Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN PARA LA VENTA ON-LINE DE COMIDA A DOMICILIO

Alumno: Juan Ramón Sánchez Sola

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Departamento de Informática

Junio, 2017

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Proyecto Fin Grado titulado: Estudio y desarrollo de un prototipo de aplicación para la venta on-line de comida a domicilio, que presenta Juan Ramón Sánchez Sola, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JUNIO de 2017

El alumno: Los tutores:

Juan Ramón Sánchez Sola José Ramón Balsas Almagro

Índice

1. CAPÍTULO 1: INTRODUCCIÓN

1.1. Propósito.

La venta de comida on-line está cada día más integrada en nuestras vidas diarias. Una de las causas más importantes es la comodidad de comprar y recibir los pedidos a domicilio sin la necesidad de movernos de casa. Esta tendencia se enmarca también dentro del 'boom' que suponen en la actualidad las aplicaciones de economía colaborativa [1] donde los usuarios registrados intercambian y comparten bienes y servicios.

La tecnología, entre sus muchas aplicaciones, cubre cada vez más necesidades humanas, haciéndonos más cómodas rutinas que realizamos diariamente.

El propósito general de este trabajo fin de grado es el estudio e implementación de una aplicación web basada en tecnologías JEE en el lado del servidor, y de la interfaz con la que se podrá comunicar el usuario adaptada a dispositivos móviles. La aplicación estará basada en principios de diseño y *Frameworks* de desarrollo especialmente adaptados a este tipo de aplicaciones. Se podrán dar de alta usuarios y los ofertantes de comida que podrán ser particulares y/o negocios profesionales con entrega a domicilio. Cada usuario es el responsable de concretar con una compañía de transporte la entrega del pedido que le han notificado.

Gracias a un mapa de *Google Maps*, podremos buscar por ubicación, calle, ciudad o pueblo que deseemos para buscar comida típica de esa zona.

El contexto en que se va a utilizar la aplicación será tanto en el ámbito profesional (negocios profesionales que ofrecen comida a domicilio) como en el particular. Cualquier persona registrada podrá ver el producto que ofrecen profesionales o particulares, y poder contactar con ellos para adquirirlo. Los usuarios podrán ver las valoraciones y opiniones de otros compradores.

La motivación principalmente es dar respuesta a una necesidad básica como es la alimentación diaria, permitiendo, quizás, conseguir precios más reducidos respecto al precio de venta en establecimientos normales, y con la posibilidad añadida de probar comida típica de una zona en particular, recibiendo el pedido que realicemos a domicilio.

1.2. Objetivos.

Para el desarrollo del presente TFG se han planteado los siguientes objetivos:

- Realizar un estudio de necesidades y soluciones para la venta on-line de comida.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas y metodologías de desarrollo web.
- Implementar un prototipo de aplicación web usando tecnologías JEE en el lado del servidor y una interfaz adaptativa basada en HTML5 en el lado del cliente.

1.3. Metodología.

A lo largo del desarrollo del TFG se han ido realizando diferentes procesos y utilizando distintos tipos de métodos y herramientas para cada una de las fases que componen un proyecto de Ingeniería del Software.

En primer lugar se hace un estudio de los requerimientos del sistema. Es una tarea de ingeniería del software que permite especificar las características funcionales del sistema, la interfaz del software con otros elementos del sistema y establecer las restricciones que debe cumplir el sistema, para dar una visión global del sistema que se pretende desarrollar. También se describen las tecnologías y Frameworks utilizados tanto para el análisis como para el desarrollo. Para la mejor compresión de los términos y/o conceptos utilizados en el presente documento, se realiza una recopilación bibliográfica sobre la temática y tecnologías que se nombran y utilizan en el desarrollo del TFG.

Se desarrollará también una estimación temporal y de costes de desarrollo. Al utilizar la metodología Ágil SCRUM [2], la estimación temporal se estudiará mediante la estimación de historias de usuario, analizando también la velocidad del equipo, iteraciones y duración del proyecto. Para la estimación de costes se detallarán aspectos como los gastos materiales, equipos, servicios, licencias de software y nóminas de los roles necesarios para acometer las diferentes fases del proyecto.

La metodología utilizada para el diseño del sistema será una metodología de orientación a objetos utilizando el patrón de diseño MVC[1]. Todos los diagramas desarrollados cumplen las normas y estándares del modelado UML [3]. Cada uno de ellos se verá detalladamente en capítulos posteriores de este presente documento.

[1] Siglas del patrón de arquitectura software Modelo-Vista-Controlador

Una vez acometido el análisis y diseño, se comenzará a implementar el prototipo del sistema.

1.4. Estructura del documento.

Este documento está organizado de la siguiente manera:

En el apartado 2 del presente documento se expone el análisis del proyecto, dividido en varios puntos:

En el primer punto se detalla el análisis preliminar, en el que se profundiza el contexto del que se parte de la aplicación, así como problemáticas y necesidades que puedan darse. Se hará hincapié en puntos positivos y negativos que habría que tener en cuenta en la aplicación.

También se detallan los *frameworks* utilizados que servirán para obtener las funcionalidades del sistema.

En el segundo punto se habla de la propuesta de solución del problema donde se explica el objetivo al que se quiere llegar en el desarrollo de la aplicación, es decir, tener una visión inicial de la aplicación que se quiere conseguir.

El tercer punto del apartado define los requerimientos del sistema, es decir, los requisitos funcionales y no funcionales del sistema. En los requisitos funcionales se detallan las funcionalidades generales y restricciones que se espera de la aplicación, como por ejemplo, registro de usuario o acceso a una determinada vista dependiendo del rol del usuario. En los requisitos no funcionales se encuentran los relacionados con la elección del lenguaje utilizado, base de datos, etc.

En el cuarto punto se estudia la planificación inicial del proyecto y se analizan costes, tiempos que dedica cada rol en el desarrollo o análisis mediante la estimación de las historias de usuario.

El quinto punto del apartado presenta un estudio de la viabilidad del proyecto, donde se realizará un análisis de costes de desarrollo, entre los que destacan, nóminas de los distintos roles que han trabajado en el proyecto y costes de licencias.

En el sexto punto se muestra el diagrama de dominio inicial explicando los aspectos más relevantes de la información del sistema.

El apartado 3 estudia el diseño del sistema. En primer lugar se estudia el diagrama entidad-relación y su forma normal explicando algunas de los aspectos más relevantes, como claves primarias y foráneas.

En el segundo punto se presenta el diagrama de clases completo del sistema.

Después, se exponen los diagramas de secuencia más relevantes de la aplicación web para tener una idea general del comportamiento interno del sistema.

En el cuarto punto se puede ver el diseño de la interfaz con algunos bocetos de pantallas de la aplicación.

El cuarto apartado es el referente a la implementación del mismo donde podremos entender la arquitectura y los detalles de la solución definitiva.

Tras la introducción, el primer punto describe la arquitectura que se ha seguido para crear la aplicación y las tablas con las las tablas resumen con las URL's[2] que ofrecen los servicios REST implementados, en el segundo punto se describen los detalles de la implementación, donde se enumeran los frameworks y tecnologías utilizadas con sus respectivas referencias.

[2] Siglas del concepto Localizador de recursos uniforme LRU (del inglés, Uniform Resource Locator)

En el quinto capítulo se presentan las conclusiones del trabajo realizado, reflexionando sobre el nivel de alcance de los objetivos inicialmente propuestos, además de otros aspectos a tener en cuenta para la comprensión general del proyecto. Por último, se exponen posibles ideas para mejorar en un futuro el proyecto.

El sexto y último apartado de este presente documento es la bibliografía, donde se han encontrado recursos para realizar este proyecto, ya sean libros o texto en internet. Para ello se ha seguido el formato IEEE donde cada referencia se identifica con un identificador únicoEl sexto y último apartado de este presente documento es la bibliografía. Para completarla se ha seguido el formato IEEE donde cada referencia se identifica con un identificador único.

Al final de la memoria se pueden encontrar algunos apéndices con información complementaria como el manual de instalación, el manual de usuario o la descripción de los contenidos adicionales suministrados en formato digital.

2. CAPÍTULO 2: ANÁLISIS

En este capítulo realizaremos en primer lugar un análisis preliminar del contexto del sistema propuesto que nos permita, posteriormente, obtener ideas para presentar una propuesta de solución donde se dará una visión general del sistema que se pretende desarrollar y un estudio de viabilidad donde se analizarán costes para tener una visión aproximada del coste total del proyecto.

2.1. Análisis preliminar

La economía colaborativa en España es un sector que está en continuo crecimiento en España y en otros lugares del mundo, como dice José María Torrego

en su reportaje [4]: “La economía colaborativa crece entre el 15 y el 17% anualmente y a nivel mundial. El antiguo trueque se ve ahora favorecido por Internet, la tecnología y las redes sociales, la mayoría de las plataformas actuales se erigen como puntos de encuentro entre personas que tienen objetivos e intereses comunes.”

En este mismo reportaje podemos encontrar grandes listas de aplicaciones basadas en empresas con un sistema de economía colaborativa, entre ellas podemos encontrar, empresas vinculadas a los viajes, empresas de cultura y ocio, intercambio y alquiler de productos o servicios, redes profesionales, etc.

Para que la economía colaborativa apareciera, han tenido que coincidir varios factores de ámbito económico, social y tecnológico como se explica en este informe [5].

En el ámbito social hay factores como la mayor densidad de población en la que aparecen las economías de red, a más usuarios conectados más valor al producto y se produce un efecto multiplicador que se asocia a la economía de la información, la sostenibilidad, donde se tiene mayor sensibilidad al desarrollo sostenible, al medio ambiente y mejor uso de recursos, como también el deseo de comunidad pueden influir en el impulso de este tipo de economía.

En el ámbito económico hay factores como la disminución de la renta, la flexibilidad económica que supone una aplicación de este tipo o la crisis económica también han influido en el impulso de esta economía.

En el ámbito tecnológico influyen factores como las redes sociales y la sociedad en red, los dispositivos móviles y las plataformas tecnológicas, o los diferentes métodos de pago seguros que existen en la actualidad.

El sistema que se pretende desarrollar es una aplicación web adaptativa a dispositivos móviles, basada en economía colaborativa en la que los usuarios podrán contactar con otros, dando la oportunidad de disfrutar de comida típica de otros lugares.

Cada vendedor será responsable de concretar con una empresa de transporte para hacer llegar el producto al comprador. Se podrá seleccionar entre los diferentes puntos en el mapa (usuarios registrados), ver productos que ofrecen, valorar y comentar productos que se hayan probado, leer opiniones de otros usuarios sobre el producto y se podrá contactar mediante mensaje privado dentro de la aplicación.

Se mostrará el teléfono de cada usuario, dando libertad total en el contacto entre vendedores y compradores, aunque no quedará constancia en la aplicación del pedido que se realice y no se podrá valorar o comentar los productos.

Si el pedido se realiza dentro de la aplicación, podemos ayudar a la comunidad a decidir en la elección de productos y fomentar la fiabilidad entre usuarios.

Este tipo de aplicación va dirigida al sector de la gastronomía donde se conectan cocineros y comensales mediante plataformas web, también adaptadas a dispositivos móviles.

A continuación se detallan algunos puntos de un estudio de mercado [6] sobre el consumo de comida a domicilio en España, el cual nos puede proporcionar una visión global de este tipo de hábitos.

El siguiente gráfico muestra datos estadísticos sobre diferentes costumbres en el consumo de comida.

Ilustración 1. Hábitos de consumo de comida [6]

Mediante el gráfico se puede deducir que:

- Al 70% le gusta cocinar pero,
- Solo uno de cada diez tiene tiempo para hacerlo
- Más de la mitad de la población tiende a rechazar la comida rápida y solo uno de cada cuatro aprecia la comida congelada o preparada.
- A la mayor parte de la población le gusta comer en casa (65%).
- El 12% tiene alguna restricción alimentaria.

En este otro gráfico podemos ver como el 40% de la población recurre al menos una vez por semana a comida preparada o a domicilio.

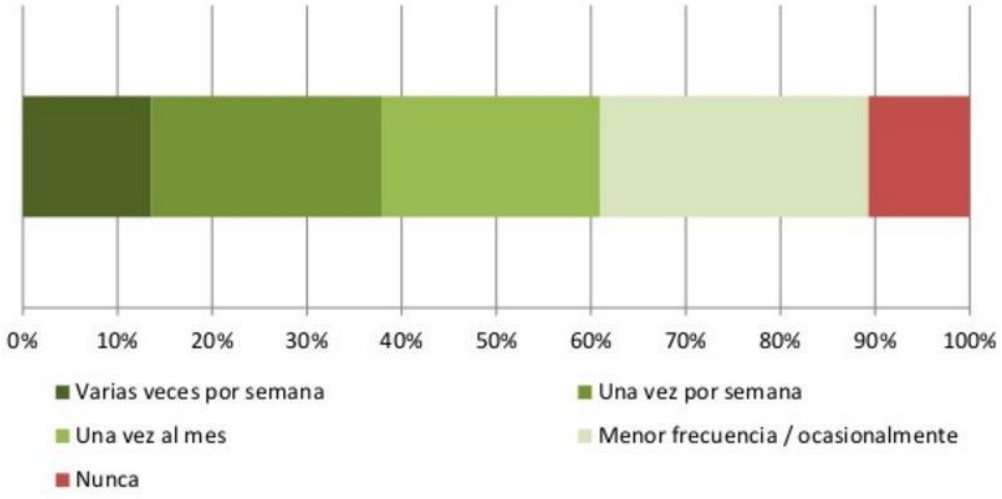

Ilustración 2. Porcentaje de personas que recurren a comida preparada o a domicilio [6]

También se deduce que una vez que los usuarios prueban a pedir comida online, aumenta el porcentaje y vuelven a repetir esta acción.

Ilustración 3. Aumento del porcentaje al pedir comida online [6]

A continuación se mencionan algunas aplicaciones basadas en economía colaborativa, comentando algunos aspectos positivos y negativos, los cuales nos permiten obtener ideas útiles para este proyecto.

En primer lugar, Blablacar[3] es una aplicación basada en economía colaborativa donde los usuarios pueden compartir viajes indicando el origen, el destino, hora de salida, el precio y plazas libres entre otras. Otro usuario precisa realizar el mismo trayecto y contacta con el anunciante para compartir el transporte.

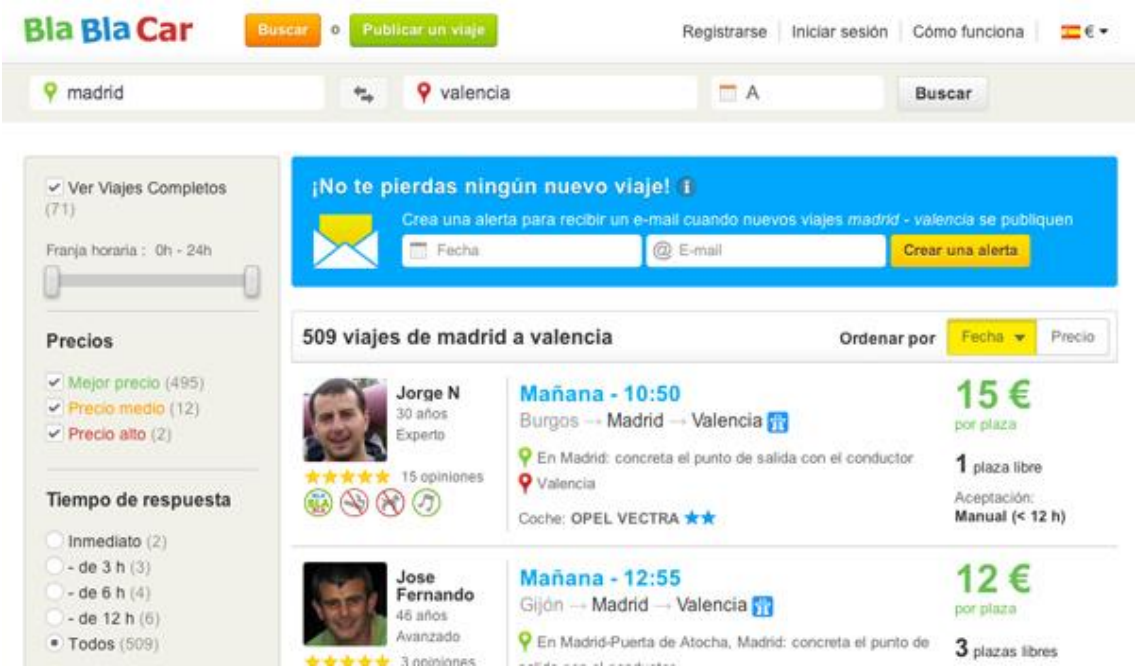

Ilustración 4. Aplicación BlaBlaCar.

Entre las ventajas de esta aplicación, cabe destacar:

[3] https://www.blablacar.es/

- La disminución de contaminación para el medio ambiente. Si cada individuo realizase el viaje de forma individual y no compartiera las plazas del transporte, la contaminación sería mayor.
- Ahorro para el consumidor. El mismo trayecto realizado a través de una empresa de transporte tendría un coste más elevado.
- Conocer gente nueva viajando juntos.
- Disponibilidad horaria. Este aplicación ofrece una disponibilidad horaria mayor frente a las empresas de transporte, gracias al elevado número de usuarios registrados.

Blablacar fue denunciada por empresas de transporte en 2015 [7], las cuales encontraron grandes pérdidas en sus ingresos. BlablaCar tubo que añadir un coste de gestión de viaje, y además eliminaron la libertad de poder contactar libremente entre los usuarios, para algunos usuarios esto añade puntos negativos a la aplicación, aunque para otros usuarios pueda ofrecer un aumento de fiabilidad.

Como esta aplicación tuvo un gran éxito aparecieron otras que ofrecen servicios similares como es el caso de Amovens[4], que por el momento permite el contacto entre usuarios de forma particular sin validación del mensaje y sin coste adicional al propuesto por el ofertante del viaje.

Otra aplicación interesante para el ámbito de la gastronomía es Foodfriends[5], ofrece ventajas como poder comer buscando en sus cercanías a otros usuarios que también le apetece comer, da la oportunidad de conocer a gente nueva, poder quedar con otros usuarios registrados en ciudades que no conocemos para poder probar la mejor opción de comida de la zona, filtrar por rating y ver opiniones de otros usuarios.

[4] https://amovens.com/
[5] https://foodfriends.net/es

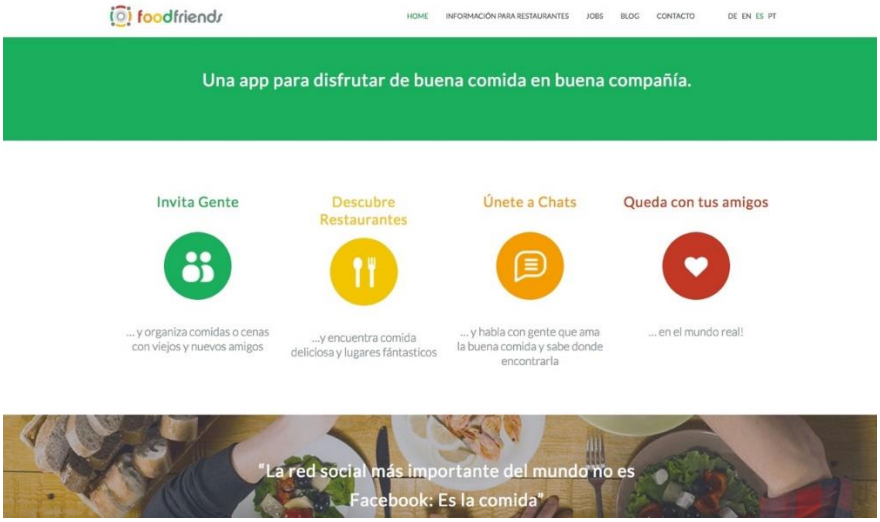

Ilustración 5. Aplicación FoodFriends.net

El sistema que se pretende desarrollar en este presente documento ofrece muchas ventajas, algunas de ellas similares a las comentadas en aplicaciones anteriores, entre las que destacan disfrutar de comida típica de otros lugares como ocurre en FoodFriends.net.

También tiene un coste reducido frente a negocios profesionales, ya que el intercambio es directamente entre particulares, como en Amovens, aunque también se incluyan profesionales.

Puedes conocer gente nueva como en las tres aplicaciones anteriormente nombradas.

Contactar con otros usuarios dentro de la aplicación con privacidad, como ofrece Amovens.

El principal inconveniente de esta idea de proyecto es el estado en que se puedan entregar los productos, el peligro que conlleva para la salud ingerir un alimento en mal estado.

Una solución para este problema es la valoración de cada usuario y producto, ofrece fiabilidad y confianza a la hora de realizar un pedido a un cierto usuario.

A continuación se justifica la utilización de las distintas tecnologías para resolver ciertos problemas descritos en la propuesta, concretamente el apartado 2.1.1 .

2.1.1. Análisis de Tecnologías

En este apartado expondremos brevemente las diferentes tecnologías que podría utilizarse para el diseño de la solución propuesta y las ventajas que podría aportar a su desarrollo, estas son las siguientes:

- Desarrollo web basado en Java EE en el lado del servidor: El Framework utilizado es Java EE 7[6]. Este tipo de aplicaciones proporcionan lógica de negocio, se gestionan de forma centralizada y su historia comienza con Java Enterprise Edition (J2EE 1.3). Aparece en 2001 para el desarrollo de aplicaciones empresariales distribuidas. Fue muy complejo y tuvo demasiadas críticas, entonces Spring Framework ganó en aceptación frente a J2EE.

  En 2006 aparece Java EE 5 rediseñado para seguir los principios de Spring Framework y más adelante, en 2009 se anunció la versión Java EE 6, donde siguen rediseñando el Framework hasta la presente utilizada.

[6] http://www.oracle.com/technetwork/java/javaee/overview/index.html

- Interfaz adaptativa basada en HTML5[7] en el lado del cliente: la interfaz de usuario se realiza con el Framework JSF 2.2 (Java Server Faces[8]), este está diseñado para ser independiente de protocolos específicos y lenguajes de marcado, pero también se utiliza en conjunto con servlets y JSP's.

En la versión 1.x se utilizan como sistema de plantillas por defecto, en cambio en esta nueva versión se utiliza Facelets.

Se dirige específicamente a guardar el estado del componente de interfaz de usuario entre las solicitudes en la sesión de un cliente web, además proporciona un mecanismo de renderizado para abordar el soporte de cada cliente web de un determinado lenguaje, dispone también de una serie de características convenientes para procesar solicitudes basadas en formularios, la validación de estos, un mecanismo para manejar excepciones para informarlas de nuevo a la interfaz de usuario. A menudo se usa con AJAX.

- Patrón de diseño MVC (Modelo-Vista-controlador): Este tipo de aplicaciones web están basados en el protocolo de transferencia de hipertexto (HTTP) sin estado. Un usuario interactúa con una página en el navegador Web y este envía una solicitud al servidor. Un controlador escucha peticiones con una URL y cuando recibe una solicitud, el controlador interactúa con el modelo y luego determina qué componente Vista le está asignado.

Se suelen implementar con ficheros XHTML y como parte de una respuesta el componente Vista puede consultar el Modelo.

[7] siglas de HyperText Markup Language
[8] http://www.javaserverfaces.org/

El modelo se define mediante beans, que no es más que una clase con atributos y métodos que devuelven, actualizan y borran valores, y estas propiedades se pueden leer y escribir desde las páginas JSF utilizando el lenguaje de expresiones EL.

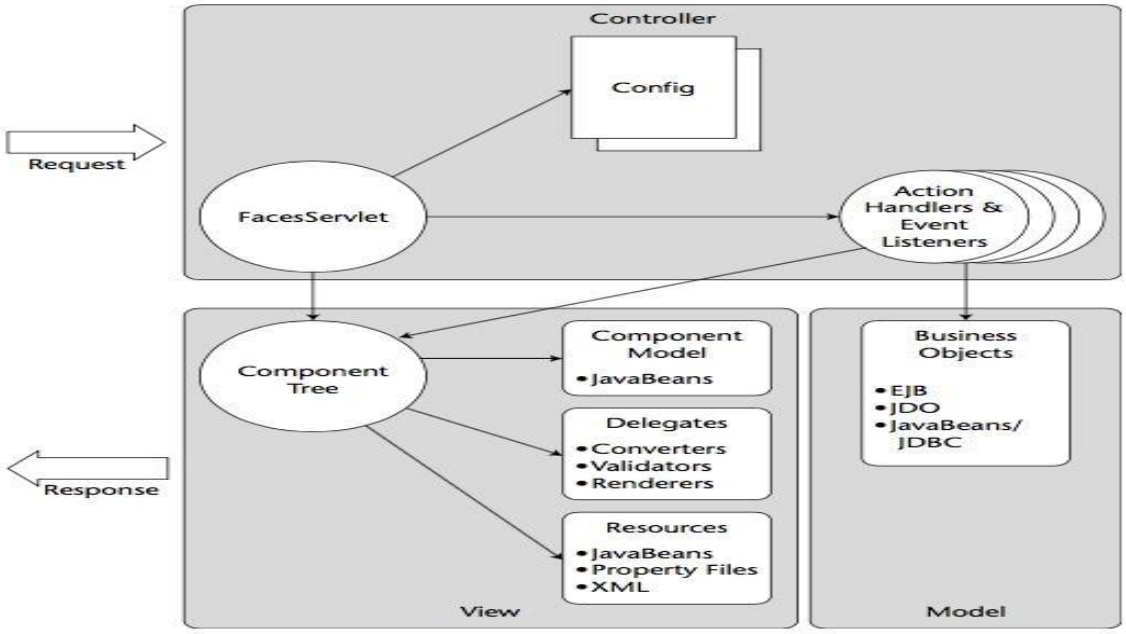

Ilustración 6. JSF Model-2 [7]

En *Ilustración 6* podemos observar que componentes pertenecen a cada parte del patrón y cómo se comunican entre sí.

- Creación de servicios web REST[9]: Como se muestra en *Ilustración 7* los servicios REST soportan diferentes formatos, los más utilizados son JSON[10] y XML[11], a diferencia de SOAP que sólo utiliza XML, esta es solo una ventaja de porqué los servicios REST han ido sustituyendo a los servicios SOAP.

[9] siglas de Representational State Transfer
[10] siglas de JavaScript Object Notation
[11] siglas de Extensible Markup Language

REST es un patrón de arquitectura, no define un protocolo ni formato de mensaje concreto, es mucho más ligero y se adapta a clientes con capacidades limitadas como los dispositivos móviles.

En este proyecto el servidor (Back-end) ofrece los servicios Web REST accesibles mediante URL's ofreciendo éstos al cliente.

El sistema a desarrollar trabajará con archivos JSON tanto el cliente como el servidor. JSON es un estándar abierto que utiliza texto plano para codificar la información, es muy utilizado entre servicios Web y API´s REST.

Para la creación de estos servicios se ha utilizado el Framework JAX-RS 2.0 server api[12], esta es una API de Java que proporciona soporte para crear servicios web REST, usa anotaciones en los beans (POJOS), atributos o métodos de este.

A partir de la versión 1.1, JAX-RS forma parte de Java EE.

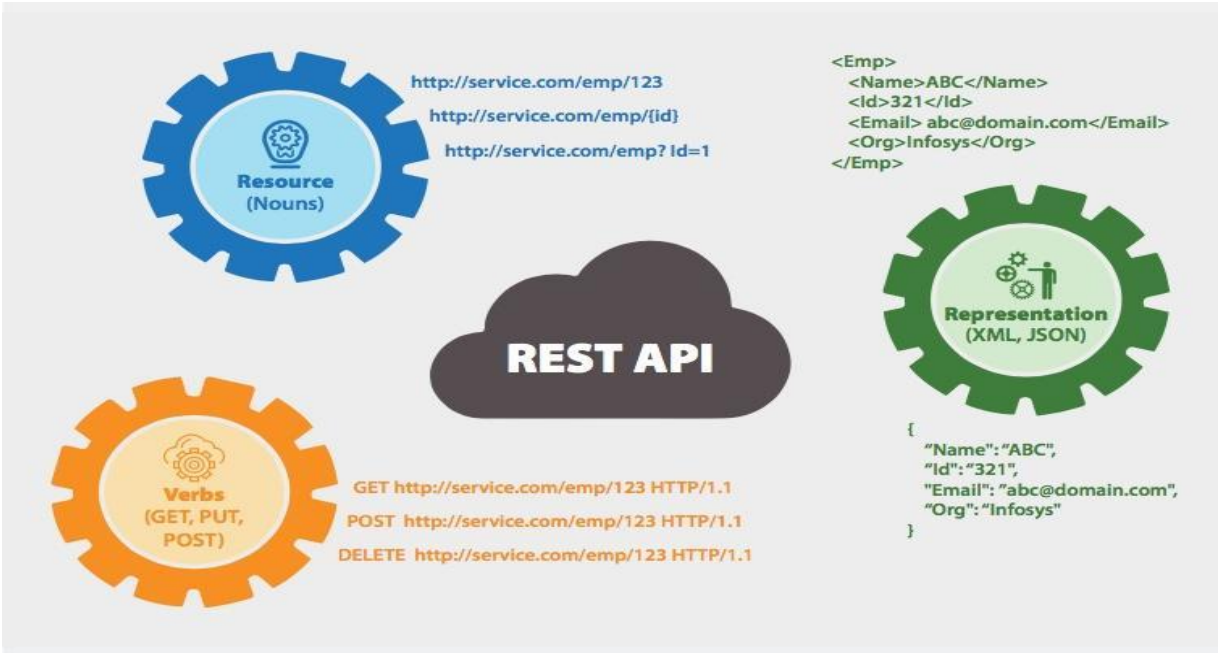

Ilustración 7. Building RESTful Web services [8]

[12] https://jax-rs-spec.java.net/nonav/2.0/apidocs/

En *ilustración 7* tenemos un esquema de los elementos fundamentales de esta plataforma web. Los recursos tienen un identificador único en la plataforma, es decir, una URL única. No tiene límites en el número de URL's que se refieren a un recurso. Los verbos son una acción HTTP, los más utilizados son POST, GET, PUT, DELETE.

- Persistencia de Base de Datos basada en JPA: El Framework utilizado para esta parte es Hibernate ORM[13], permite a los desarrolladores escribir más fácilmente aplicaciones cuyos datos sobreviven al proceso de solicitud, como Object / Relational Mapping.

Hibernate tiene que ver con la persistencia de datos que se aplica a las bases de datos relacionales (a través de JDBC). Además es también una implementación de la especificación Java Persistence API (JPA).

Se puede utilizar en casi cualquier entorno, incluyendo aplicaciones Java SE, servidores de aplicaciones Java EE, contenedores de OSGi para empresas, etc.

Permite desarrollar clases persistentes orientadas a objetos, incluyendo la herencia, polimorfismo, asociación, composición y el marco de las colecciones de Java. No requiere de interfaces o clases base para las clases persistentes y permite cualquier estructura de clases a ser persistentes.

Soporta inicialización perezosa, no requiere ninguna tabla en base de datos o campos y genera gran parte del SQL en el momento de inicialización en lugar de en tiempo de ejecución.

[13] Object / Relational Mapping

También ofrece consistentemente un rendimiento superior sobre el código JDBC directamente, tanto en términos de productividad como en rendimiento en tiempo de ejecución.

2.2. Propuesta de solución

La idea principal del proyecto a desarrollar es un aplicación web adaptativa a dispositivos móviles basada en economía colaborativa, donde usuarios registrados, ya sean particulares y/o profesionales, puedan compartir sus recetas gastronómicas de una zona en particular con otras personas, se podrá contactar con otro usuario mediante mensaje respetando la privacidad, y buscar usuarios registrados navegando por un mapa.

También se podrá buscar usuarios por provincia, ciudad o una dirección en particular.

Se podrá seleccionar usuarios registrados pulsando sobre un punto en el mapa y poder ver los productos que ofrece, valoración de otros usuarios y comentarios, esto nos ayudará mejor a decidir a quién podemos comprarle.

Los pedidos se enviarán a domicilio, cada vendedor es responsable de concretar con una empresa de transporte para entregar el pedido. El estado del pedido irá cambiando por los usuarios de pendiente a aceptado, y de aceptado a entregado.

Cuando esté entregado el comprador podrá valorar los productos y comentar.

Para implementar los servicios REST de la parte del servidor se utilizará JAX-RS server api y JPA con Hibernate para la persistencia de los datos.

La parte de cliente se desarrollará con JAX-RS Client api para las peticiones a los servicios web REST remotos, Java Server Faces para las vistas con algunos componentes de Prime Faces 5.3 y su conexión con los controladores desde ellas.

La metodología de ingeniería del software que se utilizará será SCRUM, en el capítulo 3 de [2], se define SCRUM como un método de desarrollo Ágil de software concebido por Jeff Sutherland y su equipo de desarrollo a principio de los ‘90.

Los principios SCRUM son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis. Las tareas de trabajo ocurren con un patrón de proceso llamado ‘sprint’. Se utilizan historias de usuario, pruebas de aceptación y se hacen reuniones diarias cortas donde se comentan avances, problemas o próximos objetivos.

En el caso de este proyecto el grupo SCRUM esta formado por un miembro, por lo que la reunión diaria del grupo se puede descartar, ya que los avances, problemas y objetivos son obvios.

Además la estimación de las pruebas de usuario, si el grupo estubiera formado por más de un miembro, la estimación varía, ya que cada miembro tiene una opinión diferente. En este caso la estimación es única para cada historia de usuario, sin tener que hacer la media en cada una de ellas.

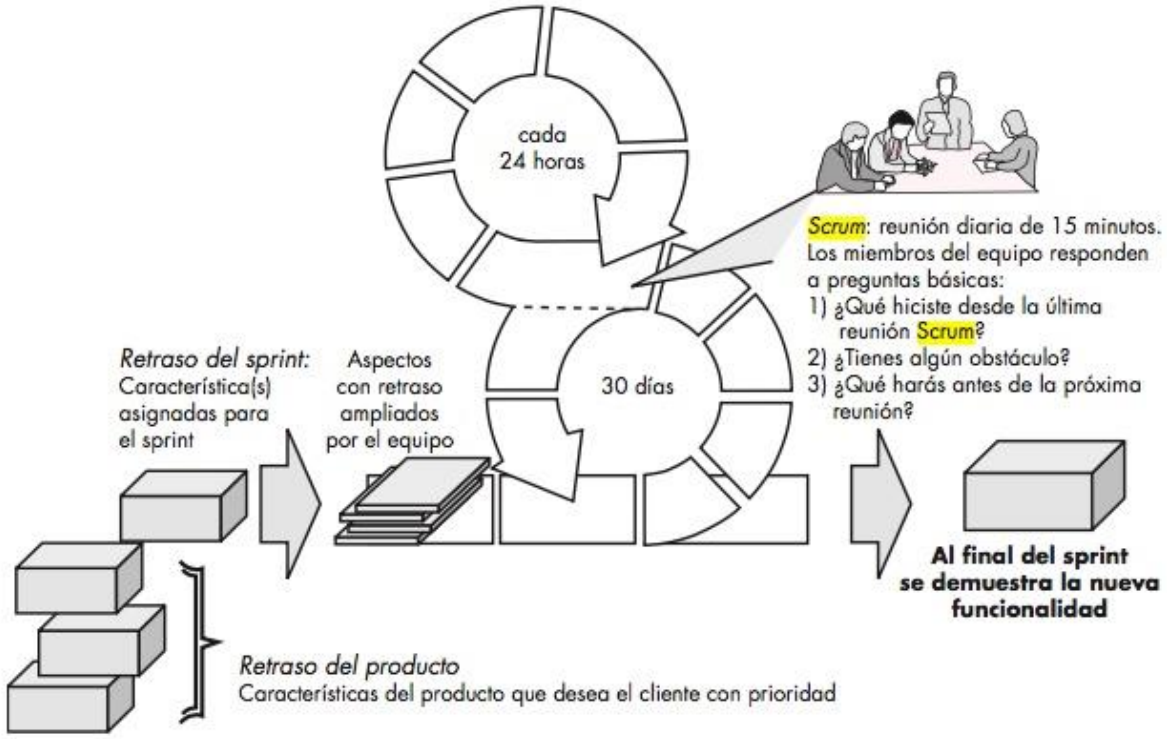

Ilustración 8. Flujo de proceso SCRUM [2]

2.3. Requisitos del sistema

Los requisitos del sistema [9] son la especificación de los estándares de calidad que se deben utilizar en el proceso, una especificación del diseño que se debe producir con una herramienta CASE particular y una descripción del proceso a seguir. Los requisitos del sistema pueden ser de dos tipos, requisitos funcionales o requisitos no funcionales.

2.3.1. Requerimientos Funcionales

Son declaraciones de ciertos servicios que debe ofrecer el sistema, de qué manera reaccionan a entradas particulares y cómo debe comportarse el sistema en situaciones particulares. A veces, también pueden declarar lo que el sistema no puede hacer, aunque lo más común es centrarse en lo que el sistema debe hacer. Pueden ser diferentes dependiendo del software que se quiera desarrollar, de los posibles usuarios que los utilizarán y del enfoque general del equipo o persona que va a desarrollar la aplicación.

Los requisitos funcionales de este sistema son los siguientes:

1. Registro de Usuario

El usuario podrá registrarse en el sistema introduciendo los campos de formulario. Nombre, email, teléfono de contacto,dirección, si el usuario es particular y/o profesional y la selección de la forma de pago. El campo dirección se auto completará gracias a la API de google, así se evitan introducir campos erróneos en la dirección, y recoger valores válidos en las coordenadas. Una vez registrado se insertará en el mapa de Google.

2. Disponer de cuenta propia y Cerrar Sesión

El usuario debe poder iniciar sesión y tener su cuenta propia con sus datos, pedidos, etc, además deberá cerrar sesión para que nadie pueda acceder.

3. Búsqueda de usuarios registrados.

Se podrá buscar por localidad, ciudad, código postal mediante un input, pudiendo ver los usuarios registrados en el mapa o navegando por él. También habrá una lista con los usuarios registrados junto al mapa.

4. Ver productos que ofrece un usuario.

Una vez seleccionado el usuario (punto en el mapa de Google Maps), podrán verse los productos que ofrece en una tabla con precio, nombre, imagen, descripción, valoración y comentarios de otros usuarios sobre el producto.

5. Añadir productos al carrito de la compra.

Para realizar un pedido, se irán añadiendo productos al carrito de la compra, habrá un pedido por cada usuario. Una vez confirmado el pedido por parte del usuario comprador, el vendedor podrá aceptarlo.

6. Ver estado del pedido.

Un usuario podrá ver el estado de su pedido en la vista *Mis pedidos,* el estado podrá ser Pendiente, Aceptado, Entregado. Mientras que el usuario vendedor no acepte, el usuario comprador tendrá Pendiente en el estado del pedido.

7. Elegir distintos tipos de Pago.

Los usuarios podrán elegir el método de pago al registrarse y podrán editarlo en la vista mi perfil.

8. Contactar con otros usuarios mediante mensaje privado.

Los usuarios podrán contactar entre sí mediante mensaje privado, además se dispondrá de una vista donde leer mensajes recibidos y enviados.

9. Valorar y comentar productos.

Después de recibir y probar los productos, se podrá valorar y comentar los productos para ayudar a los demás usuarios a decidir mejor.

10. Añadir aplicaciones cliente al sistema.

En un futuro sería conveniente añadir aplicaciones cliente al sistema, como pueden ser aplicaciones nativas para dispositivos móviles que utilicen los servicios REST que ofrece el servidor.

2.3.2. Requerimientos no funcionales

Los requerimientos no funcionales son restricciones de las funciones o servicios del sistema, no se suelen aplicar a características o servicios individuales. Pueden referirse a la usabilidad, seguridad, referentes al entorno, organizacionales, éticos, legislativos, etc.

Los requerimientos no funcionales de este sistema son:

1. El sistema debe tener interfaces gráficas con una apariencia estética coherente y adecuada.
2. El tiempo de aprendizaje de un usuario al usar el sistema debe ser menor a 2h.
3. El sistema debe proporcionar mensajes de error para informar al usuario final.
4. Los permisos para acceder al sistema solo pueden ser modificados por el administrador del lugar.

2.4. Historias de Usuario

Una historia de usuario [10] describe una funcionalidad que aporta valor al usuario. Debe ser breve y concisa, se utilizan para la especificación de requisitos y se suelen escribir en tarjetas o post-it cumpliendo el formato:

Como [rol], **quiero** [funcionalidad] **para** [beneficio].

Estas tareas deben seguir los principios INVEST[14].

A continuación se describen las historias de usuario en este sistema.

Historia de usuario 1.

Como *usuario*

quiero *registrarme en el sistema*

para *poder disfrutar de las ventajas que ofrece*

Historia de usuario 2.

Como *usuario*

quiero *iniciar sesión y cerrar sesión*

para *tener mi cuenta personal*

[14] Independent, Negotiable, Valuable, Estimable, Small, Testable

Historia de usuario 3.

Como usuario

quiero poder gestionar mis productos o pedidos en cualquier momento

para añadir, editar, mostrar o borrar los mismos

Historia de usuario 4.

Como *usuario*

quiero *buscar ofertantes de comida en un mapa*

para *poder ver productos que ofrecen*

Historia de usuario 5.

Como *usuario*

quiero *ver productos que ofrece cada ofertante*

para *poder ver datos, valoración y comentarios de otros usuarios.*

Historia de usuario 6.

Como *usuario*

quiero *añadir productos al carrito*

para *realizar un pedido*

Historia de usuario 7.

Como *usuario*

quiero *ver estado del pedido*

para *para poder estimar el tiempo para recibirlo*

Historia de usuario 8.

Como *usuario ofertante*

quiero *cambiar estado del pedido*

para *que el comprador sepa que le ha sido aceptado*

Historia de usuario 9.

Como *usuario*

quiero *elegir método de pago*

para *pagar mediante transferencia bancaria o contrareembolso*

Historia de usuario 10.

Como *usuario*

quiero *enviar mensajes privados*

para poder *contactar con otros usuarios*

Historia de usuario 11.

Como *usuario*

quiero *valorar y comentar productos*

para *colaborar con otros usuarios*

Historia de usuario 12.

Como *usuario*

quiero *disponer de una interfaz simple*

para *poder navegar por la aplicación*

Historia de usuario 13.

Como *usuario*

quiero *actualización asíncrona*

para evitar tiempo innecesario

2.5. Planificación inicial de tareas

Como se ha comentado anteriormente, la metodología ágil utilizada es SCRUM, por lo que a continuación se muestra una descomposición de las historias de usuario en tareas.

Las tareas se organizan en tablas de aceptación, siendo estas una descomposición de las historias de usuario propuestas por el equipo, una estimación de las mismas por puntos, donde se mide la complejidad.

También se describen las iteraciones que que se divide el proceso completo, la velocidad del equipo, y también el tiempo estimado para el desarrollo del proyecto a partir de los puntos.

2.5.1. Tablas de aceptación de las historias de usuario

Las historias de usuario pueden generar preguntas sobre cómo se llevarían a cabo, o cómo se ejecutarán, para solucionar este problema se utilizan las tablas de aceptación de las mismas. A continuación se muestran las pruebas de aceptación de cada historia de usuario.

Historia de Usuario	Pruebas de aceptación
Registrarse en el sistema	1. Se registran datos personales de usuario 2. Se hace la persistencia de los datos
Iniciar Sesión y Cerrar sesión	1. Se introducen las credenciales 2. Si son correctas el usuario será identificado y redirigido a la vista principal, sino, se redirigirá a una vista de error de login 3. Se debe poder cerrar sesión
Gestionar datos	1. Creación de base de datos 2. Creación de tablas mediante JPA con Hibernate
Buscar ofertantes de comida en un mapa	1. Búsqueda por ciudad, código postal, localidad o ubicación desde campo de entrada 2. Navegar por el mapa de Google Maps

Ver productos ofrecidos por un vendedor	1. Pulsar sobre marcas (usuario registrado) en Google Maps. 2. Pulsar sobre *Ver que ofrece* en *card* de información del usuario. 3. En la card del producto podremos ver datos, valoración y comentarios de otros usuarios
Añadir y quitar productos al carrito de la compra	1. En la ficha del producto pulsamos en añadir al carrito 2. En la parte lateral derecha se podrá observar nuestro carrito de la compra, con los productos, sus precios y el total a pagar, y botón de confirmar pedido 3. En el carrito podremos eliminar productos
Ver estado del pedido	1. En la vista Mis pedidos podremos ver estado del pedido, mientras el vendedor no acepte veremos el estado como Pendiente
Cambiar estado del pedido	1. En la vista Mis Peticiones, podemos aceptar el pedido 2. En la vista Mis pedidos, podremos cambiar a entregado si ha sido aceptado
Elegir método de pago	1. Al registrarnos podremos elegir el método de pago con el que se pagarán los pedidos.
Contactar con otros usuarios	1. En el listado de usuarios podremos pulsar un botón para escribir un mensaje al usuario.
Valorar y comentar productos	1. Una vez realizado un pedido, se podrá comentar y valorar los productos
Interfaz sencilla	1. La interfaz deberá ser sencilla para que el aprendizaje del usuario sea menor a 2 horas, pueda navegar sin problema por las diferentes vistas de la aplicación.
Actualización Asíncrona	1. Se actualizará parte de la vista automáticamente para evitar tiempo innecesario y la navegación por la aplicación sea más llevadera.

Tabla 1. Tabla de pruebas de aceptación de las historias de usuario

2.5.2. Estimación de historias de usuario

La estimación de las historias de usuario se puntuará con puntos usuario, los cuales se refieren más bien a la complejidad de la historia que a las horas que emplea un rol en realizarla.

En un equipo SCRUM, la complejidad de cada tarea es diferente para cada miembro, por lo que habría que llegar a un acuerdo. Como se ha comentado anteriormente, el equipo SCRUM está formado por un miembro, por lo que, no ha habido problema a la hora de decidir los puntos de historia estimados para cada tarea.

La estimación para cada historia de usuario es:

1. **Registrarse en el sistema**: 2 puntos.

Como *usuario* quiero *registrarme en el sistema* para *poder disfrutar de las ventajas que ofrece*

Hacer registro en la aplicación no tiene demasiada dificultad. Se introducirán los datos en un formulario y se realiza la persistencia en una base de datos.

2. **Iniciar Sesión y Cerrar Sesión**: 2 puntos.

Como *usuario* quiero *iniciar sesión y cerrar sesión* para *tener mi cuenta personal*

Una vez realizado el registro, hacer login consistirá en rellenar un formulario de dos campos, nombre y contraseña. Una vez realizado el login con éxito, el usuario será identificado y redirigido a la vista principal. Si el login es incorrecto se mostrará un mensaje de error.

3. **Gestionar Datos**: 4 puntos

Como *usuario* quiero *poder gestionar mis productos o pedidos en cualquier momento* para *añadir, editar, mostrar o borrar los mismos*

El el sistema deberá disponer de una base de datos para poder almacenar productos o pedidos, un sistema persistente CRUD con los métodos necesarios para cada operación.

4. **Buscar vendedores de comida**: 5 puntos.

Como *usuario* quiero *buscar ofertantes de comida en un mapa* para *poder ver productos que ofrecen*

Se introducen puntos en un mapa de Google cogiendo las coordenadas de la dirección introducida en el registro de usuario. Para buscar ofertantes de comida en el mapa, en la vista debemos introducir en un input la dirección, código postal o ubicación que nos encontremos para situar el mapa alrededor de lo buscado, aparecerá un punto por cada usuario registrado, en diferentes colores dependiendo del tipo de usuario.

5. **Ver productos ofrecidos por ofertante**: 3 puntos.

Como *usuario* quiero *ver productos que ofrece cada ofertante* para *poder ver datos, valoración que le han dado otros usuarios al producto y comentarios.*

Para ver los productos que ofrece un ofertante, deberemos pinchar sobre el punto del usuario que elijamos, a continuación, en la ventana que aparece debemos pulsar sobre ver productos, una tabla debajo del mapa aparecerá con todos los productos, precios, descripción y valoración de cada uno de ellos.

6. **Añadir y quitar productos al carrito de la compra**: 2 puntos.

Como *usuario* quiero *añadir y quitar productos al carrito*

Añadir y quitar productos al carrito será implementado con Ajax, los productos se insertarán y borrarán de la lista dinámicamente.

7. **Ver estado del pedido**: 1 punto.

Como *usuario* quiero *ver estado del pedido* para *para poder estimar el tiempo para recibirlo*

Una vez realizado el pedido, el vendedor es el encargado de cambiar el estado del pedido a aceptado, mientras no sea aceptado el comprador tendrá Pendiente el estado de su pedido. Los estados se renderizarán dinámicamente en las vistas.

8. **Cambiar estado del pedido**: 2 puntos.

Como *usuario vendedor* quiero *cambiar estado del pedido* para *que el comprador aproxime el día de la entrega*

El vendedor cambiará el estado del pedido a aceptado y a continuación se enviará.Cuando el comprador reciba y haya probado el producto, cambiará el estado a entregado y podrá valorar y comentar el producto.

9. **Elegir método de pago**: 1 puntos.

Como *usuario* quiero *elegir método de pago* para *pagar mediante transferencia bancaria o contrareembolso*

En la ventana de registro y edición del perfil se podrá elegir el método de pago mediante un select.

10. **Contactar con otros usuarios**: 2 puntos.

Como *usuario* quiero *enviar mensajes privados* para poder *contactar con otros usuarios*

En la ventana que muestra los datos del usuario se podrá pulsar sobre contactar con el usuario y se podrá enviar un mensaje privado dentro de la aplicación, o ver el número de teléfono y llamar directamente.

11. **Valorar y comentar productos**: 2 puntos.

Como *usuario* quiero *valorar y comentar productos* para *colaborar con otros usuarios*

Una vez el vendedor haya aceptado el pedido, se podrá valorar cada producto con un rating y comentar para ayudar a los demás usuarios a decidir.

12. **Interfaz sencilla** : 3 puntos.

Como *usuario* quiero *disponer de una interfaz simple* para *poder navegar por la aplicación*

Aunque no sea una tarea muy complicada, es una tarea tediosa implementar todas las vistas que aparecerán en la aplicación, la interfaz deberá ser sencilla para que usuario aprenda en menos de 2 horas a utilizarlo.

13. **Actualización Asíncrona** : 1 puntos.

    Como *usuario* quiero *actualización asíncrona* para evitar tiempos innecesarios.

    Si en la parte servidor ocurre un evento se actualiza inmediatamente al cliente.

2.5.3. Iteraciones

En la primera iteración se realiza la elaboración de la memoria, análisis preliminar y análisis de costes estimada en **10 puntos**.

En la segunda iteración se realizarán las cinco primeras historias de usuario, según la estimación dada anteriormente la velocidad de equipo será de **16 puntos**.

En la tercera iteración se realizarán las historias de usuario comenzando por la 6 a la 13 inclusive por lo que la velocidad de equipo en esta tercera iteración será de **14 puntos**.

Las historias de usuario se reparten automáticamente en días quedando **20 días para la primera iteración**, **36 días para la segunda iteración** y **24 días para la tercera iteración**.

Total horas del proyecto: 4 h/dia de Lunes a Viernes dan un total de 80h al mes, durante 4 meses, el total de horas empleadas en el desarrollo del presente TFG es de 320h.

2.6. Estudio de viabilidad

En este punto analizaremos los costes del desarrollo, los sueldos del analista y el programador basados en el convenio colectivo de Telefónica publicado en BOE [12].

Una vez analizados los costes, gastos y sueldos podremos deducir el coste total del proyecto.

2.6.1. Análisis de costes de desarrollo y explotación

Se analizarán los gastos de material durante los 4 meses de duración del proyecto, incluyendo equipo necesario, conexión a internet, licencias de software utilizado y sueldos de miembros del equipo. Es decir, cualquier elemento que intervenga en la elaboración del proyecto.

2.6.1.1. Gastos de material

Para el equipo físico utilizado se estima su tiempo de vida útil, y se calcula el coste en el tiempo utilizado en proporción al tiempo utilizado.

Material Utilizado	Tiempo de vida útil	Tiempo Consumo Material	Coste Mensual	Coste Final
Imac 21' late 2012	5 años(60 meses)	4 meses	22,91	91,66
Fibra óptica de Jazztel	-	4 meses	37,50	150

Tabla 2. Tabla gastos de material

2.6.1.2. Licencias de Software

Software	Tipo de licencia	Coste Anual	Coste Mensual	Coste Final
NetBeans 8.1	CDDL y la GPL versión 2	0	0	0
MAMP	Open Source	0	0	0
StarUML	Open Source UML/MDA Platform	0	0	0
Office Home and Student 2016	Software de pago	149	12,41	49,66
Axure RP 8	Free Students and Teacher License	0	0	

Tabla 3. Tabla Licencias de Software

2.6.1.3. Sueldos de los miembros del proyecto

El mínimo establecido en el convenio colectivo de Telefónica a fecha de 21 de Enero de 2016 [12] para el analista y programador es de 26.088,30 y 20.941,34 respectivamente.

A continuación se hará un desglose por horas de cada rol atendiendo a diferentes tareas que deben acometerse.

Historia de Usuario	Programador	Analista
Registrarse en el sistema	25h	5h
Iniciar Sesión y Cerrar sesión	15h	5h
Gestionar datos	30h	10h
Buscar ofertantes de comida en un mapa	20h	10h
Ver productos ofrecidos por un vendedor	20h	5h
Añadir y quitar productos al Carrito de la compra	10h	5h

Ver estado del pedido	7h	2h
Cambiar estado del pedido	10h	2h
Elegir método de pago	5h	1h
Contactar con otros usuarios	15h	3h
Valorar y comentar productos	10h	2h
Interfaz sencilla	10h	2h
Actualización Asíncrona	10h	2h
Elaboración de memoria	-	50h
Análisis Preliminar	-	20h
Estimación de costes	-	10h

Tabla 4. Horas en cada tarea dedicadas por rol

Total de horas del analista: 134 horas

Total de horas del programador: 202 horas

Un año tiene 365 días de los cuales 148 son no laborables, ya que tenemos 52 sábados, 52 domingos, 16 festivos, 3 días libres variables y 25 días de vacaciones.

Se deducen 217 días laborables.

217 días por 8 horas no da un total de 1736 horas anuales.

A continuación se calcula el sueldo por hora para cada rol:

Para el analista: 26.088,30/ 1736h da un total de 15,02 Euros/hora.

Por lo que el sueldo que percibe el analista sería de 134 horas por 15,02Euros/hora = 2012,68 Euros.

Para el programador: 20.941,34 / 1736h da un total de 12,06 Euros/hora.

Por lo que el sueldo que percibe el programador durante este proyecto sería de 202 horas por 12 = 2436,12 Euros.

2.6.1.4. Coste Total del proyecto

Sumando los costes calculados anteriormente :

	Coste
Analista	2012,68
Programador	2436,12
Office Home Students	49,66
Imac 21’ late 2012	91,66
Fibra óptica Jazztel	150,00
	4740,12

Tabla 5. Coste total del proyecto

El coste total del proyecto es de **4740,12** Euros.

2.7. Modelo de dominio

Un modelo de dominio [3] se utiliza como fuente de inspiración para el diseño de los objetos software y muestra clases conceptuales significativas en un dominio del problema, es el artefacto más importante durante el análisis orientado a objetos.

Utilizando la notación UML, un modelo de dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación. Pueden mostrar:

- Objetos de dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

El modelo de dominio inicial es el siguiente:

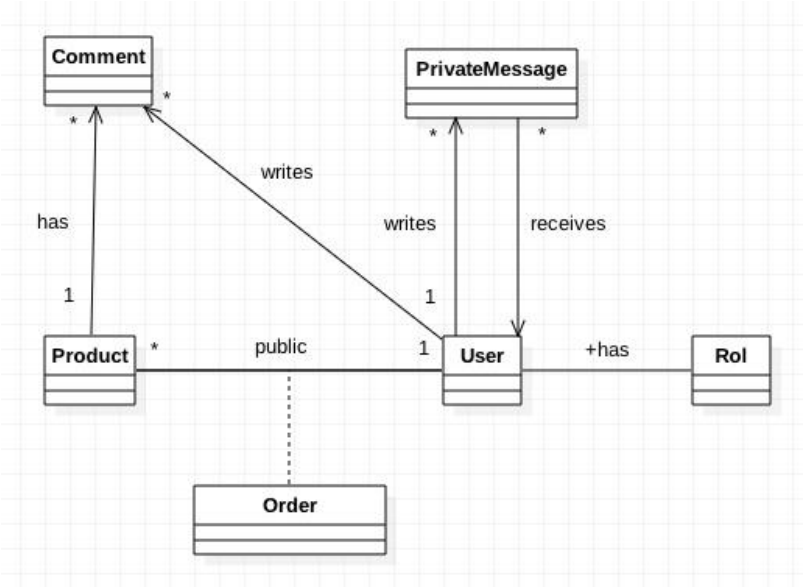

Ilustración 9. Modelo de dominio Inicial

Cada tipo de usuario tendrá un rol para determinar los permisos de acceso a diferentes vistas. Además se observa que los usuarios pueden publicar un número ilimitado de productos y que cada producto puede tener ilimitados comentarios escritos por diferentes usuarios.

Un usuario podrá añadir todos los productos que quiera del mismo vendedor para crear un pedido, también podemos observar que un usuario puede enviar ilimitados mensajes privados a otros usuarios, por lo tanto también puede recibir ilimitados mensajes privados de otros usuarios.

Para la realización del modelo de dominio se ha utilizado la herramienta para el diseño UML llamada StarUML[15].

3. CAPÍTULO 3: DISEÑO

En este capítulo se comentarán los aspectos relacionados con el diseño del sistema implementado, diagrama entidad-relación, diagramas de clases, diagramas de secuencia más relevantes del proyecto y el diseño de la interfaz explicando como se han colocado los diferentes elementos en las vistas, utilizando algunos diagramas de *layout* para reflejar ideas de este diseño.

3.1. Diagrama Entidad-Relación

El modelo entidad-relación [12] es una técnica de ingeniería de la información que se utiliza para desarrollar un modelo de datos. El modelo de datos ofrece una forma estándar de definir los datos y las relaciones entre éstos para todos los sistemas de información.

Una entidad se representa en forma de recuadro. El nombre se muestra en singular y con letras mayúsculas. El recuadro puede tener cualquier tamaño o forma, suficiente para contener un nombre sin ambigüedad y para hacer que el dibujo de una entidad-relación sea más conveniente.

Las formas normales [13] se corresponde a una teoría de normalización iniciada por el propio Codd y continuada por autores como Boyce y Fagin. Codd definió en

[15] "StarUML." 2014. 23 Jul. 2016 <http://staruml.io/>

1970 la primera forma normal, desde ese momento apareció la segunda, tercera, la cuarta y la quinta forma normal.

Una tabla puede encontrarse en primera forma normal y no en segunda forma normal, pero no al contrario. Es decir los números altos de formas normales son más

restrictivos (la quinta forma normal cumple todas las anteriores). La teoría de formas normales es una teoría absolutamente matemática.

Una tabla se encuentra en primera forma normal si impide que un atributo de una tupla pueda tomar más de un valor.

La segunda forma normal ocurre si una tabla está en primera forma normal y además cada atributo que no sea clave, depende de forma funcional completa respecto de cualquiera de las claves. Toda clave principal debe hacer dependientes al resto de atributos, si hay atributos que depende sólo de parte de la clave, entonces esa parte de la clave y esos atributos formarán otra tabla.

La tercera forma normal ocurre cuando una tabla está en 2FN y además ningún atributo que no sea clave depende transitivamente de las claves de la tabla. Es decir no ocurre cuando algún atributo depende funcionalmente de atributos que no son clave.

Para la persistencia de objetos se utilizará un sistema de base de datos relacional y para ello será necesario definir el diagrama entidad-relación para comprobar que está en 3ª forma normal, se utilizará un ORM, JPA, que es el encargado de materializar las entidades y relaciones sobre el motor relacional.

Como podemos comprobar el diagrama se encuentra en la 3FN, ya que no existen atributos que dependen de otros atributos que no sean claves, como por ejemplo *Order* se compone de una *id* que es clave primaria y otros atributos que son claves foráneas a *id_U*, *id_USaler* y a *id_Product*.

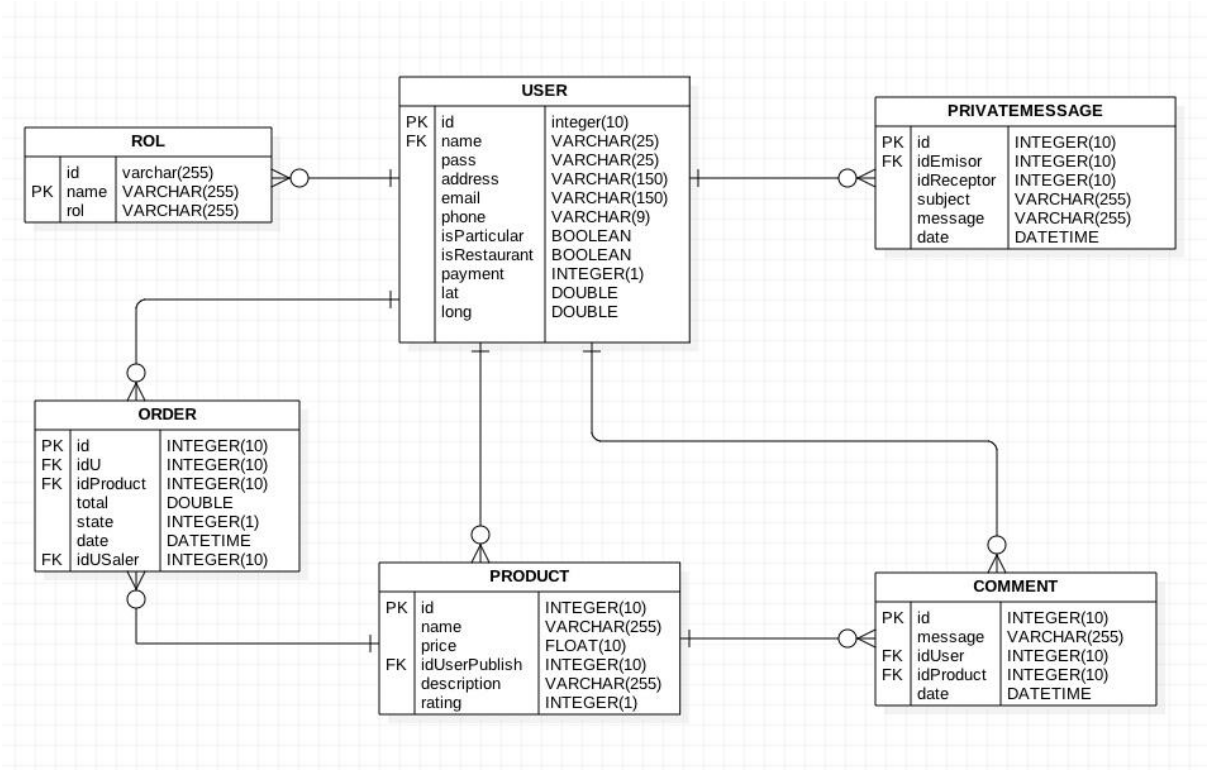

Ilustración 10. Diagrama Entidad-Relación

Las entidades persistentes son:

- ***User***: Un usuario podrá ser particular y/ó profesional, cada usuario guardará sus datos personales, el método de pago que utilizará y sus coordenadas en el mapa. La llave primaria de la entidad es la columna *id.*
- ***Rol***: Cada usuario podrá tener más de un rol, esta tabla está relacionada con la tabla de usuarios mediante el name de usuario.
- ***Product***: Cada producto será publicado por un usuario y éste a su vez tendrá varios comentarios de compradores que hayan valorado, también guardará la valoración media de todos los usuarios que lo hayan valorado. La llave primaria de la entidad es la columna *id* y la llave foránea es *uPublish_id* donde se guarda el usuario que lo publicó.

- ***Comment***: Cada producto tendrá varios comentarios. La llave primaria de la entidad es *id*, además, tiene dos claves foráneas una la *id* del usuario que comenta y la *id* del producto que ha sido comentado.
- ***PrivateMessage***: Cada usuario podrá recibir varios mensajes privados de otros usuarios.
- ***Order***: Tabla intermedia entre usuario y producto de muchos a muchos, necesita, al menos, un usuario y un producto. Muchos usuarios podrán realizar muchos pedidos, aunque cada pedido se hará solamente a un vendedor.

3.2. Diagrama de Clases

Las metodologías orientadas a objetos se enfocan en descubrir clases, atributos, métodos y relaciones entre las clases. Puesto que la programación se realiza al nivel de la clase, la definición de clases es una de las tareas más importantes en el análisis orientado a objetos. Los diagramas de clases [14] muestran características estáticas del sistema y no representan ningún procesamiento en particular. Un diagrama de clases también muestra las relaciones que existen entre ellas. Las clases se representan mediante rectángulos. En el formato más simple, el rectángulo podría incluir sólo el nombre de la clase, pero también podría incluir los atributos y métodos.

Los atributos son las características de los objetos, y los métodos u operaciones constituyen el cómo hacer las cosas. Los métodos son secciones pequeñas de código que trabajan con los atributos.

Un diagrama de clases podría mostrar el nombre de la clase, sus atributos y los métodos de esa clase. Mostrar sólo el nombre de la clase es útil cuando el diagrama

es muy complejo, si el diagrama es más sencillo, se podrían incluir atributos y métodos. Cuando se incluyen atributos, hay tres maneras de mostrar su información correspondiente. La más simple es incluir sólo el nombre del atributo, que toma la menor cantidad de espacio.

El tipo de datos, ya sea numérico, de tipo fecha, etc también se pueden incluir en el diagrama de clases.

El sistema se organiza según el patrón de diseño MVC (Modelo-Vista-Controlador) [15] , a lo largo de este apartado se verán las distintas partes de este patrón aplicados al proyecto.

El diseño de clases estará dividido en dos partes principales: la parte referente al cliente y la parte referente al servidor.

El diseño de clases en la parte del servidor se mostrará en formato apaisado para su mejor comprensión.

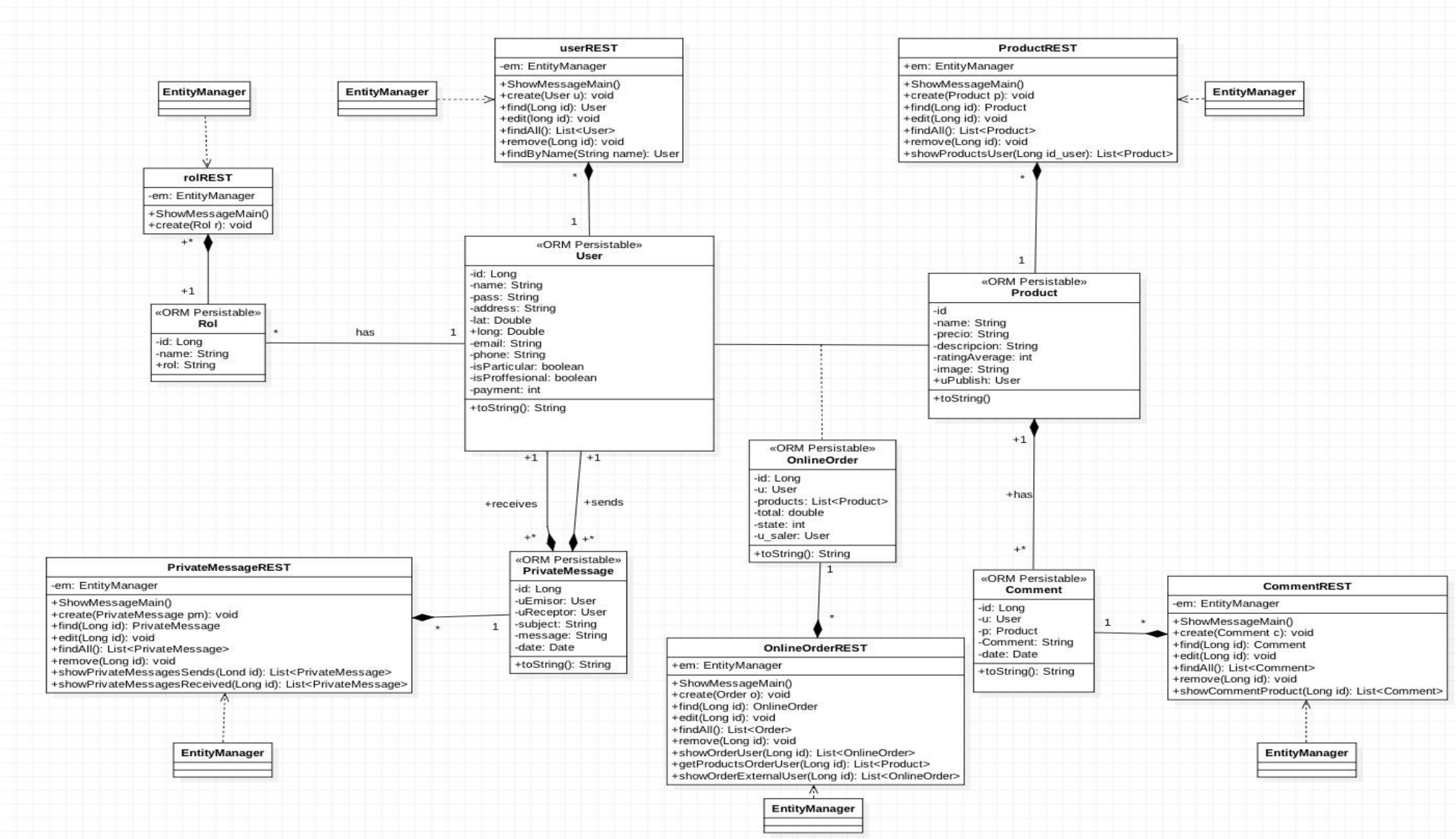

Ilustración 11. Diagrama de Clases parte Servidor

Las entidades persistentes son gestionadas por los servicios REST, los cuales intercambian las entidades del modelo de dominio entre los controladores y el motor de persistencia utilizando el objeto *entityManager* de Hibernate para esto.

Los productos serán valorados con un *rating* por los usuarios, guardado en un campo entero en la clase producto, cada ofertante tendrá una valoración definida por la media de la valoración de los productos que ofrece, este campo no se guarda dentro de la clase User, simplemente se realiza una consulta al controlador a la hora de mostrarlo.

Los servicios REST están implementados con JAX-RS *server api* y JPA para realizar la persistencia en la base de datos relacional.

El diseño de clases en la parte del cliente se mostrará en formato apaisado para su mejor comprensión.

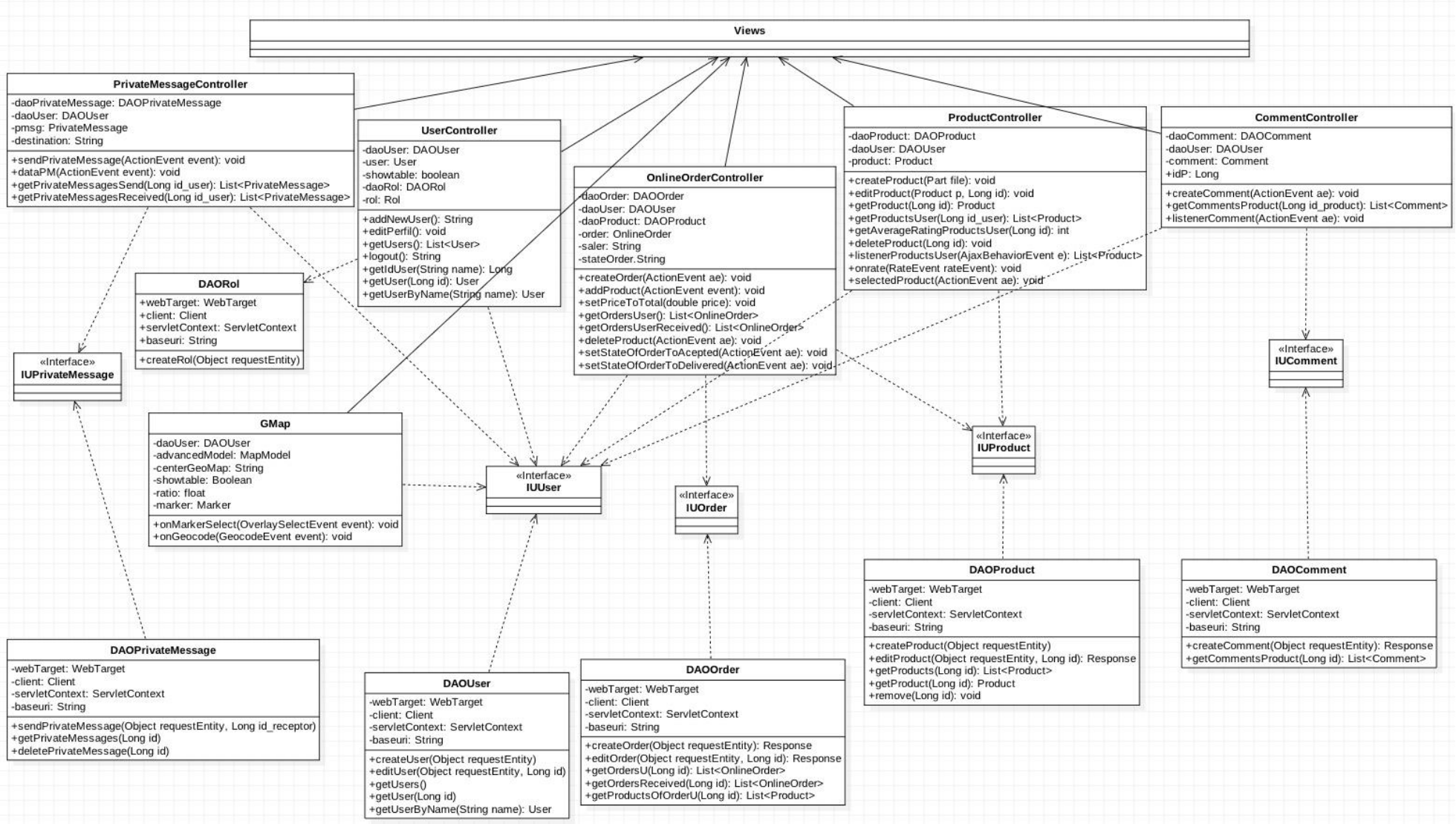

Ilustración 12. Diagrama de Clases Cliente

Todas las clases tienen constructor y sus métodos get/set aunque no se han incluido en el diagrama.

La vista [17] es la parte del patrón encargada representar de un modo visual los contenidos del modelo al usuario. Redibuja las partes necesarias cuando se produce una modificación del mismo teniendo relación de muchas a uno con el modelo. En este proyecto las vistas son archivos con extensión .xhtml[16].

El controlador [17] es el encargado de interpretar las instrucciones que realiza el usuario, realizando actuaciones sobre el modelo, actua tanto si la modificación se produce en una vista o en el modelo.

En *Ilustración 12* podemos observar como la vista hace petición al controlador, y este a la interfaz del DAO[17] donde se hace la petición al servicio REST remoto que ofrece el servidor.

Un DAO es definido en Wikipedia [18] (*Data Access Object*) como un componente software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos. Esta considerado como una buena práctica, teniendo la ventaja de usar objetos de acceso a datos sin requerir conocimiento directo del destino final de la información que manipula.

[16] eXtensible HyperText Markup Language
[17] Data Access Object

3.3. Diagramas de secuencia

Un diagrama de secuencia [16] se usa para mostrar las comunicaciones dinámicas entre objetos durante la ejecución de una tarea. Este tipo de diagrama muestra el orden temporal en el que los mensajes se envían entre los objetos para lograr dicha tarea.

El diagrama muestra los pasos involucrados, resaltando una figura en un dibujo. Por lo general, cada caja de la fila que hay en la parte superior del diagrama corresponde a un objeto, aunque es posible hacer que las cajas modelen otras cosas, como clases o vistas. La caja suele representar un objeto, un actor o una vista.

Debajo de cada caja hay una línea punteada llamada línea de vida del objeto. El eje vertical que hay en el diagrama de secuencia corresponde al tiempo, donde el tiempo aumenta conforme se avanza hacia abajo.

Muestra llamadas de método usando flechas horizontales desde el llamador hasta el llamado, etiquetado con el nombre del método y que opcionalmente incluye sus parámetros, sus tipos y el tipo de retorno.

A continuación se muestran los diagramas de secuencia más relevantes del proyecto:

- **Diagrama de secuencia Registrar de usuario.**

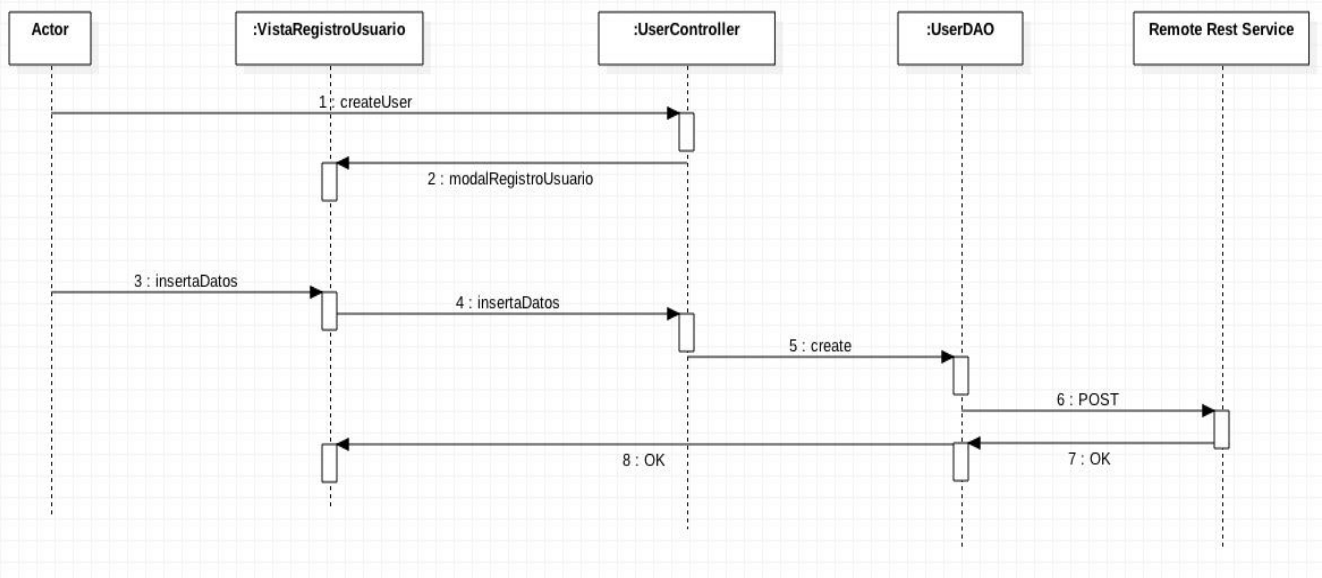

Ilustración 13. Diagrama de secuencia Registro de Usuario

El usuario pulsa sobre el enlace registrarse, esta llega al controlador de usuario y le muestra la ventana para registrarse, inserta los datos y el controlador se comunica con el *UserDAO* que a su vez manda la orden POST al servicio REST remoto para realizar la persistencia del objeto.

- **Diagrama de secuencia Login de usuario.**

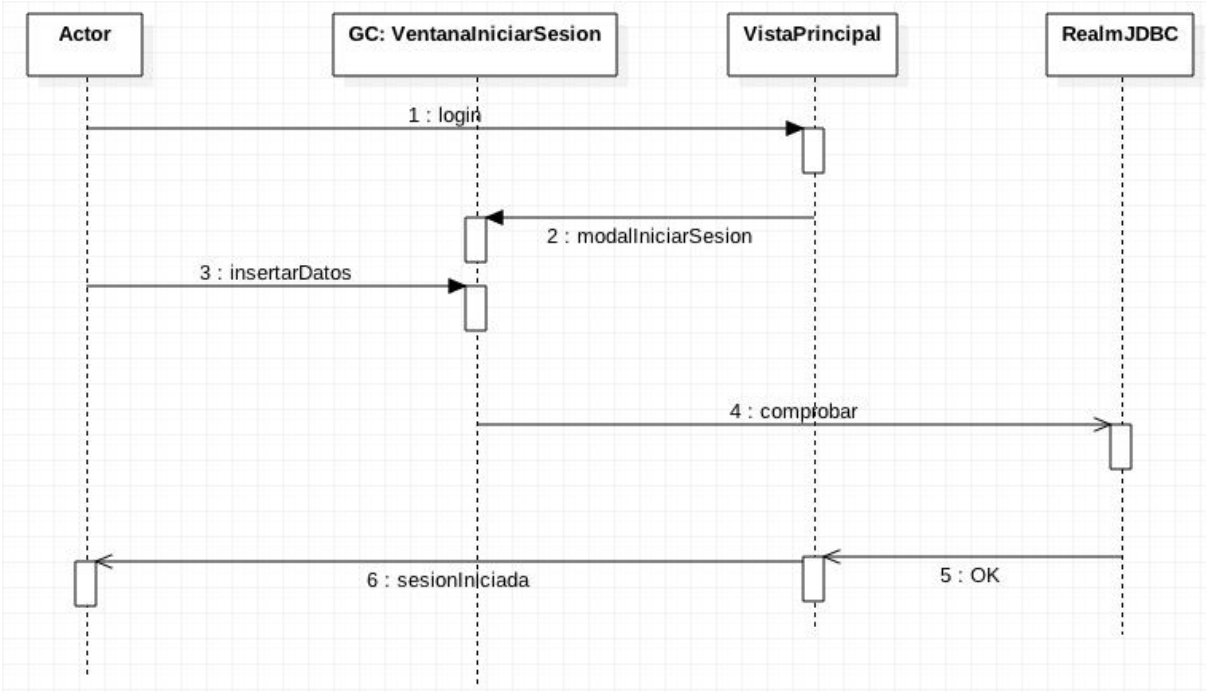

Ilustración 14 Diagrama de secuencia Iniciar Sesión

El usuario pulsa sobre iniciar sesión, a continuación en el formulario j_security_check de Java inserta *name* y *password*, este se comunica con el *JDBCRealm* configurado en *Glassfish server*, comprueba los datos en las tablas indicadas del *Realm* y decide si existe autentificación o no.

- **Diagrama de secuencia Ver productos que ofrece un vendedor.**

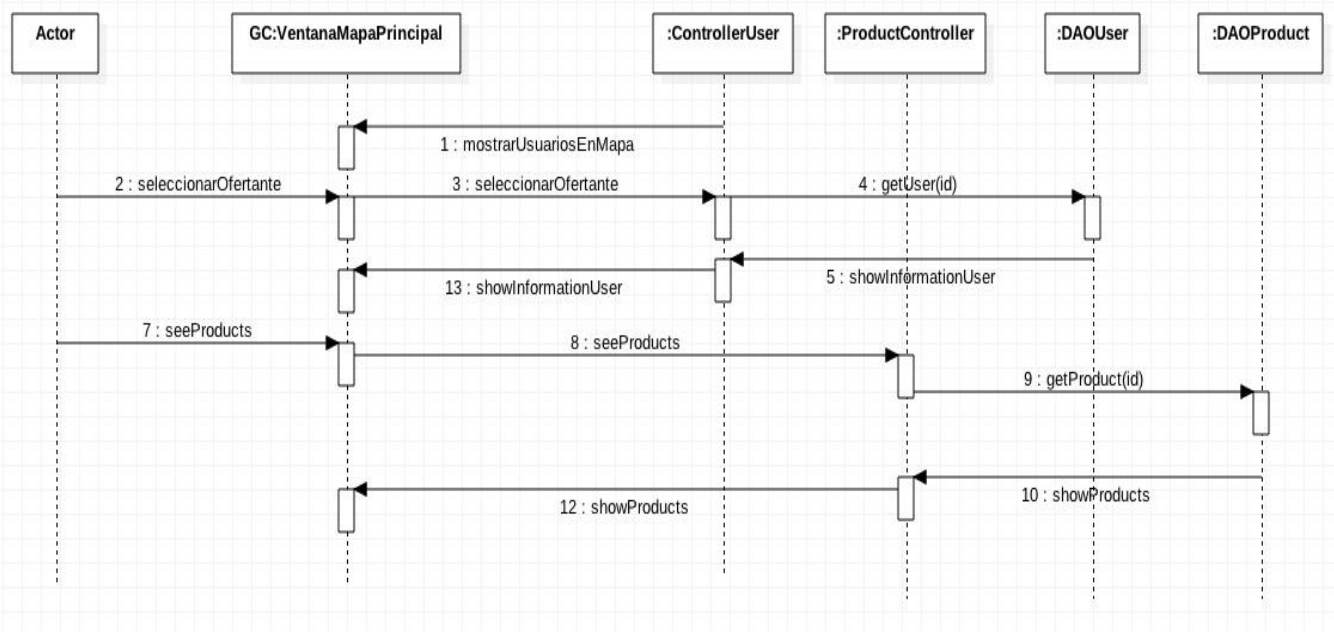

Ilustración 15. Diagrama de secuencia Ver productos vendedor

Para ver los productos que ofrece un usuario, primero se pulsa sobre la marca (usuario registrado) en el mapa, se comunica al controlador de usuario y te muestra una ventana modal con los datos del usuario y la ópción de *Ver que ofrece*. Pulsamos sobre el botón y el controlador de productos es el encargado de pedir a *DAOProduct* los productos que ofrece ese mismo usuario, para mostrarlos definitivamente en la vista.

3.4. Diseño de la Interfaz

El diseño de la interfaz de usuario [16] crea un medio de comunicación entre los seres humanos y la computadora, siguiendo un conjunto de principios de diseño de la interfaz, el diseño identifica los objetos y acciones de ésta y luego crea una plantilla de pantalla que constituye la base del prototipo de la interfaz de usuario.

Si el software es difícil de usar, fuerza al usuario a cometer errores, o si frustra sus esfuerzos para alcanzar las metas, entonces no le gustará, sin que importe el poder computacional que tenga, el contenido que entregue o las funciones que ofrezca.

La siguiente imagen corresponde al diseño de la interfaz de la vista index del proyecto.

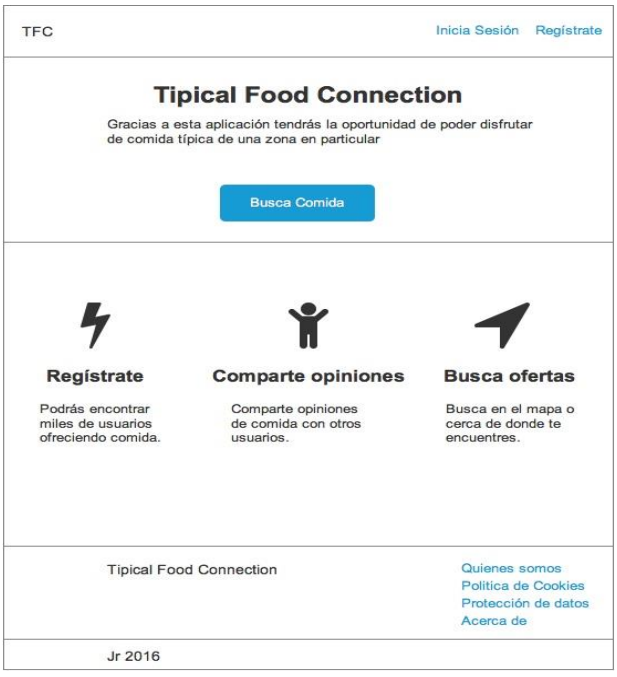

Ilustración 16. Vista *index* de la aplicación

La organización de la interfaz estará formada por:

- Barra de navegación, donde nos encontraremos en la parte izquierda el logo de la aplicación y en la parte derecha tendremos un link a la vista de registro y otro a una ventana modal para iniciar sesión si no hay sesión activa en el navegador. Si hay sesión activa aparecerá un botón con el texto de Bienvenid@ <tu_nombre>, dentro de él estará el link a mi Perfil, Mis productos, Mensajes Privados, Mis pedidos, Peticiones y Cerrar Sesión. Para dispositivos móviles, cambia el estado de la barra de navegación. En la parte superior izquierda ahora aparece un menú que contendrá el mismo contenido que contene el botón en navegadores web.
- Parte central donde se renderiza el contenido variable de las diferentes vistas.
- Pie (footer), donde podemos encontrar el nombre de la aplicación junto con el nombre del autor en la parte derecha, además de algunos link como Protección de datos ó Acerca de. Estas vistas no han sido implementadas, no redirigen a ninguna parte, es solo cuestión de estilo.

Se han implementado dos plantillas para evitar código repetido. Una contiene los tres puntos anteriores, y es utilizada en las vistas de Registro, Mi perfil, Mis Productos, Mis Pedidos y Peticiones. Esta plantilla no contiene carrito de la compra, quedando en la parte central del contenido un contenedor con un título informativo de la vista, debajo tendremos la ventana de ayuda y leyenda de la vista seguida de los datos consultados.

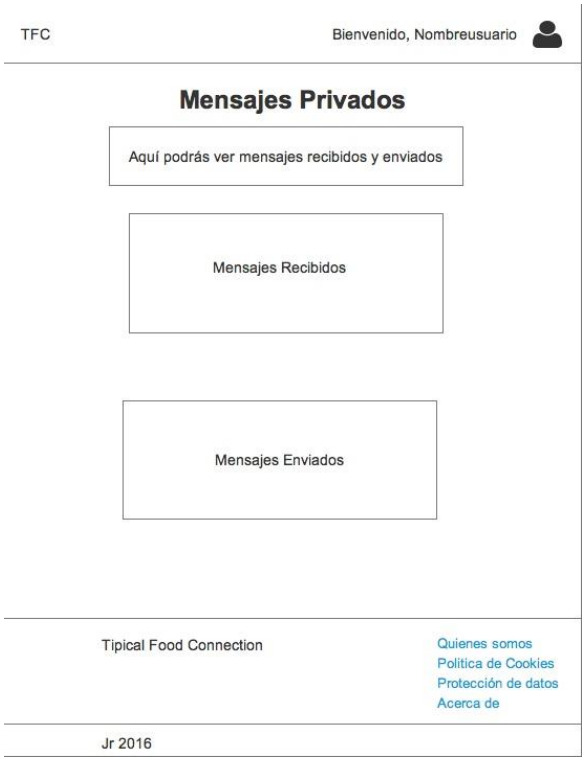

Ilustración 17. Ejemplo de vista con plantilla1

La otra plantilla es utilizada en la vista principal incluyendo además de los puntos anteriores el carrito de la compra en la parte superior derecha.

La vista principal dónde buscamos los usuarios, los productos que ofrecen y donde realizamos el pedido está organizada en dos partes principales, en la parte superior del contenido tenemos una ventana de ayuda y leyenda del mapa, justo debajo de esta se encuentra la tabla de usuarios registrados con datos de los mismos (nombre, email, teléfono, valoración y un botón para contactar con el usuario dentro de la aplicación), a la derecha se encuentra el carrito de la compra, el cual se renderiza automáticamente si se añaden o eliminan productos.

La segunda parte principal de la vista, situada en la parte inferior de la página está formada por:

- Un campo de entrada donde podremos buscar por ubicación para poner el punto de visión introducido en el mapa.
- Bajo el campo de entrada, tendremos el mapa de google maps donde se encuentran los usuarios registrados, los puntos en el mapa son de color azul si son particulares, de color amarillo si son profesionales, de color verde si el usuario es particular y profesional y de color rojo las ubicaciones que se buscan.
- En la parte derecha del mapa se renderizan los productos que ofrece cada vendedor, actualizando dinámicamente este panel cada vez que pulsamos sobre lo que ofrece cada usuario.

Los dos primeros puntos se agrupan en el mismo contenedor, y el tercero estaría en un contenedor diferente a su derecha.

Ilustración 18. Vista Pantalla Principal con plantilla2

Estos dos contenedores principales ocupan 6 columnas (m6) cada uno si nos encontramos en un navegador web ó 12 columnas (s12) si nos encontramos en dispositivos móviles.

Las tablas serán adaptativas a dispositivos móviles reorganizando automáticamente su contenido dependiendo de los píxeles de la pantalla donde estemos ejecutando la aplicación.

4. CAPÍTULO 4: Implementación

En este capítulo, en primer lugar, se expone la arquitectura en la que está basada el proyecto, explicando los elementos contituyentes de este, su función y como se comunican entre sí. En segundo lugar se nombran las funcionalidades más relevantes de cada tecnología junto con fragmentos de código para facilitar la comprensión de cada una de ellas.

4.1. Arquitectura

En este punto se describen las distintas partes en las que se divide la arquitectura de la aplicación. Entre ellas, la arquitectura entre cliente y servidor, cómo funciona el motor de persistencia y la conexión a la base de datos, las tecnologías para implementación de la aplicación utilizadas, indicando dónde se usan y cómo se interrelacionan entre sí.

Como ya se ha mencionado anteriormente, el sistema se divide en dos aplicaciones principales. Aunque las dos están unidas por un único proyecto, ambas están separadas en paquetes, respectivamente para la parte del cliente y la parte del servidor. La parte del modelo es común a las dos partes.

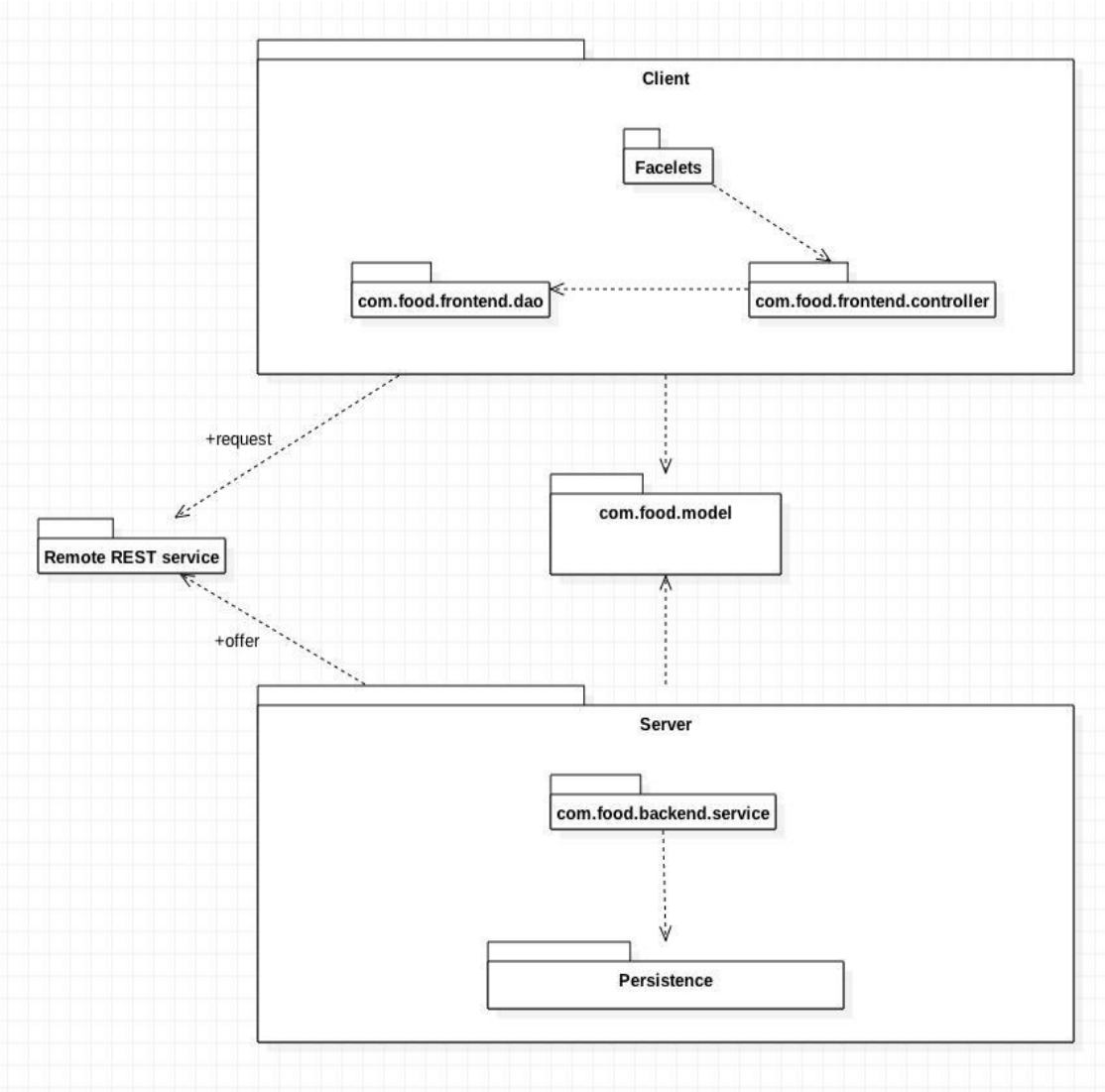

Ilustración 19. Arquitectura MVC cliente-servidor

En la parte del cliente, el actor se comunica con las vistas implementadas en JSF 2.2, éstas llaman a los controladores correspondientes que se comunican con los DAO's y éstos hacen una petición al servicio REST remoto que ofrece el servidor.

En la parte del servidor, los servicios web REST están implementados con JAX-RS server api y JPA[18] con Hibernate para la gestión de datos persistentes.

4.1.1. Tabla resumen de los servicios REST implementados en el proyecto

En este sub apartado se listan las tablas de los servicios REST implementados, por las columnas nombre del método, método HTTP utilizado por el microservicio y la URL accesible.

Los servicios REST parten de una URL principal, la URL principal de la aplicación REST.

- Servicio Web REST aplicación principal:

Función	Método HTTP	URL
Application REST	GET	/servicesREST

Tabla 6. Servicio Aplicación Principal

[18] Java Persistence API

- Servicio Web REST para la gestión de usuarios:

Función	Método HTTP	URL
create()	POST	users/{id}
findAll()	GET	users/all
find(Long id)	GET	users/{id}
remove(Long id)	DELETE	users/{id}
edit(Long id)	PUT	users/{id}
findByName(String name)	GET	users/findByName/{name}

Tabla 7. Servicio REST gestión de usuarios

- Servicio Web REST para la gestión de productos:

Función	Método HTTP	URL
create()	POST	products/{id}
showProductUser	GET	products/user/{id}
find(Long id)	GET	products/{id}
edit(Long id)	PUT	products/{id}
remove(Long id)	DELETE	products/{id}

Tabla 8. Servicio REST para gestión de productos

- Servicio Web REST para la gestión de pedidos:

Función	Método HTTP	URL
create()	POST	orders/{id}
findAll()	GET	orders/all
find(Long id)	GET	orders/{id}
edit(Long id)	PUT	orders/{id}
remove(Long id)	DELETE	orders/{id}
showOrderUser(Long id)	GET	orders/user/{id}
getProductsOrderUser(Long id)	GET	orders/products/{id}
showOrderExternalUser(Long id)	GET	Orders/received/{id}

Tabla 9. Servicio REST para gestión de pedidos

- Servicio Web REST para la gestión de comentarios:

Función	Método HTTP	URL
create()	POST	comments/{id}
findAll()	GET	comments/all
find(Long id)	GET	comments /{id}
edit(Long id)	PUT	comments /{id}
remove(Long id)	DELETE	products/{id}
showCommentProduct(Long id)	GET	Comments/product/{id_product}

Tabla 10. Servicio REST para la gestión de comentarios

4.2. Detalles sobre implementación

En este apartado se verán pequeños fragmentos del código desarrollado explicando cómo funcionan los diferentes Framework/tecnologías utilizadas a lo largo del proyecto. De esta forma se persigue facilitar la comprensión del funcionamiento interno de la aplicación y las posibles labores de mantenimiento y mejoras en un futuro.

4.2.1. Detalles de implementación en la parte del cliente

Las vistas son implementadas con JSF 2.2 junto con algunos complementos de PrimeFaces 5.3.

Ilustración 20. Logo Java Server Faces

Java Server Faces [17](JSF) es el marco estándar de interfaz de usuario orientada a componentes (UI) para la plataforma Java EE.

Fue formalizado como un estándar a través del Java Community Process, y forma parte de la plataforma Java, Enterprise Edition.

JSF 2 utiliza Facelets como su sistema de plantillas por defecto. Por el contrario, JSF 1.x utiliza Java Server Pages (JSP) como tal.

Sobre la base de un componente impulsado por la interfaz de usuario de diseño-modelo, JavaServer Faces utiliza archivos XML llamados plantillas de vista. El FacesServlet procesa las solicitudes, carga la plantilla de vista apropiada, construye un árbol de componentes, procesa eventos, y hace la respuesta (normalmente en lenguaje HTML) al cliente.

El estado de los componentes de interfaz de usuario y otros objetos de interés en el ámbito de aplicación se guarda al final de cada petición en un proceso llamado *stateSaving*, y restaurada en la próxima creación de ese punto de vista. Tanto el cliente como el servidor pueden guardar objetos y estados.

JSF se utiliza a menudo junto con el AJAX[19].

AJAX es una combinación de tecnologías que hacen posible la creación de interfaces de usuario y tuvo que ser añadido a través de JavaScript.

Debido a que JSF soporta múltiples formatos de salida, componentes habilitados para Ajax se pueden añadir fácilmente para enriquecer las interfaces de usuario basadas en JSF. La especificación JSF 2.0 proporciona soporte incorporado para AJAX mediante la estandarización de la solicitud del ciclo de vida, y proporciona interfaces de desarrollo simples para este tipo de eventos, permitiendo que cualquier evento desencadenado por el cliente para ir a través de una validación, conversión, y,

[19] https://es.wikipedia.org/wiki/AJAX

finalmente, la invocación de métodos, antes de devolver el resultado al navegador a través de una actualización del DOM[20].

En las dos capturas de pantalla siguientes de código implementado podemos observar como dentro del componente *h:commandButton* de JSF hacemos uso de AJAX, ejecutando la acción sin tener que recargar la página, y gracias al componente *<f:attribute>* recogemos el valor de dos atributos desde la vista en el controlador gracias al evento *ActionEvent*.

```
<h:commandButton  actionListener ="#{privateMessageCtrl.sendPrivateMessage}" styleClass="bt
    <f:ajax  execute="@form" />
    <f:attribute name="emisor" value="#{request.remoteUser}" />
    <f:attribute name="destinatario" value="#{privateMessageCtrl.destination}" />
</h:commandButton>
```

Ilustración 21. Ejemplo de uso de AJAX en vista.

```
public void sendPrivateMessage(ActionEvent event) {

    String emisor = (String) event.getComponent().getAttributes().get("emisor");
    User uEmisor = daoUser.getUserByName(emisor);
    Long id_emisor = uEmisor.getId();
    pmsg.setuEmisor(daoUser.getUser(id_emisor));

    String dest = (String) event.getComponent().getAttributes().get("destinatario");
    User uDest = daoUser.getUserByName(dest);
    Long id_dest = uDest.getId();
    pmsg.setuReceptor(daoUser.getUser(id_dest));

    try {

        daoPrivateMessage.create(pmsg);
        pmsg.setMessage("");
        FacesContext.getCurrentInstance().addMessage("envioCorrecto", new FacesMessage

    } catch (Exception ex) {
        FacesContext.getCurrentInstance().addMessage("envioIncorrecto", new FacesMessa

    }
}
```

Ilustración 22. Ejemplo de uso de AJAX en el controlador

[20] *Document Object Model*

También es interesante comentar la forma en la que se llama al controlador desde la vista:

```
action="#{userCtrl.addNewUser()}"/>
```

Ilustración 23. Llamada desde la vista al controlador JSF

De esta forma se llama al método *addNewUser()* del controlador userCtrl.

La forma en que damos nombre al controlador es gracias a la anotación *@Named("userCtrl")*.

```
/**
 *
 * @author juanramon
 */
@Named("userCtrl")
@ViewScoped
public class UserController implements Serializable {

    private static Logger log = Logger.getLogger(User.class.getName());

    @Inject
    private DAOUser daoUser;
    private User user;
```

Ilustración 24. Dando nombre al controlador con anotaciones

Ilustración 25. Logo Prime Faces

Prime Faces es una librería de componentes para JavaServer Faces de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web. Primefaces está bajo la licencia de Apache License V2. Una de las ventajas de utilizar Primefaces, es que permite la integración con otros componentes como por ejemplo RichFaces.

Entre sus propiedades destacan:

- Conjunto de componentes ricos (Editor de HTML, autocompletado, cartas, gráficas o paneles, entre otros).
- Soporte de ajax con despliegue parcial, lo que permite controlar qué componentes de la página actual se actualizarán y cuáles no.
- 25 temas prediseñados.
- Componente para desarrollar aplicaciones web para teléfonos móviles.

Los DAO's de este proyecto se implementan con JAX-RS 2.0 client api, como se dice en [18], Java EE 7 con JAX-RS 2.0 aporta varias características útiles, que simplifican aún más el desarrollo y conducen a la creación de aplicaciones con arquitectura RESTful para Java SE/EE aún más sofisticadas.

Como podemos observar en *Ilustracion 23. Implementación DAOUser,* utilizando la anotación *@Path("users"),* estamos definiendo la url principal del servicio, en este caso */users*.

Los métodos que estén declarados dentro de esta clase que también ofrezcan subservicios, la uri del método se concatena a la principal, como por ejemplo si hacemos uso del método *getUser()* la uri final para acceder al servicio sería *{baseURI}/users/1*, el cual nos devuelve el usuario con *id=1*.

```
@Dependent
@Path("users")
public class DAOUser implements Serializable, IUUsers {

    private WebTarget webTarget;
    private Client client;
    private ServletContext servletContext = (ServletContext) FacesContext.getCurrentInstance().getExternalContext().getContext();
    private String baseuri = getServletContext().getInitParameter("BaseUri");

    public DAOUser() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(baseuri).path("users");
    }
```

Ilustración 26. Implementación DAOUser

También cabe comentar que existen anotaciones para decirle al método que tipo de objeto va a recibir, para ello utilizamos la anotación @Consumes(MediaType.APPLICATION_JSON), por lo que este método espera este tipo de dato. Como ya se ha comentado en capítulos anteriores, la elección para este proyecto a sido objetos JSON, pudiendo haber optado por XML.

```
@Consumes(MediaType.APPLICATION_JSON)
public User getUser(Long id) {
    User u
            = webTarget
            .path("{id}").resolveTemplate("id", id)
            .request(MediaType.APPLICATION_JSON)
            .get(User.class);
    return u;
}
```

Ilustración 27. Método getUser() de DAOUser

También cabe comentar que se han utilizado algunos componentes para css del Framework MaterializeCSS, como por ejemplo las 'card' donde se muestran los productos.

Creado y diseñado por Google que combina principios clásicos con la innovación y la tecnología.

Ilustración 28. Logo MaterializeCSS [20]

4.2.2. Detalles de Implementación en la parte del servidor

En este apartado se hará mención del los Framework/tecnologías que influyen en la implementación de la parte del servidor.

En primer lugar se va a comentar el servidor Web que se a utilizado para el despliegue de la aplicación, este es llamado *GlassFish Server*, con licencia *OpenSource*, el cual está incluido dentro de NetBeans.

Ilustración 29. Logo *GlassFish Server*

GlassFish[21] es un servidor de aplicaciones de software libre desarrollado por *Sun Mycrosystems*, compañía adquirida por *Oracle Corporation*, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación.

Está basado en el código fuente donado por Sun y Oracle Corporation. GlassFish tiene como base el servidor *Sun Java System Application Server* de *Oracle Corporation*.

Ilustración 30. Logo Java EE

Los desarrolladores de hoy tienen cada vez más la necesidad de distribuir aplicaciones portátiles que aprovechen la velocidad, la seguridad y la fiabilidad de la tecnología en el lado del servidor.

[21] "GlassFish Server." 2006. 23 Jul. 2016 <https://glassfish.java.net/>

Las aplicaciones empresariales proporcionan la lógica de negocio para una empresa. Se gestionan de forma centralizada y, a menudo interactúan con otro software de la empresa. En el mundo de la tecnología de la información, las aplicaciones empresariales deben estar diseñados, construidos y producidos por menos dinero, con mayor velocidad, y menos recursos.

Con *Java Enterprise Edition*, el desarrollo de aplicaciones empresariales *Java* nunca ha sido más fácil y rápido. El objetivo de esta plataforma es proporcionar un potente conjunto de APIs y abreviar el tiempo de desarrollo, reduciendo la complejidad de la aplicación, y la mejora de rendimiento de las aplicaciones.

Se desarrolla a través del *Java Community Process* (JCP), que es responsable de todas las tecnologías Java. Los grupos de expertos integrados por las partes interesadas han creado *Java Specification Requests* (JSR) para definir las diversas tecnologías Java EE. El trabajo de la Comunidad Java bajo el programa JCP ayuda a asegurar los estándares de la tecnología Java, su estabilidad y compatibilidad entre plataformas.

Utiliza un modelo de programación simplificado. Un desarrollador puede simplemente introducir la información como una anotación directamente en un archivo de código fuente de Java, y el servidor Java EE configura el componente en la implementación y ejecución. Estas anotaciones se utilizan generalmente para ser empotrado en un programa de datos que de otro modo se suministra en un descriptor de despliegue. Con anotaciones, se pone la información de las especificaciones en el código junto al elemento del programa afectado.

La inyección de dependencia se puede aplicar a todos los recursos, ocultando efectivamente la creación y la búsqueda de recursos de código de la aplicación. La inyección de dependencia puede ser utilizada en envases de *Enterprise JavaBeans* (EJB), contenedores Web y clientes de la aplicación permitiendo además que el contenedor Java EE inserte automáticamente referencias a otros componentes o recursos requeridos, utilizando anotaciones.

La base de datos utilizada en este proyecto es MYSQL[22], muy conocida en el ámbito de las bases de datos relacionales.

Ilustración 31. Logo MYSQL

Es una de las bases de datos de código abierto más conocidas del mundo, puede ser utilizada como propiedad web de crecimiento acelerado, un proveedor de software o una amplia organización.

Puede ofrecer de manera rentable aplicaciones de base de datos escalables y de alto desempeño.

Para la persistencia se ha utilizado JPA [19], es uno de los APIs más conocidos y utilizados en el mundo JAVA EE, es una especificación de Java para acceder, persistir y administrar datos entre objetos Java y una base de datos relacional, en este caso MYSQL es considerado el enfoque estándar de la industria de Object to Relational Mapping (ORM) en la industria de Java.

JPA permite que las asignaciones objeto-relacional de un objeto se definan mediante anotaciones estándar, también define una API de EntityManager de ejecución para procesar consultas y transacciones a nivel de objeto, JPQL[23], permitiendo la consulta de objetos de la base de datos.

[22] https://www.mysql.com/
[23] *Java Persistence Query Language*

El servidor utiliza Hibernate para realizar la persistencia.

Ilustración 32. Logo Hibernate[24]

Hibernate permite a los desarrolladores escribir fácilmente aplicaciones cuyos datos sobreviven al proceso de solicitud. Utiliza la técnica Object Relational Mapping (ORM) a través de JDBC, siendo también una implementación de la especificación Java Persistence API (JPA).

No requiere de interfaces o clases base para las clases persistentes y permite a cualquier estructura de clases o datos a ser persistentes.

Soporta inicialización perezosa, no requiere ninguna tabla en base de datos o campos y genera gran parte del código SQL en el momento de inicialización en lugar de en tiempo de ejecución.

También ofrece consistentemente un rendimiento superior sobre el código JDBC directamente, tanto en términos de productividad como en el rendimiento en tiempo de ejecución.

En este proyecto, el objeto de tipo EntityManager que utilizará la unidad de persistencia se llama *TFGFoodPU* utilizando la anotación *@PersistenceContext*.

[24] http://hibernate.org/orm/

```
public class UserREST {

    @PersistenceContext(unitName = "TFGFoodPU")
    private EntityManager em;
    @Context
    private UriInfo uriInfo;
```

Ilustración 33. Unidad de Persistencia

Para acceder a los micro servicios REST implementados, se utiliza JAX-RS *server api*, a continuación se mostrarán alguno ejemplos de como funcionan los microservicios.

Como podemos observar en *Ilustración 30*, vemos varias anotaciones con las que se implementa el servicio. Por ejemplo, con @GET declaramos la acción HTTP que usará el micro servicio, con @Path se define la url accesible, que después el cliente utilizará, con @Produces/@Consumes le diremos al micro servicio el tipo de objeto que va a consumir o producir.

También podremos utilizar consultas SQL para consultar, insertar, editar y borrar objetos gracias al entityManger de Hibernate.

```
@GET
@Path("findByName/{name}")
@Produces(MediaType.APPLICATION_JSON)
public User findByName(@PathParam("name") String name) {
    User u = em.createQuery("SELECT u FROM User u WHERE u.name=?1", User.class).setParameter(1,name).getSingleResult();
    return u;
}

@DELETE
@Path("{id}")
public void remove(@PathParam("id") Long id) {
    getEntityManager().remove(id);
}
```

Ilustración 34. Micro servicios REST con JAX-RS server api

5. CAPÍTULO 5: Conclusiones

A partir de los resultados obtenidos y reflejados en la presente memoria, podemos concluir que se han alcanzado los objetivos inicialmente propuestos. En particular, se ha realizado un análisis de aplicaciones basadas en economía colaborativa, del sector de la gastromía, estadísticas en el hábito de la alimentación, etc, obteniendo puntos positivos y negativos los cuales ayudan en la decisión final de lo que se quiere desarrollar. Además se han obtenido requisitos para la aplicación, un desglose de costes con el que se ha deducido la viabilidad del proyecto.

Para el objetivo de diseñar un sistema informático especialmente orientado a la utilización de arquitecturas y metodologías de desarrollo web se ha cumplido completamente ya que la metodología de la aplicación está basada en el patrón de diseño modelo-vista-controlador, es de los más utilizados en el desarrollo web, utilizando código HTML, en los archivos *.xhtml*, junto con AJAX para hacer la aplicación dinámica.

También se ha propuesto una arquitectura cliente-servidor la cuál nos permitirá conectar diferentes aplicaciones cliente en un futuro.

El Framework utilizado para las vistas y comunicación con los controladores (JSF 2.2) me ha parecido fantástico, está basado en Java, un lenguaje que hemos utilizado en varias asignaturas del Grado, la facilidad que ofrece a la hora de gestionar eventos, comunicar las vistas con los controladores, simplemente dando nombre al controlador y accediendo desde la vista a sus atributos y métodos con *#{}*.

Para algunas funcionalidades, como para el mapa de *Google Maps*, Primefaces es muy interesante, fácil de añadir al código fuente, como un componente más de JSF.

La persistencia con Hibernate, se hace muy cómoda, pudiendo realizar consultas SQL desde el servicio REST del servidor definiendo una *query* con el objeto *EntityManager*.

Considero que las funcionalidades de la aplicación han quedado completadas, y que puede tener utilidad real entre el colectivo de usuarios objetivo, dando una publicidad adecuada y consiguiendo más usuarios registrados,por lo tanto más variedad de productos y así atraer a más usuarios.

El aspecto más relevante desde un punto de vista personal es haber aprendido a desarrollar una aplicación con servicios web REST basada en Java EE. Este tipo de aplicaciones están en auge en la actualidad y ha sido interesante poder profundizar en ellas, ya que pienso que es de gran utilidad en mi futuro profesional. La mayor parte de ofertas de empleo que he encontrado en los últimos meses son de este tipo.

Este proyecto me ha permitido aplicar varios conocimientos adquiridos en diferentes asignaturas del Grado, como pueden ser Programación orientada a objetos, Estructura de Datos, Desarrollo de aplicaciones web, Ingeniería del software ó Desarrollo de aplicaciones empresariales.

5.1. Mejoras y trabajos futuros

Entre las funcionalidades que se podría haber profundizado más, cabe comentar la inclusión de un sistema de pago como pay-pal, tpv virtual, etc . No obstante, para todas las propuestas sugeridas, considero que gracias al diseño planteado y tecnologías seleccionadas, su integración resultaría viable y cómoda disponiendo del tiempo adecuado.

También, gracias a la arquitectura REST propuesta en el diseño, se podría desarrollar una aplicación nativa para dispositivos móviles que accedan a los micro servicios ofrecidos por el servidor, por ejemplo con Ionic, que es un *Framework open source* para desarrollar aplicaciones híbridas multiplataforma que utiliza *HTML5*, *CSS* y *Cordova* como base.

Otra posible mejora sería incluir todo el texto en diferentes idiomas para abarcar usuarios de todo el mundo.

Además pienso que puede ser interesante adaptar la columna de el número de teléfono en la lista de usuarios, y desarrollar las funcionalidades en dispositivos móviles de contactar por Whatsapp[25] o llamar directamente.

[25] https://web.whatsapp.com/

Apéndice I. Manual de Instalación del sistema

En este apéndice se explican los pasos desde cero hasta instalar y ejecutar la aplicación completamente.

El entorno de desarrollo del proyecto es NetBeans IDE 8.1, para poder instalarlo necesitamos disponer en el equipo de Java JDK 1.8 que no es más que herramientas de desarrollo para Java. Una vez instalado, podremos instalar el entorno de desarrollo.

Antes de crear la aplicación, tenemos que crear la base de datos, para la generación de las tablas se encarga Hibernate automáticamente.

Antes de crear la base de datos, debemos instalar XAMPP(Windows), MAMP(Mac os x) ó LAMP(entornos Linux). Es un paquete de instalación independiente de plataforma, Open Source que consiste básicamente en el sistema de gestión de base de datos MYSQL, y servidor web Apache entre otras. Debemos tener estos servicios activados para trabajar con la aplicación.

Dentro de de el entorno de desarrollo, creamos la base de datos.

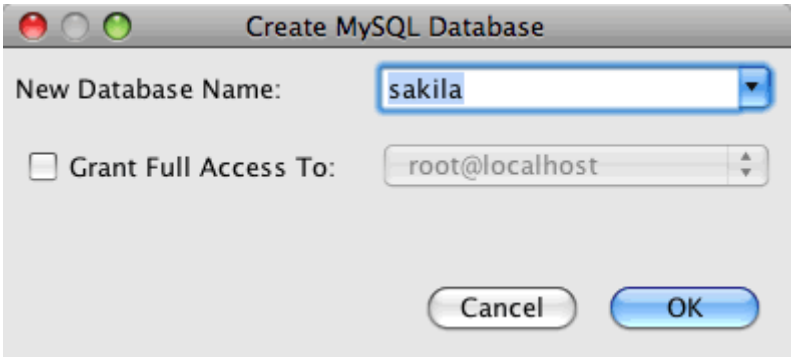

Ilustración 35. Creación de base de datos MYSQL

Una vez creada procedemos a crear la aplicación web, seleccionando los Framework que vamos a utilizar, como podemos observar en *Ilustración 35* seleccionamos JavaServer Faces e Hibernate y seleccionaremos la base de datos que hemos creado previamente.

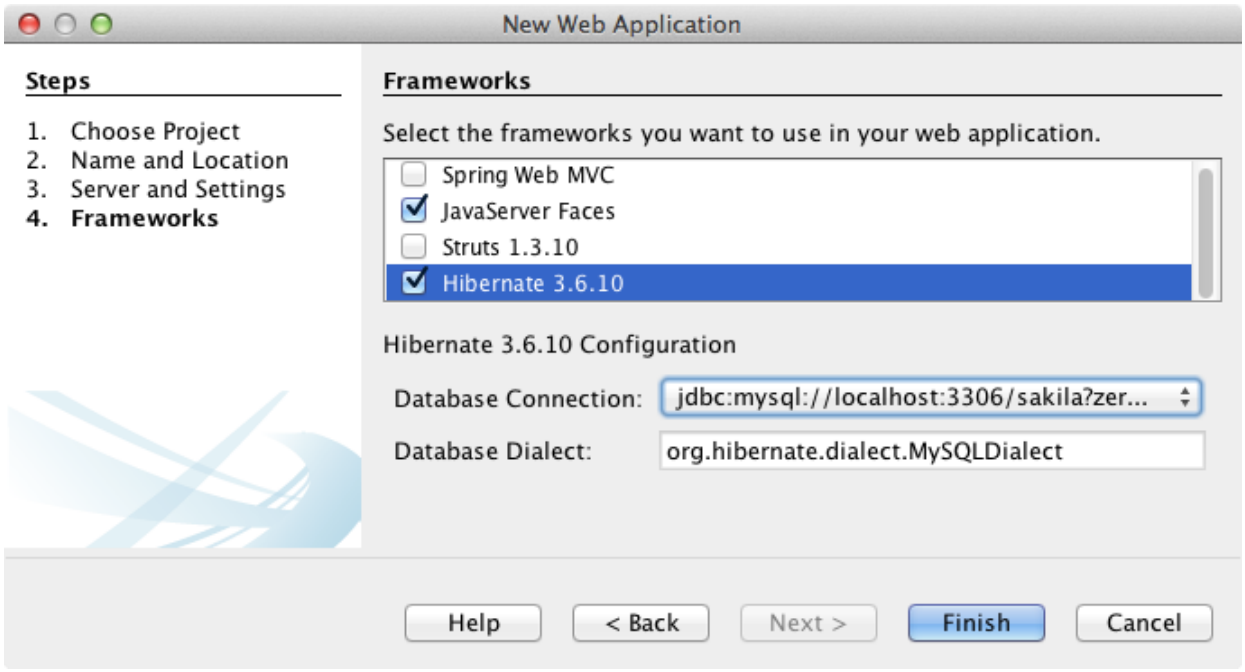

Ilustración 36. Selección de Framework al crear aplicación

Todas las librerías están incluidas dentro del proyecto, la mayoría son proporcionadas por NetBeans, aunque otras han tenido que ser incluidas del exterior.

Para crear una carpeta de librerías en el proyecto, dentro de NetBeans pulsamos botón derecho sobre nuestro proyecto Properties-->Libraries y en la parte superior derecha del input libraries folder pulsamos sobre Browser, veremos como nos detecta todas las librerías, incluidas las externas y nos copia todas a la misma carpeta ./lib.

Una vez realizado este paso, no tendremos que preocuparnos más por la ruta de las librerías, las nuevas que vayamos agregando también irán directas a este directorio.

Las librerias incluidas en el proyecto son:

- Hibernate 4.3.x(JPA2.1)
- Hibernate 4.3.x
- Mysql-connector-java-5.1.41-bin.jar
- Jandex-jandex-2.0.2.Final.jar
- PrimeFaces 5.3 – primefaces-5.3.jar
- JSF 2.2 – javax.faces.jar
- JSTL 1.2.1 –jstl-impl.jar

Una vez creado el proyecto aplicación web, tenemos que conectar la aplicación con el motor de persistencia, para ello lo primero que debemos crear es *JDBC Connection Pool*, lo llamaremos tfgfood, y la configuración es la siguiente:

- Nombre: tfgfood
- ResourceType: javax.sql.DataSource
- Datasource Classname: com.mysql.jdbc.jdbc2.optional.MysqlDataSource

Acto seguido, debemos crear un *JDBC resource* donde elegiremos el *Connection Pool* anteriormente creado, lo llamaremos *jdbc/tfgfood*.

Por último definimos un *jdbcRealm* mediante la consola de Glassfish para la autenticación y autorización de la aplicación.

La configuración del mismo se muestra en la siguiente imagen, como podemos ver utiliza el *connetionPool* anteriormente indicado jdbc/tfgfood para posteriormente definir las tablas de user y rol y sus columnas en las que buscará para hacer la autenticación.

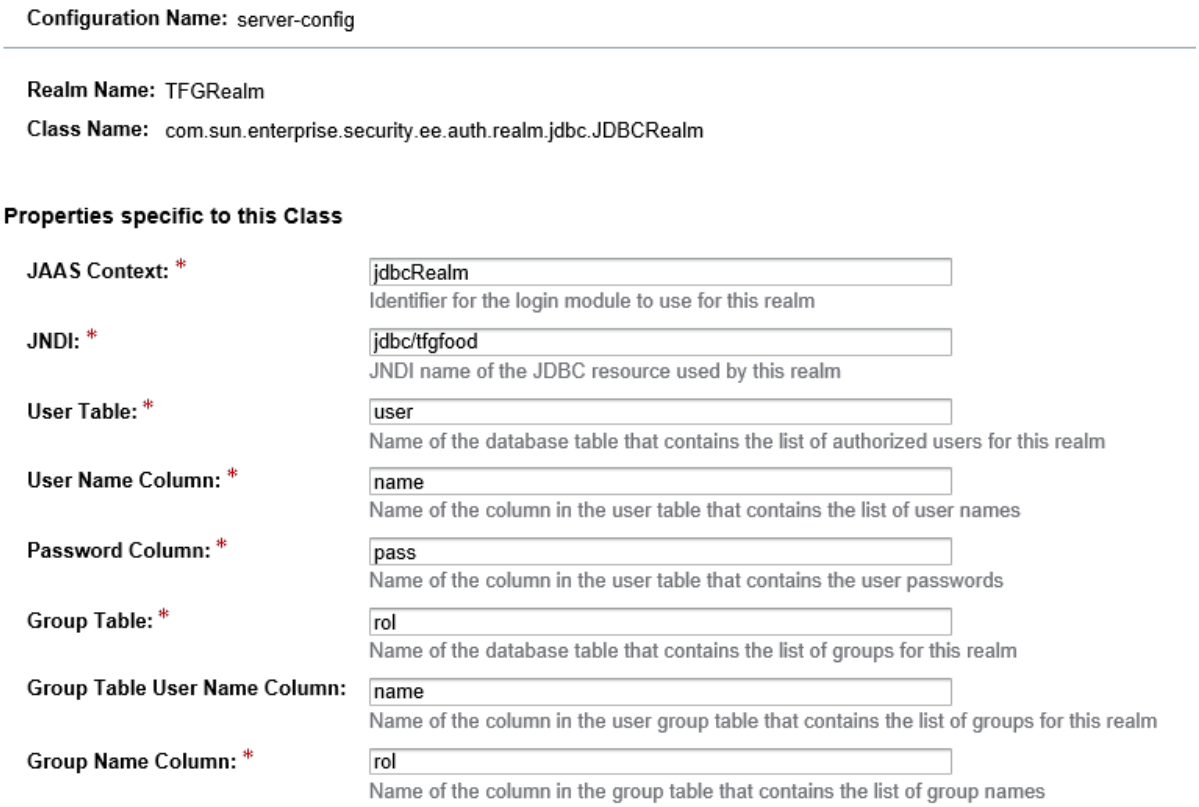

Ilustración 37 Creación de JDBCRealm

Apéndice II. Manual de Usuario

En este apéndice se explicará el flujo de la aplicación por las distintas vistas, un manual de usuario con capturas de pantalla para su mejor comprensión, desde que se ejecuta la aplicación hasta completar las funcionalidades más interesantes.

Tras ejecutar la aplicación nos encontramos en la página index.xhtml de la aplicación.

Ilustración 38. Vista inicial de la aplicación

En la barra de navegación, en la parte derecha tenemos los enlaces a iniciar sesión y registrarse, si no estamos registrados no se podrá acceder al dashboard donde se desarrolla las funciones principales de la aplicación, como ver los usuarios

registrados con rating, el mapa con los usuarios y sus productos ofrecidos, el carrito de la compra, enviar mensajes privados a los usuarios, etc.

Una vez registrados, y con la sesión iniciada, en la parte superior derecha dispondremos de un submenu, donde podemos encontrar las vistas Mi perfil, Mis productos, Mensajes Privados, Mis Pedidos, Mis peticiones y Cerrar Sesión.

En la opción Mis Productos, podremos añadir y ver los productos que ofrecemos. Al pulsar el botón añadir se abrirá una ventana modal para hacer la inserción del producto.

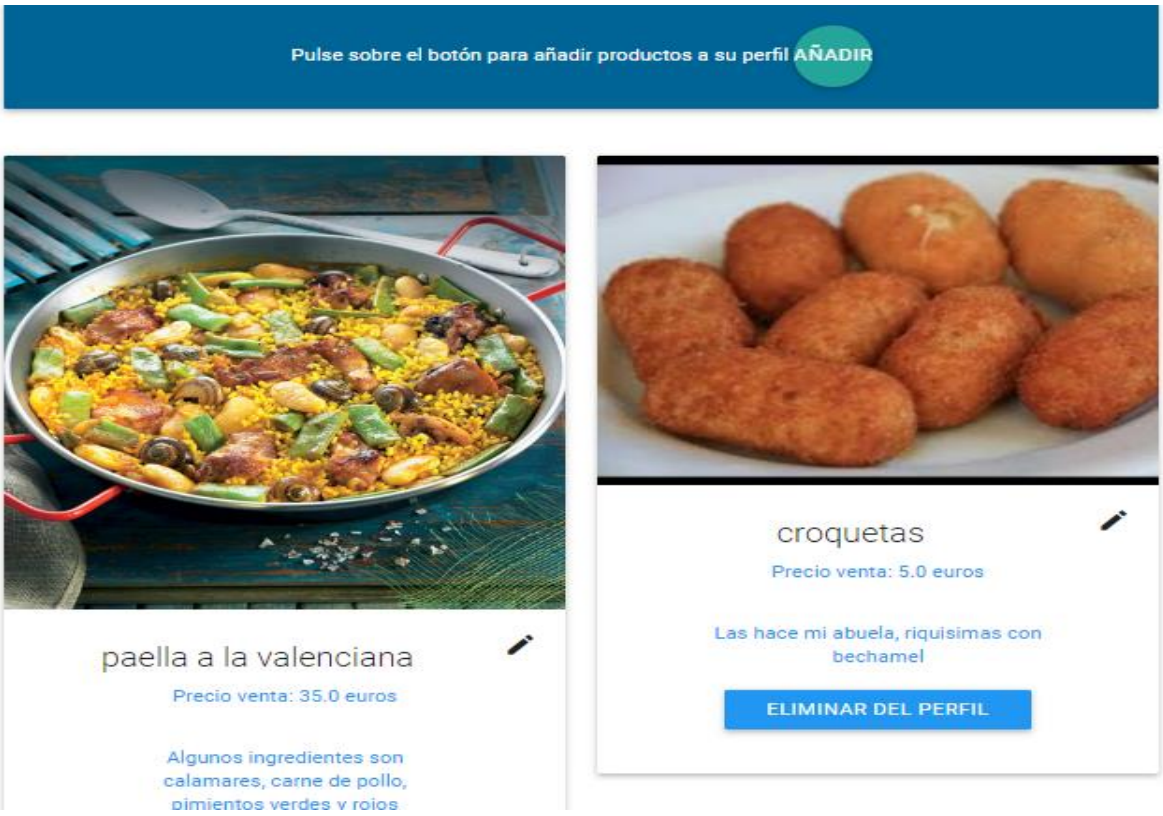

Ilustración 39. Vista Mis Productos

Para el envío de mensajes privados, debemos acceder a la vista principal de la aplicación, y en la tabla de usuarios registrados disponemos de una columna que pulsando sobre el botón aparecerá una modal para enviar el mensaje privado, una vez enviado se cierra la ventana modal.

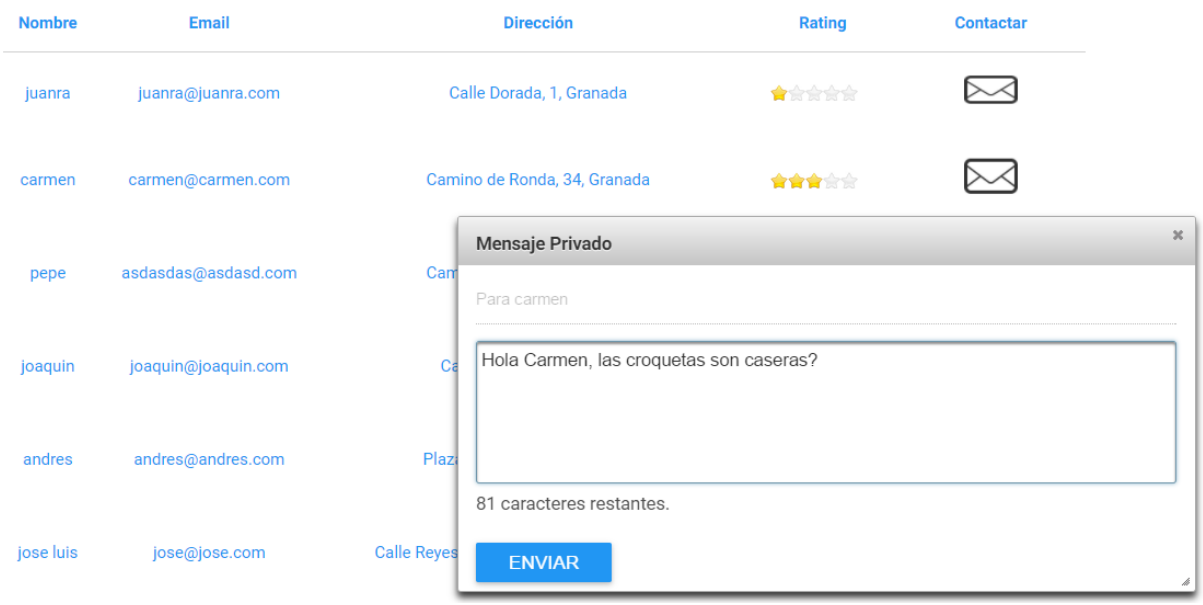

Ilustración 40. Enviar Mensaje Privado

En el sub menu de la barra de navegación tenemos el enlace a Mensajes Privados donde podremos ver los mensajes recibidos y mensajes enviados.

Para realizar nuestro pedido, lo primero que tendremos que hacer será buscar el usuario que queramos por el mapa, una vez elegido pulsamos sobre el punto en el

mapa, se nos abrirá una ventana con datos del usuario y pulsamos sobre el botón *Ver qué ofrece.*

Se nos desplegará dinámicamente un panel con los productos que ofrece, donde podremos ver el rating con la media calculada por el voto de los compradores, podremos pedir el producto y ver comentarios que otros usuarios han escrito.

Ilustración 41. Ver productos que ofrece un usuario

Una vez añadidos productos al carrito deberemos confirmar el pedido.

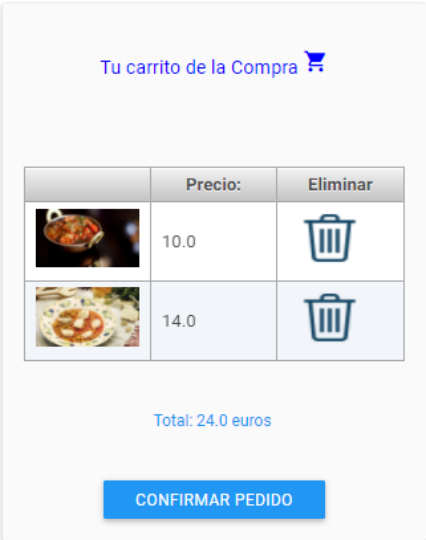

Ilustración 42. Carrito de la compra

Una vez confirmado el pedido, si entramos en la vista Mis Pedidos, tendremos lo siguiente:

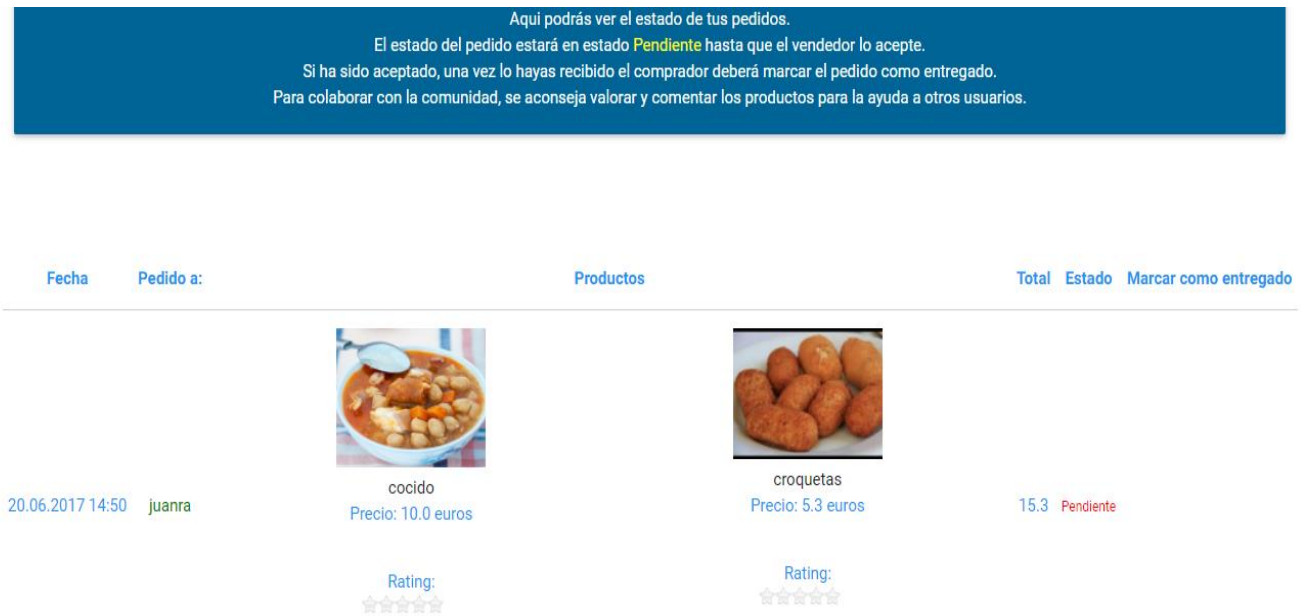

Ilustración 43. Pedido en estado pendiente

Como podemos observar, el pedido se encuentra en estado pendiente hasta que el vendedor lo acepte.

Si entramos desde la cuenta del vendedor, en la vista de Peticiones podremos aceptar el pedido.

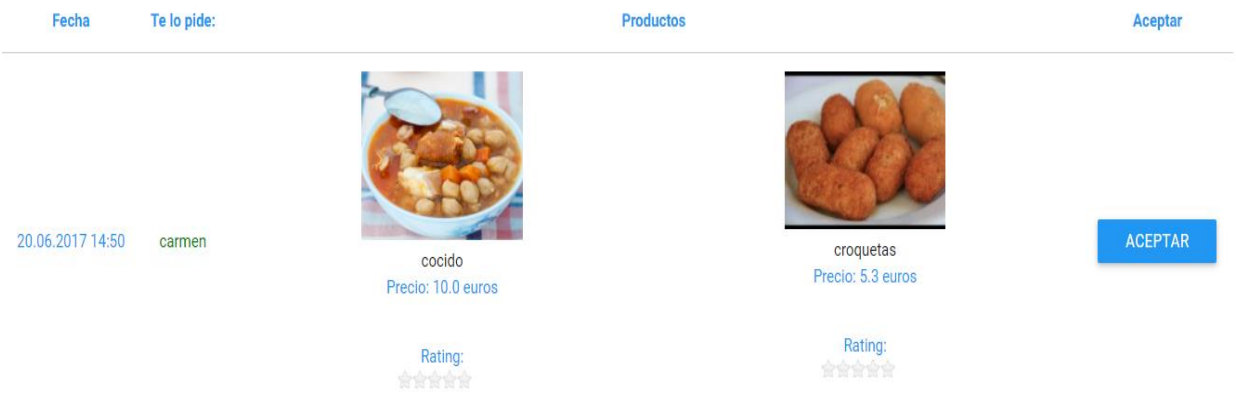

Ilustración 44. Aceptar Petición

Si volvemos a entrar con la cuenta del comprador, veremos que que la vista ha cambiado y el pedido se encuentra en estado aceptado, una vez lo hayamos recibido y probado, lo marcamos como entregado y podremos valorar con un rating y comentar cada producto comprado.

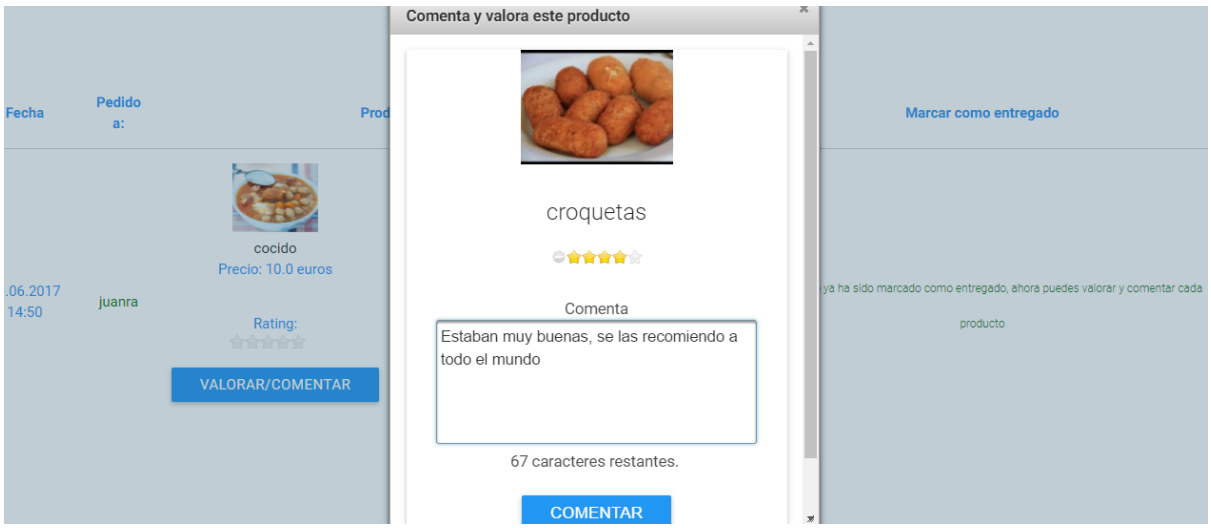

Ilustración 45. Valorar y comentar un producto

Si colaboramos votando y comentando productos, ayudamos a los demás usuarios que aún no probaron esa receta a que decidan mejor a la hora de hacer pedidos.

Como podemos ver en *Ilustración 46*, podemos leer comentarios de los productos que nos apetezca.

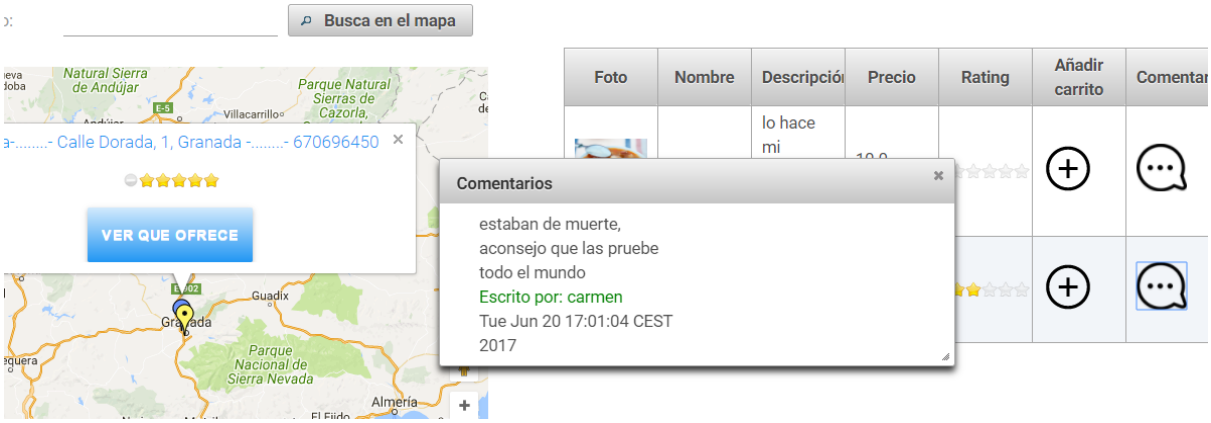

Ilustración 46. Ver comentarios

Apéndice III. Descripción de contenidos suministrados

En este apéndice se describen los contenidos suministrados en la entrega del presente TFG.

Un Pen-drive USB con los siguientes contenidos:

- Memoria en formato PDF.
- Código Fuente del proyecto en formato Netbeans.
- Código Fuente del proyecto comprimido en un archivo .zip llamado TFGFood.zip
- Fichero .war de la aplicación compilada para su despliegue.
- Carpeta llamada UML, donde se encontrarán los archivos con extensión *.mdj* creados por StarUML con todos los diagramas de la parte de Ingeniería del software del proyecto y en formato *.jpeg*.
- Carpeta llamada DI, dónde se encontrará el archivo con extensión *.rp* conteniendo el diseño de la interfaz del proyecto realizado con Axure RP 8 en el que se incluyen las diferentes vistas y los archivos *.jpeg* de los mismos.

Las librerías de terceros necesarias para el proyecto se encuentran dentro de la propia carpeta del proyecto.

6. Bibliografía

[1] A. Cañigueral, Los retos de la economía colaborativa. Dossieres Economistas Sin Fronteras, 2014.

[2] M. G. Software, «Scrum Methodology and Project Management,» 2014. [En línea]. Available: https://www.mountaingoatsoftware.com/agile/scrum. [Último acceso: 18 Julio 2016].

[3] C. Larman, UML y Patrones, 2ª ed., Pearson.

[4] I. K. /. J. M. Torrego, «El auge de la economía colaborativa en España, evolución de un sector en crecimiento,» 2015. [En línea]. Available: http://www.elreferente.es/tecnologicos/directorio-plataformas-economia-colaborativa-espana-28955. [Último acceso: 25 Octubre 2015].

[5] C. O. d. I. d. Telecomunicación, «Informe sobre Economía Colaborativa - Grupo de Políticas Públicas y Regulación».

[6] G. Laino, «Estudio de mercado comida a domicilio en españa,» [En línea]. Available: http://es.slideshare.net/GerardoLaino/estudio-de-mercadocomida-a-domicilio-en-espaa-2015. [Último acceso: 18 Julio 2016].

[7] E. pais, «economia.elpais.com,» 2015. [En línea]. Available: http://economia.elpais.com/economia/2015/08/10/actualidad/1439231577_419842.html. [Último acceso: 23 Abril 2017].

[8] J. L. B. W. L. M. Bill Dudney, Mastering JavaServer Faces, Burns ed., Indiana: Wiley Publishing, Inc, 2004.

[9] D. Kumar, «Best Practices for Building RESTful Web services,» InfoSys, 2015.

[10] «Requerimientos funcionales y no funcionales.,» 2014. [En línea]. Available: https://es.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales. [Último acceso: 24 Julio 2016].

[11] M. d. S. Manager, «SCRUM Manager v2.6.Guía de formación,» 2016.

[12] D. g. d. empleo, «Boletín oficial del Estado. Núm 18,» 2016.

[13] R. Barker, El modelo entidad-relación CASE* methodtm, Díaz de Santos ed., 1994.

[14] J. Sánchez, Principios sobre base de datos relacionales, 1ra ed., Creative Commons, 1994.

[15] K. E. K. a. E. J. KENDALL, Análisis y diseño de sistemas, K&K, 2011.

[16] Y. González, «Patrón Modelo-Vista-Controlador,» 2012. [En línea]. Available: http://revistatelematica.cujae.edu.cu/index.php/tele/article/viewFile/15/10.

[17] J. d. Andalucía, «Marco de Desarrollo de La Junta de Andalucía,» [En línea]. Available: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122. [Último acceso: 10 Junio 2017].

[18] Wikipedia, «Data Access Object,» [En línea]. Available: https://es.wikipedia.org/wiki/Data_Access_Object. [Último acceso: 03 Junio 2017].

[19] R. S.Pressman, Ingeniería del Software Un enfoque práctico, McGrawHill.

[20] «JavaServer Faces.org,» 2005. [En línea]. Available: http://www.javaserverfaces.org/. [Último acceso: 23 Julio 2016].

[21] «Java EE 7 y JAX-RS 2.0,» [En línea]. Available: http://www.oracle.com/technetwork/es/articles/java/java-ee7-y-jax-rs-2-2108434-esa.html.

[22] G. Inc, «Materilize Documentation,» 2014. [En línea]. Available: http://materilizecss.com/. [Último acceso: 23 Julio 2016].

[23] «Java Persistence,» [En línea]. Available: https://en.wikibooks.org/wiki/Java_Persistence.