mismas.

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/views/"/>
  <property name="suffix" value=".jspx"/>
</bean>
```

La manera habitual de conseguir una referencia de una sesión Hibernate es a través de la implementación de la interface SessionFactory de Hibernate. SessionFactory es la responsable de abrir, cerrar y gestionar las sesiones en Hibernate. En Spring la manera de conseguir una SessionFactory de Hibernate es por mediación de uno de los 'factory beans' de Spring. En nuestro caso hemos configurado LocalSessionFactoryBean, indicándole la fuente de datos con la que se trabaja (la definida en el bean id="dataSource") ,que packages escanear en busca de clases de domino, etc.

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="packagesToScan" value="com.apress.isf.java.model" />
      <property name="hibernateProperties">
          <props>
              <prop key="hibernate.dialect">org.hibernate.dialect.PostgreSQL82Dialect</prop>
          </props>
      </property>
      <property name="annotatedClasses">
              <list>
                  <value>com.apress.isf.java.model.Hotel</value>
              </list>
      </property>
</bean>
```

Este bean se configura para dotar de comportamiento transaccional al bean id="sessionFactory" si se utiliza la anotación @Transactional.

```
<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
      <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

## *Aspectos a destacar de las anotaciones en el código*

```java
package com.apress.isf.spring.web;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.PathVariable;

import com.apress.isf.java.model.Hotel;
import com.apress.isf.java.model.Oferta;
import com.apress.isf.java.model.Order;
import com.apress.isf.java.model.Comentario;
import com.apress.isf.java.service.HotelService;

@Controller
@RequestMapping("/hotels")
public class HotelController {

    @Autowired
    HotelService hotelFacade;

    @RequestMapping(method=RequestMethod.GET)
    public @ResponseBody List<Hotel> getHotels(){
        List<Hotel> hotels = hotelFacade.getAllHotels();
        return hotels;
    }

    @RequestMapping(value="/{hotelid}",method=RequestMethod.GET)
    public @ResponseBody List<Oferta> findHotel(@PathVariable String hotelid){
        return hotelFacade.findHotelOfertas(hotelid);
    }
```

*Fragmento de código de la clase HotelController.*

*Todas las clases _Controller de la aplicación presentan las siguientes anotaciones:*

*@Controller*
*@Autowired*
*@RequestMapping*
*@PathVariable*

```java
package com.apress.isf.spring.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Component;

import com.apress.isf.java.model.Comentario;
import com.apress.isf.java.model.Hotel;
import com.apress.isf.java.model.Oferta;
import com.apress.isf.java.model.Order;
import com.apress.isf.java.service.HotelService;
import com.apress.isf.spring.data.HotelDAO;

@Component
public class HotelServiceFacade implements HotelService {

    @Autowired
    HotelDAO hotelDAO;

    public List<Hotel> getAllHotels(){
        return hotelDAO.getAll();
    }
```

*Fragmento de código de la clase HotelServiceFacade.*

*Todas las clases _ServiceFacade de la aplicación presentan las siguientes anotaciones:*

*@Component*
*@Autowired*

```java
package com.apress.isf.spring.data;

import java.util.List;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.apress.isf.java.model.Hotel;
import com.apress.isf.java.model.Oferta;
import com.apress.isf.java.model.Order;
import com.apress.isf.java.model.Comentario;

@Repository
@Transactional
public class HotelRepository implements HotelDAO{

    private SessionFactory sessionFactory;

    @Autowired
    public HotelRepository(SessionFactory sessionFactory){
        this.sessionFactory=sessionFactory;
    }

    @SuppressWarnings("unchecked")
    public List<Hotel> getAll(){
        return (List<Hotel>) sessionFactory.getCurrentSession().createQuery("from Hotel").list();
    }
```

*Fragmento de código de la clase HotelRepository.*

*Todas las clases _Repository de la aplicación presentan las siguientes anotaciones:*

*@Repository*
*@Transactional*
*@Autowired*
(Con esta última anotación se inyecta un bean de tipo SessionFactory)

***@Autowired***
Marca un constructor, campo o método para ser autoenlazado por el contenedor Spring, utilizando técnicas de inyección por dependencia (dependency injection).

***@Component***
Indica que la clase anotada es un componente. Estas clases son consideradas como candidatas para la detección automática cuando se utiliza la configuración basada en anotación y escaneo `(<context:component-scan base-package="com.apress.isf"/>)`.

***@Controller***
Indica que la clase anotada es un controlador. Estas clases son consideradas como candidatas para la detección automática cuando se utiliza la configuración basada en anotación y escaneo `(<context:component-scan base-package="com.apress.isf"/>)`.

***@Repository***
Indica que la clase anotada es un 'repositorio'. Estas clases son consideradas como candidatas para la detección automática cuando se utiliza la configuración basada en anotación y escaneo `(<context:component-scan base-package="com.apress.isf"/>)`. Todas las excepciones en la capa de persistencia de una plataforma especifica (Hibernate en nuestro caso) son capturadas y relanzadas como una excepción propia de Spring (permiten afinar mejor cuál ha sido el error en la capa de persistencia) por el bean *PersistenceExceptionTranslationPostProcessor*. Todas las clases anotadas con @Repository podrán utilizar este servicio.

***@Transactional***
Toda clase anotada con @Transactional recibirá todo el equipamiento software necesario para dotarla de un comportamiento transaccional.

***@RequestMapping***

Mapea un método manejador con la URL y el tipo de petición http a manejar.

***@PathVariable***

Anotación que indica que el parámetro de método se extrae de una variable en la URL que el método maneja.

***@ResponseBody***

# Conclusiones

Estoy muy satisfecho de las tecnologías seleccionadas para la realización de este Trabajo Final de Grado. Considero que Spring es un Framework muy consolidado y con gran cantidad de herramientas CASE nacidas para ser utilizadas en su entorno (MAVEN podría ser un ejemplo). Quizás al principio es algo arduo entender su arquitectura y funcionamiento, pero estoy convencido que a medio y largo plazo, es una ganancia de tiempo y eficiencia importante. No había trabajado nunca con Spring, pero sin lugar a dudas lo voy a empezar a hacer a partir de ahora.

Por su parte, el Framework AngularJS ha sido un gran descubrimiento para mi, creo que es un Framework de gran potencia y no me cabe la menor duda que se convertirá en líder indiscutible. Está avalado por el sello Google y considero que es un Framework elegante y muy eficiente.

El trabajo ha sido realmente duro y muchas horas y sacrifico se ha invertido en ello, pero sin lugar a dudas ha merecido la pena.

**Treball Final del Grau en Enginyeria Informàtica**
**Universitat Oberta de Catalunya**
**13 de Gener de 2015**
**Sergi Martín Sandoval**