5-3-2019

|  |
| --- |
| Proyecto Software | Ebrozon Development Team |



|  |  |
| --- | --- |
| EBROZON | PLAN DE GESTIÓN, ANÁLISIS, DISEÑO Y MEMORIA DEL PROYECTO. |

Contenido

[1. INTRODUCCIÓN 2](#_Toc2704424)

[2. ORGANIZACIÓN DEL PROYECTO 3](#_Toc2704425)

[3. PLAN DE GESTIÓN DEL PROYECTO 4](#_Toc2704426)

[3.1. PROCESOS 4](#_Toc2704427)

[3.1.1. PROCESOS DE INICIO DEL PROYECTO 4](#_Toc2704428)

[3.1.2. PROCESOS DE EJECUCIÓN Y CONTROL DEL PROYECTO 4](#_Toc2704429)

[3.1.3. PROCESOS TÉCNICOS 5](#_Toc2704430)

[3.2. PLANES 5](#_Toc2704431)

[3.2.1. PLAN DE GESTIÓN DE CONFIGURACIONES 5](#_Toc2704432)

[3.2.2. PLAN DE CONSTRUCCIÓN Y DESPLIEGUE DEL SOFTWARE 6](#_Toc2704433)

[3.2.3. PLAN DE ASEGURAMIENTO DE LA CALIDAD 7](#_Toc2704434)

[3.2.4. CALENDARIO DEL PROYECTO Y DIVISION DEL TRABAJO 7](#_Toc2704435)

[4. ANÁLISIS Y DISEÑO DEL SISTEMA 9](#_Toc2704436)

[4.1. ANÁLISIS DE REQUISITOS 9](#_Toc2704437)

[4.2. DISEÑO DEL SISTEMA 11](#_Toc2704438)

[5. MEMORIA DEL PROYECTO 11](#_Toc2704439)

[5.1. INICIO DEL PROYECTO 11](#_Toc2704440)

[5.2. EJECUCIÓN Y CONTROL DEL PROYECTO 11](#_Toc2704441)

[5.3. CIERRE DEL PROYECTO 11](#_Toc2704442)

[6. CONCLUSIONES 11](#_Toc2704443)

[ANEXO I. GLOSARIO 11](#_Toc2704444)

[ANEXO II. ACTAS DE TODAS LAS REUNIONES REALIZADAS 11](#_Toc2704445)

[ANEXO III. OTROS ANEXOS QUE SE CONSIDEREN NECESARIOS 11](#_Toc2704446)

# 1. INTRODUCCIÓN

El proyecto sobre el que trata este documento consiste en la creación de un sistema de compra-venta de objetos a través de una aplicación web.

El sistema permite publicar anuncios con objetos que se deseen vender, estos productos en venta son mostrados por el sistema y pueden ser comprados mediante el precio fijado por el vendedor o bien se puede enviar una petición de compra para llegar a un acuerdo entre las partes interesadas. Además, existe la posibilidad de que el vendedor pueda poner el producto a subasta y los posibles compradores pujen hasta que un único comprador se haga con dicho producto. El sistema dispone de filtrado para búsquedas para mostrar solo aquellos productos en los que se tenga interés, así como un sistema de geolocalización, el cual permite mostrar productos de vendedores cercanos. Por último, el sistema facilita la comunicación entre comprador y vendedor una vez llegado a un acuerdo sobre el precio de un producto para que la transacción sea ejecutada como deseen.

El propósito del desarrollo de este sistema descrito es la venta del mismo, por el precio acordado con el cliente y cumpliendo con los requisitos establecidos por el mismo, los cuales han sido brevemente descritos en el párrafo anterior.

El objetivo de este proyecto es la realización del sistema descrito partiendo desde cero, cumpliendo los requisitos acordados con el cliente al inicio del mismo, así como la realización de pruebas sobre el sistema final para verificar que se cumplen, además del correcto funcionamiento del sistema. También consta como objetivo la documentación de los pasos a seguir para realizar el objetivo anterior, los cuales quedan reflejados en este documento.

El sistema consta de dos tipos de clientes, un cliente web y un cliente móvil, este último desarrollado de forma nativa para Android, un único servidor y una única base de datos.

Se han acordado con el cliente cuatro entregas, la primera de ellas se llevará a cabo el 6 de marzo de 2019, en la cual se entregará un prototipo de la interfaz de usuario para que el cliente dé el visto bueno a una primera versión sobre la cual irá evolucionando, además de recibir posibles ideas o mejoras por su parte. La segunda entrega se llevará a cabo la segunda semana de abril donde se presentará una primera versión funcional del sistema con el objetivo de obtener retroalimentación por parte del cliente, con el fin de llevar a cabo posibles cambios o seguir por una determinada línea. La tercera entrega se llevará a cabo la tercera semana de mayo, en la que se entregará una versión muy desarrollada, pero no final, del sistema, con el fin de obtener posibles cambios de última hora y con el tiempo suficiente para que estos puedan llevarse a cabo. La entrega final ha sido acordada para final de mayo, en la cual se presentará al cliente la versión final del sistema. En las reuniones que impliquen la presentación de una versión del sistema, se hará entrega también de las fuentes que compongan cada versión.

En los siguientes apartados de este documento se describen la organización del mismo, miembros del equipo, sus roles y en que trabaja cada uno (apartado 2); como se ha construido el sistema, que procesos se han seguido, que herramientas se han utilizado, etc. (apartado 3); fase análisis y fase de diseño del sistema (apartado 4); memoria del proyecto (apartado 5); conclusiones obtenidas por el equipo de desarrollo una vez finalizado el proyecto (apartado 6).

# 2. ORGANIZACIÓN DEL PROYECTO

|  |  |  |
| --- | --- | --- |
| **Integrantes** | **Rol** | **Responsabilidades** |
| Saúl Alarcón Cano | Director del proyecto  Coordinador y desarrollador del grupo de back end | Realización de tareas de gestión (edición de memoria y otros documentos)  Responsable de redactar algunas actas en reuniones con el profesor  Control de la distribución de trabajo (elaboración de calendario) y revisión de esfuerzos  Encargado del diseño e implementación de la base de datos  Desarrollador de modelos, repositorios y controladores de la API  Encargado del despliegue del back end sobre Heroku |
| Alejandro Cano Somalo | Desarrollador del grupo de back end | Realización de tareas de gestión (edición de memoria y otros documentos)  Desarrollador de modelos, repositorios y controladores de la API |
| Andrés Gavín Murillo | Coordinador y desarrollador del grupo de front end de móvil | Realización de tareas de gestión (edición de memoria y otros documentos)  Control de la distribución de trabajo y coordinación dentro del grupo front end de móvil  Previsiblemente encargado de unificar las partes de la aplicación móvil y llevar a cabo el despliegue |
| Eduardo Gimeno Soriano | Desarrollador del grupo de back end | Realización de tareas de gestión (edición de memoria y otros documentos)  Desarrollador de modelos, repositorios y controladores de la API  Diseño del sistema por diagramas |
| Félix García Rodríguez | Desarrollador del grupo de front end de la aplicación web | Realización de tareas de gestión (edición de memoria y otros documentos)  Creación del diagrama E/R a usar en la base de datos  Diseñar pantallas de la aplicación de la web  Implementar la lógica de la aplicación de la web |
| Israel Solanas Navarro | Desarrollador del grupo de front end de la aplicación web | Realización de tareas de gestión (edición de memoria y otros documentos)  Diseñar pantallas de la aplicación de la web  Implementar la lógica de la aplicación de la web  Responsable de redactar algunas actas en reuniones con el profesor |
| Jorge Fernández Muñoz | Coordinador y desarrollador del grupo de front end de web | Realización de tareas de gestión (edición de memoria y otros documentos)  Diseñar pantallas de la aplicación de la web  Implementar la lógica de la aplicación de la web  Encargado de llevar a cabo el despliegue de la web |
| Sergio Álvarez Peiro | Desarrollador del grupo de front end de la aplicación móvil | Realización de tareas de gestión (edición de memoria y otros documentos)  Diseñar pantallas de la aplicación de Android  Implementar la lógica de la aplicación de Android |
| Víctor Sisqués Cortés | Desarrollador del grupo de front end de la aplicación móvil | Realización de tareas de gestión (edición de memoria y otros documentos)  Realización del diagrama de Gantt del proyecto y división del trabajo  Diseñar pantallas de la aplicación de Android  Implementar la lógica de la aplicación de Android |

# 3. PLAN DE GESTIÓN DEL PROYECTO

## 3.1. PROCESOS

### 3.1.1. PROCESOS DE INICIO DEL PROYECTO

Para el despliegue de la aplicación se va a hacer uso de Google Cloud, aunque posteriormente, con la entrega del software al cliente, este podrá desplegarlo en el servidor que prefiera utilizar sin ningún tipo de problema. En cuanto a las distintas aplicaciones, la aplicación web funcionará en los navegadores Chrome, Firefox y Opera, y la aplicación para Android podrá ser utilizada en la versión de Android 5.0 en adelante, siendo realizadas las pruebas sobre un Android 5.0.

Una vez definidas las tecnologías de las que se va a hacer uso para la realización de este proyecto, correrá a cargo de cada uno de los integrantes del grupo la tarea de formarse en las tecnologías que vayan a ser utilizadas en función del rol del mismo, apoyándose principalmente en los tutoriales y formación proporcionada por las paginas oficiales de las tecnologías a utilizar.

### 3.1.2. PROCESOS DE EJECUCIÓN Y CONTROL DEL PROYECTO

A la hora de distribuir las tareas del proyecto, principalmente se harán tres equipos:

Front-End en el que se encuentran Félix, Israel y Jorge, siendo este último el encargado del equipo.

Back-End en el que se encuentran Andrés, Alejandro y Saúl el encargado de este.

Android en el que se encuentran Sergio, Víctor y Eduardo, siendo este último el encargado del equipo.

La comunicación entre los miembros del grupo más informal se realiza a través de la aplicación de mensajería WhatsApp, a través del grupo Ebrozon (nombre de la aplicación). Además, existen 3 grupos diferentes en función del equipo de la aplicación Front-end, Back-end y Android.

Para el tema de las reuniones del grupo y paso de archivos de importancia media, se emplea la aplicación Discord, y se emplea un servidor con el nombre Ebrozon. Dentro de este se encuentran 4 canales de voz, uno para cada equipo de la aplicación, y un canal general en el que se realizan las reuniones entre todos los miembros del grupo o entre los líderes de cada apartado para realizar reuniones de carácter general o hacer puesta en común de los diferentes apartados. Además, existe un canal de texto para cada canal de voz y un canal de texto llamado archivos al que se suben archivos de importancia media, como notas informales de un acta y documentación semi-elaborada en forma de resumen de la documentación formal.

Por último, se emplea el repositorio GitHub, para trasferir los archivos propios de la aplicación, las actas, memorias, y otros ficheros clave en el desarrollo de la aplicación y entrega al cliente. Dentro de GitHub existen un repositorio para cada grupo y un repositorio de documentación que puede ser modificado por todos los miembros del equipo. Es importante destacar que en los repositorios destinados a cada equipo solo puede mergear en la rama master el encargado de dicho equipo.

En las actas de las reuniones se especificará la fecha, miembros presentes y objetivo.

Se realizará aproximadamente una reunión mensual entre todos los miembros del proyecto.

El proceso que se sigue en cada reunión es la siguiente:

1.-Puesta en común de los avances de los miembros del grupo, por ejemplo, coordinación de formatos entre cliente Android y cliente web, o adaptación de los códigos del cliente a las funcionalidades actuales del servidor.

2.-Revisión de las tareas pendientes en el futuro, bien sean del issue traker de GitHub, o la creación de nuevas, las cuales se subirán al issue tracker.

3.-Propuesta del objetivo de avance hasta la siguiente reunión y ver si coincide con el del calendario, sino proponer una modificación para este.

Además de en las reuniones de todos los miembros del grupo, las tareas pueden ser creadas en las reuniones de los encargados de cada grupo o en las reuniones de grupo. Estas son asignadas tras un diálogo entre los mismos, según su disponibilidad. En el caso de que los miembros del grupo no lleguen a un acuerdo el encargado de cada grupo decidido cómo se organizan las tareas.

Por lo general, los tres grupos han decidido hacer un reparto horizontal de las tareas, ocupándose cada uno de unos módulos de la base de datos para el equipo de Back-end o de diferentes pantallas para el equipo de Android o Front-end.

El reparto de tares de la memoria será diferente, pues algunos apartados son redactados por los miembros del grupo que tienen una mayor disponibilidad de forma voluntaria, y otros como el calendario solo pueden ser redactados por algunos usuarios (generalmente encargados del proyecto). Además, existen apartados más específicos para los que un miembro que ya haya comenzado a redactarlos está más preparado que el resto de miembros del equipo, por lo que preferiblemente ese miembro del equipo siempre es el encargado de ese apartado para ahorrar tiempo.

El proceso de validación de esfuerzos de los diferentes miembros del grupo se realiza una vez por semana y es el siguiente:

1.-Se revisan las horas dedicadas por cada uno de los desarrolladores, comprobando que se han dedicado al menos 5 horas.

2.-Se revisan las tareas realizadas por cada miembro del equipo.

3.-Se revisa el el número de commits de cada miembro.

4.-Evaluación entre pares sobre la repercusión del aporte de cada miembro del grupo al sistema.

La entrega de los resultados se hará mediante de la siguiente manera:

-Código fuente necesario para el despliegue el GitHub Ebrozón teniendo la última versión disponible en la rama Master de cada equipo Front-end los fuentes del cliente web, Back-end los fuentes del servidor y Android los fuentes del cliente móvil.

-Base de datos y aplicación web desplegadas en Heroku para mostrar el funcionamiento, y apk android.

En caso de disputa el primer encargado de solucionarla será el coordinador de cada subgrupo y si no será un consejo formado por todos los miembros del grupo quien lo decidirá.

### 3.1.3. PROCESOS TÉCNICOS

## Para el control de versiones del software desarrollado en todo el proyecto será git, mediante un repositorio público en GitHub.

## El desarrollo del software en la parte del Back-end el entorno Eclipse de Spring. En la parte del Front-end para el desarrollo de la web se trabajará con editores con soporte javascript y HTML como Atom y el entorno WebStorm. Para el desarrollo de la aplicación en android se usa el entorno de Android Studio haciendo uso de su integración de git para el control de versiones con el repositorio del Frontend de la aplicación.

## Para el despliegue local del software durante el desarrollo se usará Xampp debido al conocimiento de uso de esta y posteriormente se cambiará a la plataforma en la nube Heroku.

## 3.2. PLANES

### 3.2.1. PLAN DE GESTIÓN DE CONFIGURACIONES

La convención de nombres utilizadas para nombrar los distintos archivos sería la siguiente:

nombre.tipo.versión.revisión

Cada vez que se cree una nueva versión, pero sus cambios sean menores, como resolución de errores, se modificará su número de revisión, pero no de versión.

Las versiones solo se modificarán cada vez que se produzcan cambios suficientemente importantes, como por ejemplo la implementación de una nueva funcionalidad.

Para generar la documentación de los ficheros se emplearán herramientas contrastadas que la generen de forma automática a partir de los comentarios del código fuente (p.ej. javadoc en java).

Además, en los ficheros de documentación en los que se expliquen las diversas funcionalidades que tiene la aplicación y que errores se han ido resolviendo, cuando estos sean de una nueva versión o revisión solo se ofrecerá la información sobre los cambios que existan entre esta y la versión o revisión anterior, pero siempre que se cambie la versión se documentarán los cambios respecto a la primera revisión de la versión anterior (p.ej. La versión 2.8 solo contendrá las novedades respecto a la versión 2.7, pero la versión 3.0 contendrá todos los cambios que hayan sucedido desde la versión 2.0 aunque la mayoría se hayan documentado ya en las revisiones).

El estándar a utilizar en el código, puesto que va ser una aplicación web, se utilizará html5.

Los responsables de realizar la puesta en marcha se irían turnando. Una vez a la semana se elegirá un encargado que será el responsable de realizar la puesta en marcha de la versión más reciente producida.

Otras actividades como la creación de copias de seguridad y semejantes se realizarían de manera automática gracias a las prestaciones de GitHub.

El repositorio que se creará con todos los archivos referentes al proyecto se encontrará en GitHub, para que todos los integrantes del proyecto puedan acceder fácilmente a los archivos. Además, GitHub tiene integrado un gestor de incidencias, lo que hace que no sea necesario realizar estas gestiones mediante herramientas externas.

El proyecto estará dividido en varios repositorios. Uno para la documentación, otro de pruebas, otro para las versiones funcionales del proyecto y dos para el desarrollo del Front-End y el Back-End. Se intentará que no se modifique el mismo fichero por dos personas al mismo tiempo para evitar problemas.

En el repositorio de pruebas, se subirán los ficheros que hayan sido modificados y no se hayan vuelto a revisar si el comportamiento es correcto, además de los ficheros que realizan la pruebas.

Para que lo desarrollado en cada uno de estos repositorios pase al repositorio funcional, cada líder de las respectivas partes revisará el código actualizado y si todo está correcto se considerará válido.

Todos son capaces de modificar los ficheros de los repositorios excepto en el de las versiones, el cual solo podrán subir archivos y modificarlos los líderes del Front-End y el Back-End.

### 3.2.2. PLAN DE CONSTRUCCIÓN Y DESPLIEGUE DEL SOFTWARE

Durante el desarrollo, toda construcción y despliegue del back-end (servidor API y BD) para pruebas se va a llevará acabo con la aplicación XAMPP en su versión 3.2.2, compilada el 12 de noviembre de 2015, y Spring Tool Suite, el cual lanza un servidor Tomcat. Estas nos facilitan un despliegue local rápido y sencillo, que nos permite probar nuevas características de forma muy eficiente. Además, usando estas herramientas concretas, nos aseguramos de que todos trabajamos y realizamos pruebas sobre un sistema con unas características concretas iguales para todos. De XAMPP se usarán los módulos de Apache y MySQL para la base de datos, permitiendo que se conecten a los puertos que vienen configurados por defecto, siendo especialmente importante para el módulo de MySQL, ya que, si no se controlase, muy fácilmente podría producirse un problema de puertos a la hora de juntar las diversas partes.

Para el despliegue real del back-end, se realizará el despliegue del servidor y la base de datos sobre la plataforma Heroku, usando para la base de datos el add-on Heroku-Postgresql, ya que, es open source y era el que mayor almacenamiento gratuito proporcionaba. Las aplicaciones web y móvil se conectarán a este a la hora de realizar peticiones, tanto durante las pruebas como durante el funcionamiento real. Gracias a esto dispondrán de un servidor abierto constantemente para realizar sus pruebas sin depender del equipo del back-end, mientras el equipo de este continua el desarrollo de nuevas funcionalidades.

La elección de Heroku como plataforma de despliegue para el servidor se debe a su facilidad y rapidez de uso, al encargarse la propia plataforma del despliegue como tal tras haberle concretado algunos detalles como el lenguaje y el tipo de base de datos. Al despreocuparnos despliegue, podremos centrarnos en el desarrollo como tal.

Se procurará que 1 vez a la semana (ya que el back-end es simple de lanzar) se junten todos los cambios realizados en el back-end, para asentarla como una última versión, sobre la que se deberá trabajar la semana siguiente. Esta versión será montada por un encargado de despliegue.

El proceso de despliegue con XAMPP y Spring Tool Suite es el siguiente:

1. Lanzar la base de datos, para lo cual solo es necesario abrir XAMPP y darle a start a Apache y MySQL.
2. Aunar los ficheros fuente en un mismo proyecto Spring, lo cual supone solamente incluir los fuentes nuevos en los paquetes correspondientes.
3. Comprobar posibles errores de compilación, y corregirlos en caso de haber.
4. Darle a Run al proyecto de Spring Tool Suite, y en caso de mostrar errores durante el despliegue corregirlos, y darle a Run de nuevo.
5. Nota: en caso de tener el proyecto corriendo, si se hace alguna modificación en los fuentes, al hacer ctrl+s, obviamente se guardan, y se relanza automáticamente.

El proceso de despliegue sobre Heroku (una vez creada la BD sobre Postgresql) es el siguiente:

1. En la carpeta de nuestro PC donde esté el repositorio dedicado a Heroku, en el proyecto ya configurado, se actualizan los fuentes correspondientes a modelos, repositorios y controladores.
2. Se abre la terminal , se accede a dicha carpeta desde ella y se ejecuta “git add .”
3. Se ejecuta “git Commit –m mensaje”
4. Se ejecuta “git push heroku master”

Configuración para desarrollo en XAMPP y Sprint Tool Suite:

Ruta base de datos: mysql://localhost:3306/ebrozon\_bd

Puerto base de datos: 3306

Puertos apache: 80 y 443

Usuario base de datos: root (sin contraseña)

Ruta peticiones: localhost:8080

Configuración heroku:

Host base de datos: ec2-184-73-210-189.compute-1.amazonaws.com

Ruta base de datos:

postgres://ec2-184-73-210-189.compute-1.amazonaws.com:5432/d3311baks5knve

Puerto base de datos: 5432

Usuario base de datos: tvsujbnpveceag

Contraseña base de datos:

db96040cf34b037240cce6d42c971ee642aa8a5ee31206c43132a43de4411c3b

Ruta peticiones: protected-caverns-60859.herokuapp.com/

Respecto a la aplicación móvil solo nombrar que, obviamente, no necesita un despliegue, ya que se ejecuta localmente, y ejecutará sus peticiones sobre el servidor ya nombrado.

Y sobre la web, decir que una vez la web sea funcional y falten mejoras y correcciones menores, se realizará su despliegue sobre un servidor cloud, probablemente Google Cloud, que se configurará en dicho momento, y se realizarán los métodos de automatización correspondientes para facilitar el relanzamiento de la web, para poder realizar actualizaciones de forma eficiente. Este punto del desarrollo se espera que al menos unas 2 semanas antes de la entrega final, para tener tiempo de solucionar todos los problemas que puedan surgir al desplegarlo sobre un entorno real. Puesto que, en todo momento durante el desarrollo, se habrán estado comunicando con el servidor final, este punto no debería generar problemas más allá de complicaciones del despliegue de la web como tal.

### 3.2.3. PLAN DE ASEGURAMIENTO DE LA CALIDAD

Se seguirán las guías de estilo de cada herramienta utilizada, es decir, para Java https://google.github.io/styleguide/javaguide.html, para HTML https://google.github.io/styleguide/htmlcssguide.html, para Javascript https://google.github.io/styleguide/jsguide.html, para Spring https://github.com/spring-projects/spring-framework/wiki/Code-Style y para MySQL https://www.sqlstyle.guide/.

Para el diseño gráfico de las GUI se seguirá el estándar de Google Material Design https://material.io/design/ que simplifica el apartado gráfico para el usuario y se garantizará la usabilidad con unas acciones sencillas y una respuesta rápida de estas.

La documentación será realizada a través de herramientas de generación de documentación automática, en este caso Javadoc.

Para garantizar la calidad de código se seguirá una jerarquía de los miembros del grupo facilitada por las herramientas que incluye GitHub, entre las que se encuentran la división de los miembros en tres equipos (Back-End, Front-End y Android) con sus respectivas ramas de desarrollo. Además, la rama principal sólo podrá ser actualizada por el responsable de cada equipo, siendo la rama de desarrollo la única modificable por los demás miembros de cada equipo.

Opcionalmente, cada equipo podrá tener más sub-ramas de desarrollo, por ejemplo una para cada miembro del equipo, que serán actualizadas con cambios no siempre funcionales y cuando sean más estables se volcarán a la rama de desarrollo principal.

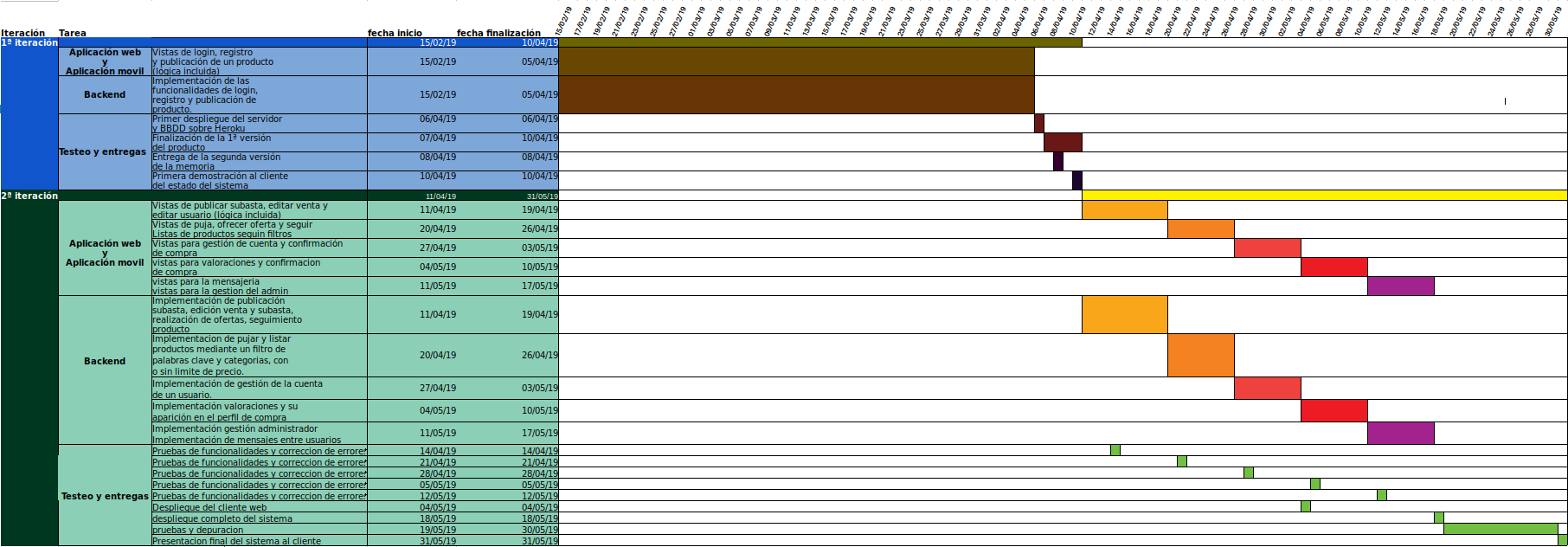
En la rama principal sólo podrá haber una versión funcional del sistema, que antes de ser subida será sometida a diferentes test automáticos, entre los que se incluirán test para comprobar la estabilidad del sistema (pruebas de sobrecarga) y test que revisarán las acciones disponibles para comprobar los requisitos que se han resuelto.

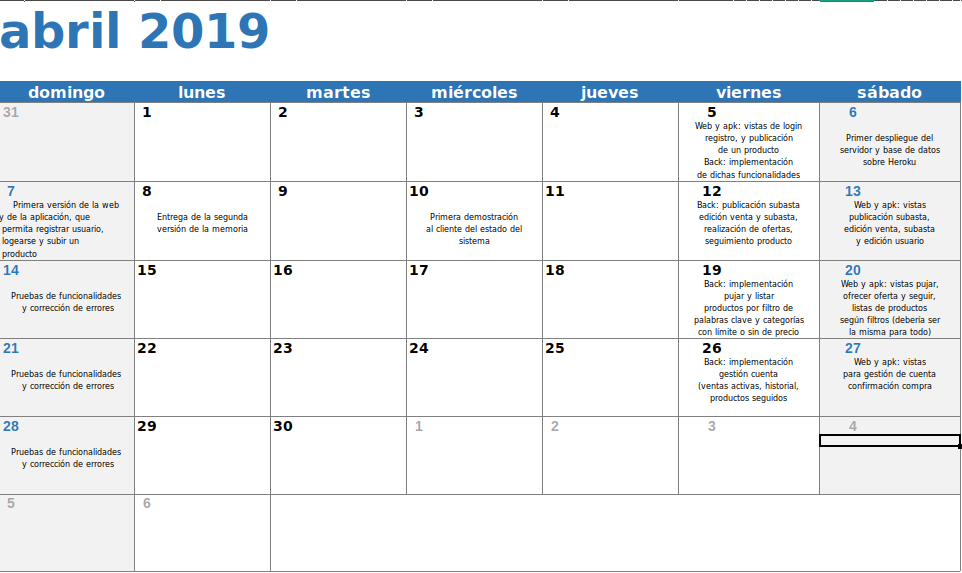
### 3.2.4. CALENDARIO DEL PROYECTO Y DIVISION DEL TRABAJO

En la primera iteración del proceso de diseño nos centraremos en desarrollar las funcionalidades principales del sistema, mientras que en la segunda iteración se corregirán todos los errores encontrados en la primera, se implementarán las funcionalidades secundarias y se afinara el diseño de la página web y de la aplicación Android para que sean más agradables al usuario.

Para la primera iteración se planea permitir la creación, edición y borrado de usuarios con sus credenciales básicos: nombre de usuario, nombre real, correo, contraseña y ciudad. También se permitirá la puesta en venta de objetos para compra inmediata y para su compra mediante subasta; los objetos se verán definidos por un nombre, foto, descripción, precio y etiquetas. Una etiqueta es una o más palabras clave que el sistema utilizará para clasificar los productos. También se permitirá la edición de la información de productos ya publicados por parte del cliente, además de otorgar la posibilidad al usuario de retirar estos. Adicionalmente, el sistema permitirá a los usuarios realizar ofertas por un producto y a los vendedores aceptar o rechazar estas. Finalmente, el sistema permitirá la búsqueda de productos mediante palabras clave.

Para la segunda iteración se finalizarán los requisitos que, por falta de tiempo, no pudieron ser completados en la primera y se añadirán funcionalidades al sistema. Estas funcionalidades son: permitir información adicional en los perfiles de usuario ( como puede ser una foto de perfil, una descripción, etc.), un sistema de calificación/reportes de usuarios tras una compra/venta, el bloqueo de una cuenta en caso de que tenga varias calificaciones negativas y/o reportes, el envio de mensajes entre usuarios, la posibilidad de añadir productos a listas de seguimiento y se mejorara el sistema de búsqueda para que se puedan filtrar los productos en función de localización, categoría, precio etc. Finalmente, el sistema registrara todas las acciones realizadas por los usuarios.





|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Saúl | Sergio | Eduardo | Félix | Jorge | Israel | Alejandro | Víctor | Andrés |
| Análisis de requisitos. | TODOS | | | | | | | | |
| Diagramas de análisis del sistema. | TODOS | | | | | | | | |
| Diagramas de diseño del sistema. | TODOS | | | | | | | | |
| Diseño e implementación del interfaz web. |  |  |  | ✓ | ✓ | ✓ |  |  |  |
| Diseño e implementación de la lógica del cliente web. |  |  |  | ✓ | ✓ | ✓ |  |  |  |
| Diseño e implementación del interfaz móvil. |  | ✓ |  |  |  |  |  | ✓ | ✓ |
| Diseño e implementación de la lógica del cliente móvil. |  | ✓ |  |  |  |  |  | ✓ | ✓ |
| Diseño e implementación de la base de datos. | ✓ |  |  |  |  |  |  |  |  |
| Diseño e implementación de la capa DAO. | ✓ |  | ✓ |  |  |  | ✓ |  |  |
| Pruebas de funcionalidad. | TODOS | | | | | | | | |
| Pruebas de mal uso. | TODOS | | | | | | | | |

Las tareas de la fase de requisitos, análisis y diseño se han repartido equitativamente entre todos los integrantes ya que todos hemos cursado la asignatura ingeniería software y debido a esto tenemos conocimientos básicos sobre esta fase del diseño de un sistema. Lo mismo ha sucedido con las pruebas. Durante esta primera iteración se realizarán 2 tipos de pruebas; pruebas de mal uso (pruebas en las que el usuario utiliza el sistema de manera incorrecta) y pruebas de funcionalidades (pruebas en las que se comprueba el cumplimiento de los requisitos).

Las personas asignadas a las tareas de diseño de la aplicación web (Diseño e implementación del interfaz web y diseño e implementación de la lógica del cliente), son aquellas con mayor experiencia a la hora de programar en HTML y/o en JavaScript.

Las personas asignadas a las tareas de diseño de la aplicación Android (Diseño e implementación del interfaz web y diseño e implementación de la lógica del cliente), son aquellas con mayor experiencia a la hora de programar en java y/o en XML.

Las personas asignadas a las tareas de Back-End (Diseño e implementación de la base de datos y Diseño e implementación de la capa DAO), son aquellas con mayor experiencia a la hora de programar en java y/o SQL. El trabajo se ha distribuido en base a la modularización del sistema en base a paquetes, y se han centrado en los más cruciales, siendo estos aquellos de los que más depende el sistema.

# 4. ANÁLISIS Y DISEÑO DEL SISTEMA

## 4.1. ANÁLISIS DE REQUISITOS

- El sistema debe permitir la existencia de usuarios.

- Un usuario debe tener un nombre y un correo electrónico.

- El sistema debe permitir que un usuario tenga una descripción y una foto.

- El sistema debe permitir que un usuario incluya una geolocalización aproximada, es decir, un radio (en kilómetros) en el que se encuentra.

- El sistema debe permitir que un usuario modifique su información.

- El sistema debe permitir un sistema de calificación de usuarios.

- El sistema debe permitir que un usuario califique a otro si ambos han participado en una compraventa.

- El sistema debe permitir el bloqueo de la cuenta de un usuario en caso de que tenga una calificación negativa.

- El sistema debe permitir a los usuarios poner en venta objetos, proporcionando información del producto y un precio de compra inmediata.

- La información de un producto se compone por nombre o título del producto, fotos, vídeos, una descripción y al menos una etiqueta.

- Una etiqueta será una o más palabras clave que se entenderán a modo de categoría, para poder clasificar los productos.

- El sistema permitirá crear etiquetas nuevas, fuera de las predefinidas, durante la creación de una venta.

- El sistema debe permitir a los usuarios poner en subasta objetos, siendo estas definidas por un precio mínimo, una fecha límite y un precio de compra inmediata, además de la información del producto.

- El sistema debe permitir al vendedor editar tanto la información como el precio de un producto propio antes de que se haya realizado la compraventa.

- El sistema debe permitir al vendedor retirar la venta de un producto propio (siempre que el producto no haya recibido pujas o compras inmediatas).

- El sistema debe permitir a los usuarios realizar ofertas a los vendedores por un objeto en venta inmediata.

- El sistema debe permitir a los vendedores aceptar o rechazar ofertas de un producto propio en venta inmediata.

- El sistema debe retirar un producto de la lista de productos en venta, cuando el vendedor ha aceptado una oferta por este o se ha llegado a la fecha límite.

- El sistema debe permitir a los usuarios realizar pujas por un producto en subasta.

- El sistema debe permitir a los usuarios añadir productos a su lista de "Siguiendo", siendo esta la lista de productos en los que ha indicado que está interesado.

- El sistema debe permitir a los usuarios retirar productos de su lista de "Siguiendo".

- El sistema debe permitir la búsqueda de productos mediante palabras clave (incluidas en el nombre y descripción de un producto en venta).

- El sistema debe permitir filtrar una lista de productos a la venta en función de localización, categoría, precio...etc.

- El sistema debe permitir a los usuarios ver las valoraciones propias recibidas y las de otros usuarios.

- El sistema debe permitir a los usuarios reportar a otros usuarios en caso de una infracción las reglas de uso (p.e. caso de estafa).

- El sistema debe permitir a los usuarios enviar mensajes a otros usuarios.

- El sistema debe permitir a los usuarios recibir mensajes de otros usuarios.

- El producto resultante será una aplicación web soportada en la mayoría de navegadores.

- La aplicación web tendrá un diseño responsive enfocada a los dispositivos móviles.

- El sistema requerirá vincular la cuenta de un usuario a una cuenta de google.

- El sistema deberá guardar la información de todas las acciones realizadas por los usuarios.

- Las acciones de los usuarios abarca el logearse, el editar su perfil, el seguir productos, el realizar ofertas y pujas por productos, el crear ventas y subastas, el editar la información de una venta y la creación de una etiqueta.

## 4.2. DISEÑO DEL SISTEMA

Para la aplicación web se plantearon Vue, React y Angular como tecnologías de desarrollo, decidiéndose finalmente el uso Vue. Para la aplicación se barajaron Android Studio (XML y Java), WebPack (HTML) y Flutter (Dart), y se eligió finalmente un desarrollo con XML y Java (Android Studio), por el hecho de que era un entorno y lenguajes con los que ya se tenía experiencia, y porque de este modo la aplicación sería completamente nativa, mejorando el rendimiento y por ende la experiencia de uso. Y para el servidor de la API se barajaron dos opciones, SQLAchemy (Python) y Spring Boot (Java), eligiéndose finalmente Spring Boot, ya que se tenía mucha experiencia con Java en comparación con Python, junto con una base de datos MySQL para pruebas y Postgresql para despliegue real. Para la interacción entre el cliente (tanto web como móvil) y el servidor se desarrollará una API RESTful, es decir, una API sin estado que responde a las peticiones basándose únicamente en la información de la base de datos, y la pasada en la propia petición. Se planteó el no hacerla REST, pero se consideró que este enfoque facilitaba el desacoplamiento entre cliente y servidor, facilitando la modularización del trabajo y quitando problemas de dependencias entre ambas partes, que podría hacer afectado al ritmo de desarrollo (p.e. si es el servidor el que se encarga de la redirección entre páginas, el back-end ya depende de que estén las páginas para saber a cuál redirigir y qué información dejarle, o por el contrario el cliente no puede probar realmente la navegación).

En cuanto a las operaciones, se podría decir que todas son síncronas, ya que siempre que el cliente lance una petición, este esperará la respuesta y actuará acorde a ella. Las únicas operaciones asíncronas serían algunas relacionadas con el registro de información de las acciones de los usuarios, de la cual el cliente no necesita saber nada, por lo que no esperará nada relacionado con ellas.

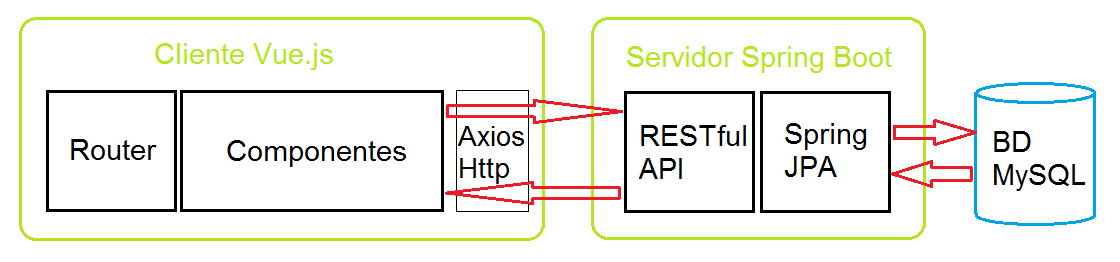
Tecnologías concretas:

– Java -Vue -XML

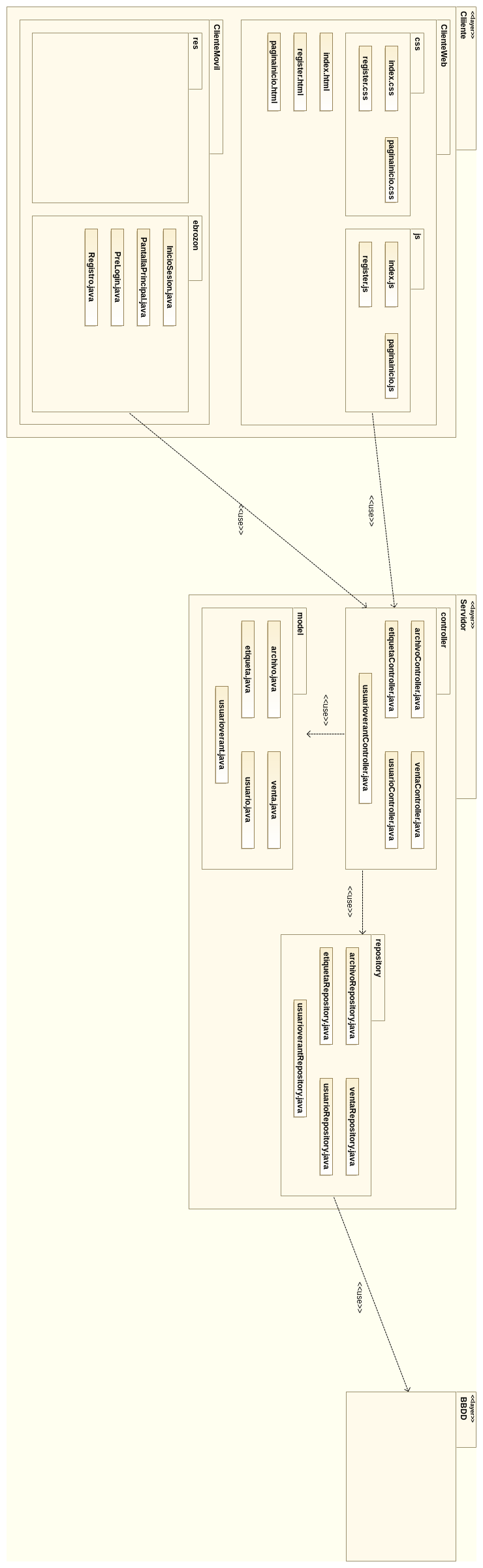
– Maven -Vue Router

– Spring Boot -Axios

Esquema arquitectura web



En cuanto a la arquitectura, solo destacar que toda la lógica relacionada con la gestión de datos se realizará en el servidor, limitándose el cliente (tanto web como móvil) a gestionar la lógica de la GUI y la sesión, y a realizar las peticiones correspondientes al servidor. De esto se concluiría que la arquitectura que seguiremos será la de cliente ligero y servidor pesado, al delegar al servidor los “cálculos” más complejos y costosos.

**Vista de módulos**

Se propone una arquitectura por capas, las cuales se describen a continuación.

* **Capa cliente**

Capa más externa, contiene la capa de presentación. Se divide en dos bloques, el correspondiente a la web y a la aplicación Android.

* + - **ClienteWeb**

Contiene los elementos que conforman la web de la aplicación, el módulo provee una interfaz gráfica al usuario, mientras que hace uso de la interfaz proporcionada por el servidor para llevar a cabo las operaciones correspondientes.

* + - * + css

Contiene los estilos de los elementos que hay en las vistas.

index.css

register.css

paginainicio.css

* + - * + index.html
        + register.html
        + paginainicio.html

Los elementos html indican que elementos hay en las vistas.

* + - * + js

Contiene la lógica de la web, para poder llevar a cabo las acciones requeridas por el usuario.

index.js

register.js

paginainicio.js

* + - **ClienteMovil**

Contiene los elementos que conforman la aplicación Android, el módulo porvee una interfaz gráfica al usuario, mientras que hace uso de la interfaz proporcionada por el servidor para llevar a cabo las operaciones correspondientes.

* + - * + ebrozon

Contiene la lógica necesaria para poder llevar a cabo las acciones requeridas por el usuario en las distintas pantallas.

InicioSesion.java

Contiene la lógica referente al inicio de la sesión.

PantallaPrincipal.java

Contiene la lógica referente a la pantalla principal de la aplicación.

PreLogin.java

Contiene la lógica referente a la pantalla de elección entre login o registro.

Registro.java

Contiene a lógica referente al registro en la aplicación.

* + - * + res

Contiene las pantallas que conforman la aplicación, proporcionan una interfaz gráfica al usuario.

* **Capa servidor**

Capa intermedia, contiene la capa de la lógica de la aplicación o de negocio.

* + - **Controller**

Contiene la API con la declaración de peticiones que es capaz de responder, y a las cuales da respuesta gracias a las funciones de repository.

* + - * archivoController.java

Implementa la API con las peticiones necesarias sobre un archivo. Hace uso de las interfaces ofrecidas por usuario y archivo del módulo Models y archivoRepository del módulo Repository. Ofrece la siguiente interfaz.

* + - * + uploadFile

Operación utilizada para guardar un fichero en la base de datos. Requiere como parámetro un tipo de dato Multipartfile, devuelve un entero.

* + - * + loadFile

Operación utilizada para cargar un fichero desde la base de datos. Requiere como parámetro un entero, devuelve un string.

* + - * + deleteFile

Operación utilizada para eliminar un fichero de la base de datos. Requiere como parámetro un entero, devuelve un string.

* + - * etiquetaController.java

Implementa la API con las peticiones necesarias sobre una etiqueta. Hace uso de las interfaces ofrecidas por usuario y etiqueta del módulo Models y de usuarioRepository y etiquetaRepository del módulo Repository. Ofrece la siguiente interfaz.

* + - * + guardar

Operación utilizada para guardar una etiqueta en la base de datos, Requiere como parámetros dos string y un tipo de dato Date, devuelve un string.

* + - * + recuperarEtiqueta

Operación utilizada para recuperar una etiqueta de la base de datos. Requiere como parámetro un string, devuelve un tipo de dato etiqueta.

* + - * + actualizarEtiqueta

Operación utilizada para actualizar una etiqueta en la base de datos. Requiere como parámetros dos string y un tipo de dato Date, devuelve un string.

* + - * usuarioController.java

Implementa la API con las peticiones necesarias sobre un usuario. Hace uso de las interfaces ofrecidas por usuario del módulo Model y usuarioRepository del módulo Repository. Ofrece la siguiente interfaz.

* + - * + registrar

Operación utilizada para guardar los datos de un usuario en la base de datos. Requiere como parámetros nueve string, dos enteros y un tipo de dato Multipartfile, devuelve un string.

* + - * + actualizarUsuario

Operación utilizada para actualizar los datos de un usuario en la base de datos. Requiere como parámetros siete string, dos enteros y un tipo de dato Multipartfile, devuelve un string.

* + - * + logear

Operación utilizada cuando el usuario hace login en la aplicación. Requiere como parámetros dos string, devuelve un string.

* + - * + recuperarUsuario

Operación utilizada para recuperar la información de un usuario. Requiere como parámetro un string. devuelve un tipo de dato usuario.

* + - * + banearUsuario

Operación utilizada para banear un usuario de la aplicación. Requiere como parámetro un string, devuelve un string.

* + - * + desbanearUsuario

Operación contraria a la anterior. Requiere como parámetro un string, devuelve un string.

* + - * usuarioverantController.java

Implementa la API con las peticiones necesarias sobre un usuarioverant. Hace uso de las interfaces ofrecidas por usuario y usuarioverant del módulo Model y usuarioverantRepository del módulo Repository. Ofrece la siguiente interfaz.

* + - * + guardar

Operación utilizada para guardar un usuarioverant en la base de datos. Requiere como parámetro un tipo de dato usuario.

* + - * ventaController.java

Implementa la API con las peticiones necesarias sobre un una venta. Hace uso de las interfaces ofrecidas por usuario y venta del módulo Model y ventaRepository del módulo Repository. Ofrece la siguiente interfaz.

* + - * + publicarVenta

Operación utilizada para publicar una venta en la aplicación. Requiere como parámetros tres string, un double y un tipo de dato Multipartfile, devuelve un string.

* + - **Model**

Contiene las estructuras de datos que representan las tablas de la base de datos y sus restricciones.

* + - * archivo.java

Implementa la tabla archivo. Ofrece como interfaz los getters y setters sobre sus atributos.

* + - * etiqueta.java

Implementa la tabla etiqueta. Ofrece como interfaz los getters y setters sobre sus atributos.

* + - * usuario.java

Implementa la tabla usuario. Ofrece como interfaz los getters y setters sobre sus atributos.

* + - * usuarioverant.java

Implementa la tabla usuarioverant. Ofrece como interfaz los getters y setters sobre sus atributos.

* + - * venta.java

Implementa la tabla venta. Ofrece como interfaz los getters y setters sobre sus atributos.

* + - **Repository**

Contiene las funciones necesarias para la persistencia de los modelos en la base de datos.

* + - * archivoRepository.java

Ofrece las operaciones necesarias para la persistencia de un archivo en la base de datos.

* + - * + existsByIdentificador

Operación utilizada para saber si existe en la base de datos un archivo con un identificador determinado. Requiere como parámetro un entero, devuelve un boolean.

* + - * + findByIdentificador

Operación utilizada para encontrar en la base de datos un archivo con un identificador determinado. Requiere como parámetro un entero, devuelve un archivo si está en la base de datos.

* + - * + findByurl

Operación utilizada para encontrar en la base de datos un archivo con una url determinada. Requiere como parámetro una url, devuelve un archivo si está en la base de datos.

* + - * etiquetaRepository.java

Ofrece las operaciones necesarias para la persistencia de una etiqueta en la base de datos.

* + - * + existsBynombre

Operación utilizada para saber si existe en la base de datos una etiqueta con un nombre determinado. Requiere como parámetro un string, devuelve un boolean.

* + - * + findBynombre

Operación utilizada para encontrar en la base de datos una etiqueta con un nombre determinado. Requiere como parámetro un string, devuelve una etiqueta si está en la base de datos.

* + - * usuarioRepository.java

Ofrece las operaciones necesarias para la persistencia de un usuario en la base de datos.

* + - * + existsBynombreusuario

Operación utilizada para saber si existe en la base de datos un usuario con un nombre determinado. Requiere como parámetro un string, devuelve un boolean.

* + - * + existsBycorreo

Operación utilizada para saber si existe en la base de datos un usuario con un correo determinado. Requiere como parámetro un string, devuelve un boolean.

* + - * + findBynombreusuario

Operación utilizada para encontrar en la base de datos un usuario con un nombre determinado. Requiere como parámetro un string, devuelve un usuario si está en la base de datos.

* + - * + findBycorreo

Operación utilizada para encontrar en la base de datos un usuario con un correo determinado. Requiere como parámetro un string, devuelve un usuario si está en la base de datos.

* + - * + registrarLogin

Operación utilizada par guardar el login de un usuario en la base de datos. Requiere como parámetro un string.

* + - * usuarioverantRepository.java

Ofrece las operaciones necesarias para la persistencia de un usuarioverant en la base de datos.

* + - * ventaRepository.java

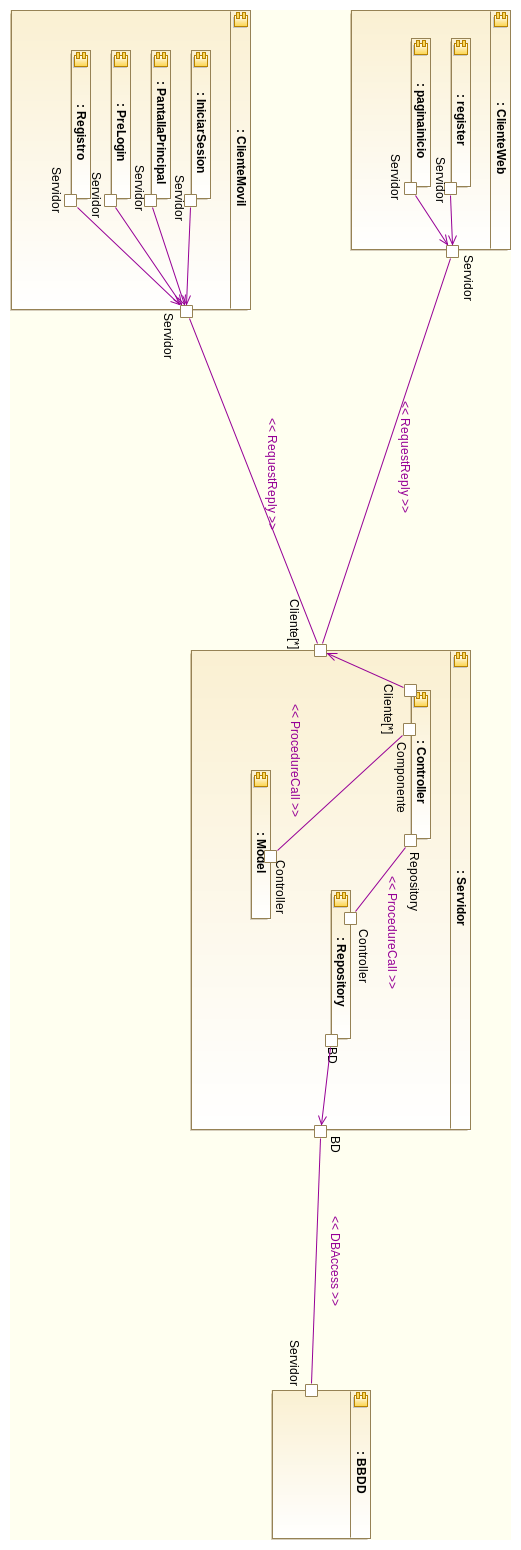
Ofrece las operaciones necesarias para la persistencia de una venta en la base de datos.

* + - * + archivoApareceEnVenta

Operación utilizada para saber si un determinado archivo aparece reflejado en una venta. Requiere como parámetros un entero, un string y un tipo de dato Date.

* **Capa base de datos**

Capa más interna, contiene la capa de gestión de datos o recursos.

**Vista de componente y conector**

**Componentes**

* **ClienteWeb**

Componente que se encarga de la aplicación web, atiende las peticiones del usuario. Este componente cuenta con una serie de componentes internos que se encargan de tareas específicas.

* + - **paginainicio**

Entre las funciones de este componente se encuentra la presentación de la página de inicio, así como gestionar las acciones que realice el usuario sobre dicha página.

* + - **register**

Entre las funciones de este componente se encuentra la presentación de la página de registro, así como gestionar las acciones asociadas al registro que el usuario tiene disponibles.

Estos componentes requieren una interfaz para la comunicación con el servidor.

* **ClienteMovil**

Componente que se encarga de la aplicación Android, atiende las peticiones del usuario. Este componente cuenta con una serie de componentes internos que se encargan de tareas específicas.

* **IniciarSesion**

Entre las funciones de este componente se encuentra el inicio de sesión en el sistema.

* **PantallaPrincipal**

Entre las funciones de este componente se encuentran mostrar productos en venta, acceso al perfil, búsqueda de productos, de usuarios.

* **PreLogin**

Entre las funciones de este componente se encuentra la elección de realizar el login o el registro en el sistema.

* **Registro**

Entre las funciones de este componente se encuentra realizar el registro de un usuario en el sistema.

Estos componentes requieren una interfaz para la comunicación con el servidor.

* **Servidor**

Componente que se encarga del servidor del sistema, atiende la peticiones que llegan desde los clientes y proporciona las respuestas necesarias. Este componente cuenta con una serie de componentes internos que se encargan de tareas específicas.

* + - **Controller**

Entre las funciones de este componente se encuentran gestionar las peticiones de los clientes. Ofrece una interfaz que estos pueden utilizar, requiere una interfaz para gestionar los modelos de datos y otra para poder realizar peticiones sobre la base de datos.

* + - **Model**

Entre las funciones de este componente se encuentra gestionar los modelos de datos. Ofrece una interfaz al componente Controller.

* + - **Repository**

Entre las funciones de este componente se encuentra gestionar las peticiones sobre la base de datos. Ofrece una interfaz al componente Controller, requiere otra para realizar operaciones sobre la base de datos.

* **BBDD**

Componente que se corresponde con el motor de base de datos SQL transaccional

encargado del almacenamiento de datos, permitiendo su creación, modificación,

borrado y consulta. Ofrece una interfaz utilizada por el servidor.

**Conectores**

En la vista se pueden observar tres tipos de conectores: conectores ProcedureCall entre subcomponentes de componente servidor, DBAccess para el acceso a la base de datos por parte del componente servidor y RequestReply entre los componentes cliente y el servidor para la gestión de peticiones por parte del usuario.

* **ProcedureCall**

Los conectores ProcedureCall modela el flujo de control entre los componentes mediante invocaciones. Realizan además la transferencia de datos de datos entre los componentes que interactúan mediante el uso de parámetros y valores de retorno.

Por ello, son conectores de comunicación y de coordinación. En este caso, todos los conectores son llamadas a métodos Java, que proveen la transferencia de sus parámetros por referencia, un valor de retorno, su accesibilidad puede ser pública o privada, entre otras características.

Todos los conectores que permiten la comunicación entre subcomponentes de Servidor son de este tipo.

* **DBAccess**

El conector DBAccess permite el acceso a los datos almacenados en otro componente almacén de datos. Provee servicios de comunicación. En la vista se presenta un conector que permite la comunicación entre la base de datos y el servidor.

El servidor envía peticiones y consultas (queries) a la base de datos gracias a la interfaz proporcionada por esta y la base de datos devuelve la información correspondiente a cada petición.

* **RequestReply**

El conector RequestReply es un conector de tipo evento, el flujo se desencadena por la ocurrencia de un evento, esto permite que exista comunicación asíncrona entre los distintos clientes y el servidor.

Todos los conectores que permiten la comunicación entre el cliente web, el cliente Android y el servidor son de este tipo.

**Interfaces**

De entre las interfaces proporcionadas por los componentes anteriormente mencionados destaca las proporcionadas por los componentes Controller, Model y Repository.

* **Controller**

Todas las acciones que llegan desde los clientes se realizan mediante llamadas a las funciones ofrecidas por Controller

* + - int uploadFile(@RequestParam("file") MultipartFile file)
    - String loadFile(@RequestParam("identificador") int id)
    - String deleteFile(int id)
    - saveFile(MultipartFile file, String filename) throws IOException
    - String guardar(@RequestParam("tn") String tn, @RequestParam("cd") Date cd, @RequestParam("crt") String crt)
    - etiqueta recuperarEtiqueta(@RequestParam("tn") String tn)
    - String actualizarEtiqueta(@RequestParam("tn") String tn, @RequestParam("cd") Date cd, @RequestParam("crt") String crt)
    - String registrar(@RequestParam("un") String un, @RequestParam("cor") String cor, @RequestParam("pass") String pass, @RequestParam(value = "tel", required=false) Integer tel, @RequestParam("na") String na, @RequestParam("lna") String lna, @RequestParam(value = "cp", required=false) Integer cp, @RequestParam(value = "ci", required=false) String ci, @RequestParam(value = "pr", required=false) String pr, @RequestParam(value = "lat", required=false) String lat, @RequestParam(value = "lon", required=false) String lon, @RequestParam(value = "im", required=false) MultipartFile im)
    - String actualizarUsuario(@RequestParam("un") String un, @RequestParam("tel") Integer tel, @RequestParam("name") String na, @RequestParam("lna") String lna, @RequestParam("cp") Integer cp, @RequestParam("ci") String ci, @RequestParam("pr") String pr, @RequestParam("lat") String lat, @RequestParam("lon") String lon, @RequestParam("im") MultipartFile im)
    - String logear(@RequestParam("un") String un, @RequestParam("pass") String pass)
    - usuario recuperarUsuario(@RequestParam("un") String un)
    - String banearUsuario(@RequestParam("un") String un)
    - String desbanearUsuario(@RequestParam("un") String un)
    - guardar(usuario u)
    - publicarVenta(@RequestParam("un") String un, @RequestParam("prod") String prod, @RequestParam("desc") String desc, @RequestParam("pre") double pre, @RequestParam(value = "arc", required=false) MultipartFile arc)
* **Model**

La gestión de los modelos por parte del componente Controller se realiza mediante llamadas a las funciones de Model.

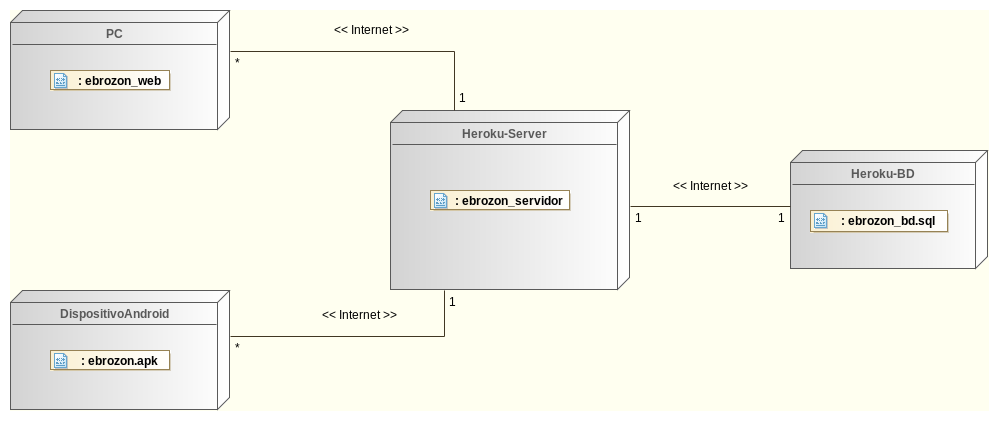
* + - **archivo**(String url, int borrada)
    - int getIdentificador()
    - setIdentificador(int identificador)
    - String getUrl()
    - setUrl(String url)
    - int getBorrada()
    - setBorrada(int borrada)
    - **etiqueta**(String nombre, Date fechacreacion, String creador)
    - String getNombre()
    - setNombre(String nombre)
    - getFechacreacion()
    - setFechacreacion(Date fechacreacion)
    - getCreador()
    - setCreador(String creador)
    - **usuario**(String nombreusuario, String correo, String contrasena, String nombre, String apellidos)
    - **usuario**(String nombreusuario, String correo, String contrasena, int telefono, String nombre, String apellidos, int codigopostal, String ciudad, String provincia)
    - **usuario**(String nombreusuario, String correo, String contrasena, int telefono, String nombre, String apellidos, int codigopostal, String ciudad, String provincia,float latitud, float longitud, int archivo)
    - String getNombreusuario()
    - setNombreusuario(String nombreusuario)
    - String getCorreo()
    - setCorreo(String correo)
    - String getContrasena()
    - setContrasena(String contrasena)
    - getTelefono()
    - setTelefono(int telefono)
    - String getNombre()
    - setNombre(String nombre)
    - String getApellidos()
    - setApellidos(String apellidos)
    - getCodigopostal()
    - setCodigopostal(int codigopostal)
    - String getCiudad()
    - setCiudad(String ciudad)
    - String getProvincia()
    - setProvincia(String provincia)
    - float getLatitud()
    - setLatitud(float latitud)
    - float getLongitud()
    - setLongitud(float longitud)
    - int getArchivo()
    - void setArchivo(int archivo)
    - int getActivo()
    - setActivo(int activo)
    - setUrlArchivo(String url)
    - String getUrlArchivo()
    - **usuarioverant**(usuario u)
    - String getNombreusuario()
    - setNombreusuario(String nombreusuario)
    - Date getFecha()
    - setFecha(Date fecha)
    - getTelefono()
    - setTelefono(int telefono)
    - String getNombre()
    - setNombre(String nombre)
    - String getApellidos()
    - setApellidos(String apellidos)
    - int getCodigopostal()
    - setCodigopostal(int codigopostal)
    - String getCiudad()
    - setCiudad(String ciudad)
    - String getProvincia()
    - setProvincia(String provincia)
    - float getLatitud()
    - setLatitud(float latitud)
    - float getLongitud()
    - setLongitud(float longitud)
    - int getImagen()
    - setImagen(int imagen)
    - int getActivo()
    - setActivo(int activo)
    - **venta**(String usuario, String producto, String descripcion, double precio, int tienearchivo, int activa)
    - String getUsuario()
    - setUsuario(String usuario)
    - getFechainicio()
    - setFechainicio(Date fechainicio)
    - getFechaventa()
    - setFechaventa(Date fechaventa)
    - String getProducto()
    - setProducto(String producto)
    - String getDescripcion()
    - setDescripcion(String descripcion)
    - double getPrecio()
    - setPrecio(double precio)
    - double getPreciofinal()
    - setPreciofinal(double preciofinal)
    - String getComprador()
    - setComprador(String comprador)
    - Date getFechapago()
    - setFechapago(Date fechapago)
    - int getTienearchivo()
    - setTienearchivo(int tienearchivo)
    - int getActiva()
    - setActiva(int activa)
    - int getEsSubasta()
    - setEsSubasta(int esSubasta)
    - usuario getUser()
    - setUser(usuario us)
* **Repository**

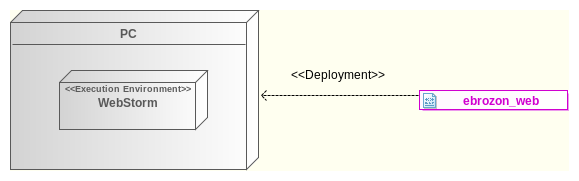
Todas las operaciones de base de datos se manejan dentro de la aplicación mediante llamadas a funciones contenidas en Repository.

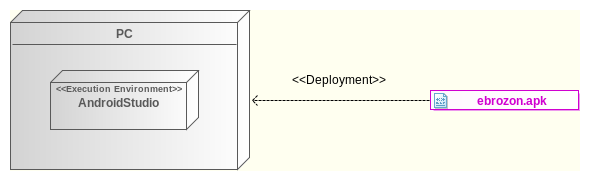
* + - boolean existsByidentificador(int identificador)
    - Optional<archivo> findByidentificador(int identificador)
    - Optional<archivo> findByurl(String url)
    - boolean existsBynombre(String nombreetiqueta)
    - Optional<etiqueta> findBynombre(String nombre)
    - boolean existsBynombreusuario(String nombreusuario)
    - Optional<usuario> findBynombreusuario(String nombreusuario)
    - boolean existsBycorreo(String correo)
    - Optional<usuario> findBycorreo(String correo)
    - registrarLogin(String un)

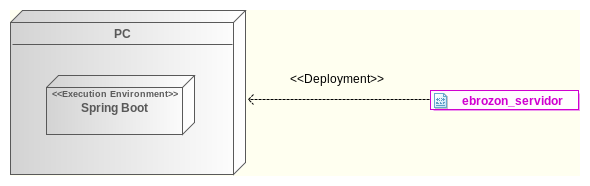
archivoApareceEnVenta(int arc, String un, Date fi)

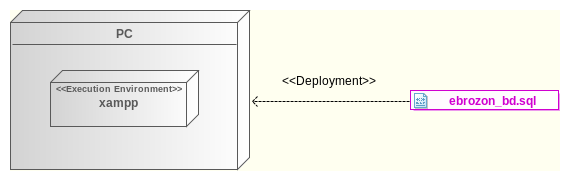
**Vistas de distribución**

**Vista de despliegue**

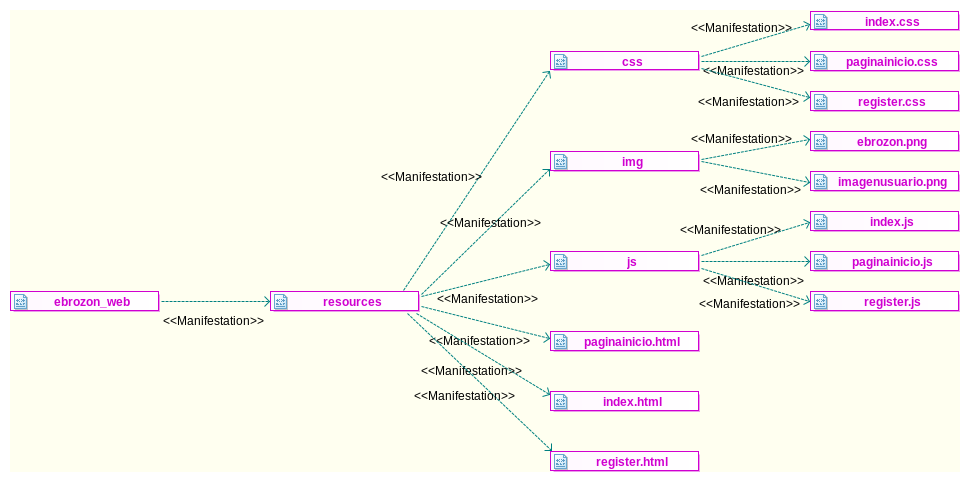
**Vista de desarrollo - Front End**

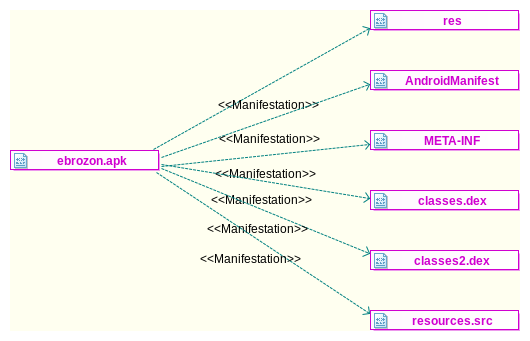
**Vista de desarrollo - Android**

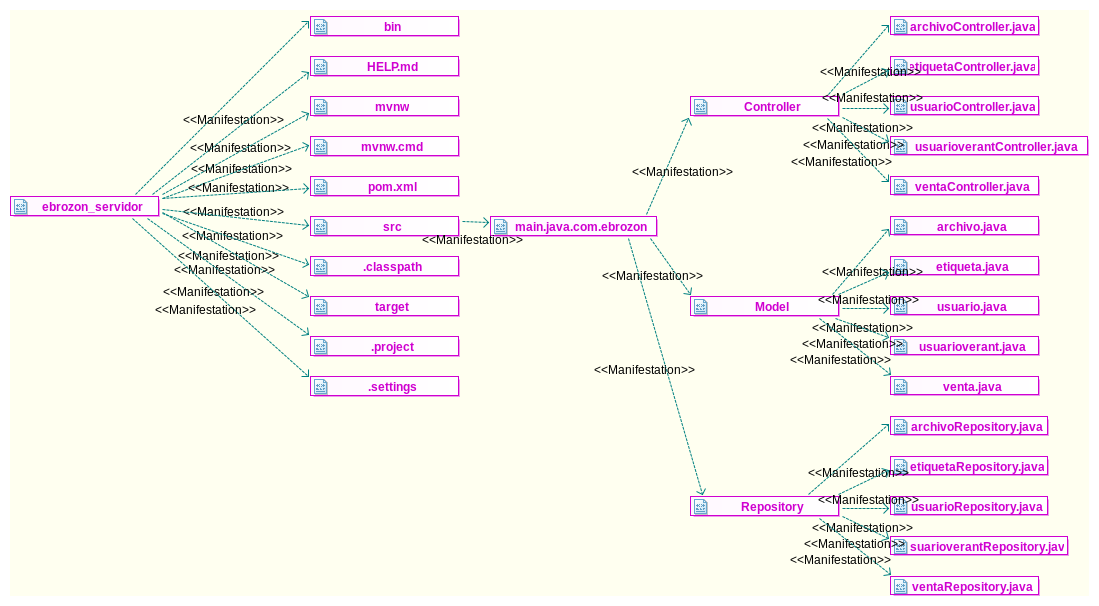
**Vista de desarrollo - Back End**

**Vista de desarrollo – BD**

**Vista de instalación - Front End**

****

**Vista de instalación – Android**

**Vista de instalación - Back End**

**Vista de distribución de trabajo**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ebrozon** | | | **Comentarios** |
| **Segmento** | **Subsistema** | **Unidad de organización** |
| **Back-end** | Controller | Saul, Eduardo, Alejandro | Cada miembro implementa una tabla al completo |
| Repository |
| Model |
| **Front-End** | html | Jorge, Israel, Félix | Cada miembro implementa una pantalla al completo |
| css |
| js |
| **Android** | xml | Andrés, Victor, Sergio | Cada miembro implementa una pantalla al completo |
| java |
| **Documentación** |  | Equipo completo | Cada miembro se ocupa de un subapartado, puede no ser siempre el mismo. |

# 5. MEMORIA DEL PROYECTO

-

## 5.1. INICIO DEL PROYECTO

-

## 5.2. EJECUCIÓN Y CONTROL DEL PROYECTO

-

## 5.3. CIERRE DEL PROYECTO

-

# 6. CONCLUSIONES

-

# ANEXO I. GLOSARIO

-

# ANEXO II. ACTAS DE TODAS LAS REUNIONES REALIZADAS

-

# ANEXO III. OTROS ANEXOS QUE SE CONSIDEREN NECESARIOS

-