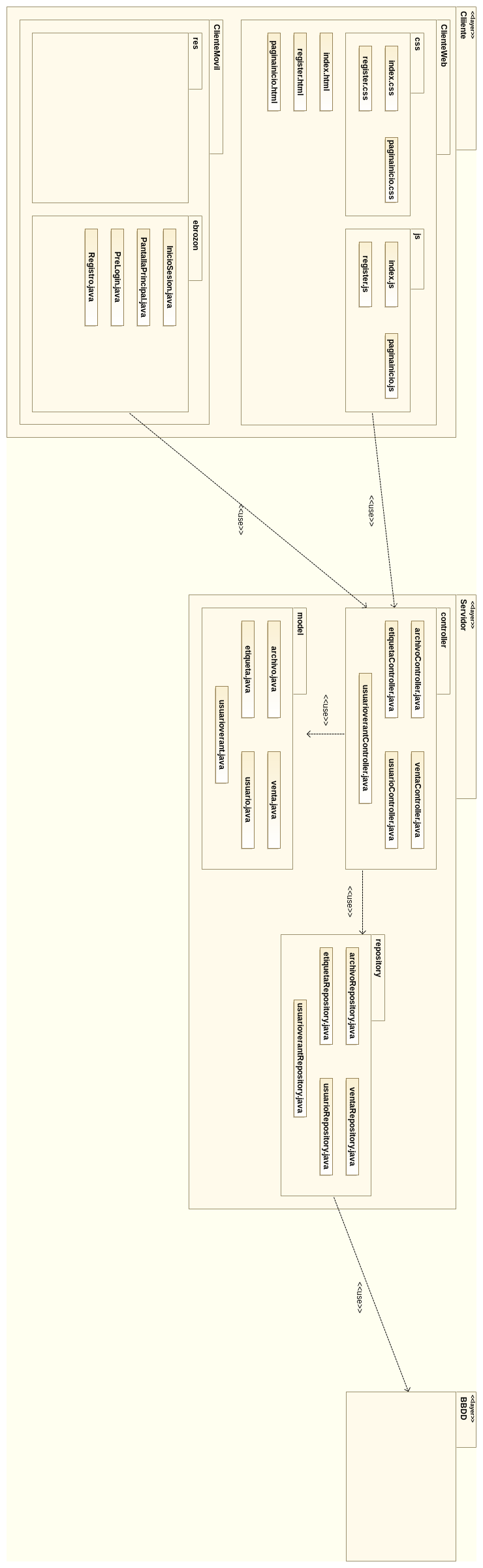
## 4.2. DISEÑO DEL SISTEMA

**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.

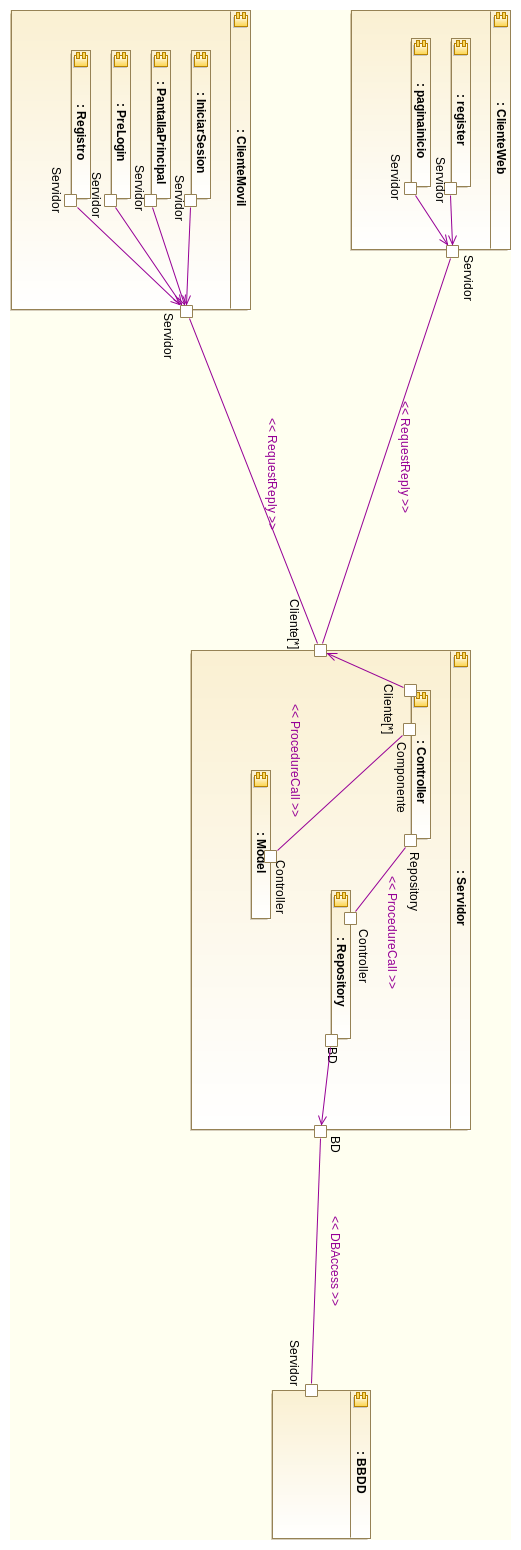
* + - * + archivoApareceEnVenta

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.

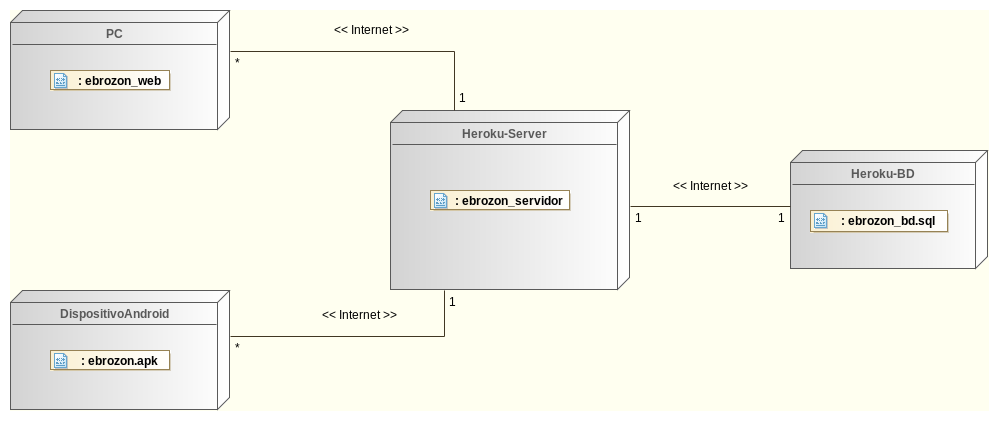
* + - **archivo**(String url, int borrada)
    - int getIdentificador()
    - setIdentificador(int identificador)
    - String getUrl()
    - setUrl(String url)
    - int getBorrada()
    - setBorrada(int borrada)
    - **etiqueta**(String nombre, Date fechacreacion, String creador)
    - String getNombre()
    - 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.

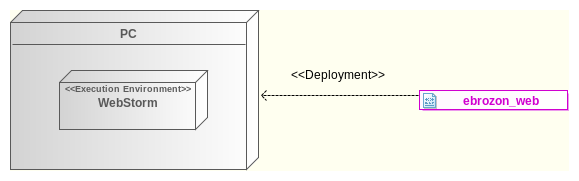
* + - boolean existsByidentificador(int identificador)
    - Optional<archivo> findByidentificador(int identificador)
    - Optional<archivo> findByurl(String url)
    - boolean existsBynombre(String nombreetiqueta)
    - Optional<etiqueta> findBynombre(String nombre)
    - boolean existsBynombreusuario(String 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)

**Vista de distribución**

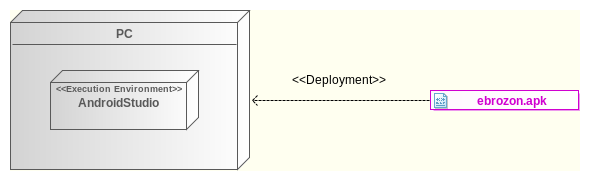
**Vista de despliegue**



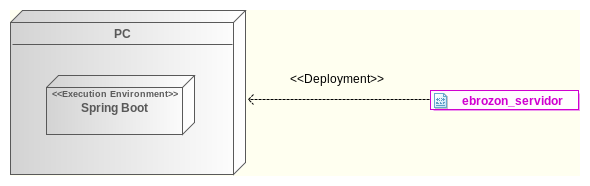
**Vista de desarrollo - Front End**

****

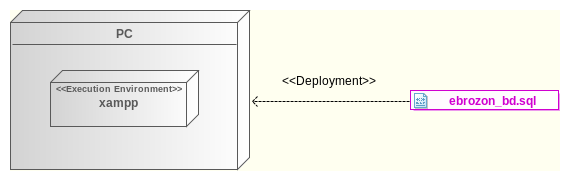
**Vista de desarrollo - Android**

****

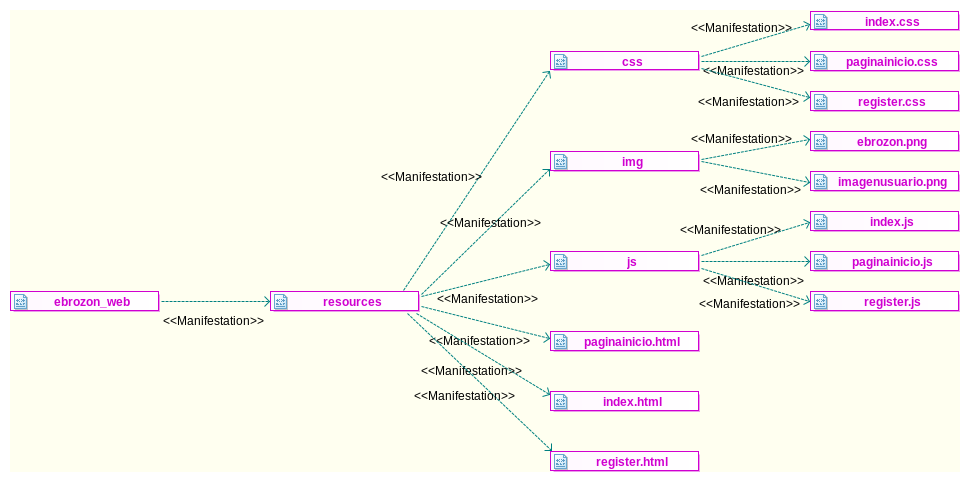
**Vista de desarrollo - Back End**

****

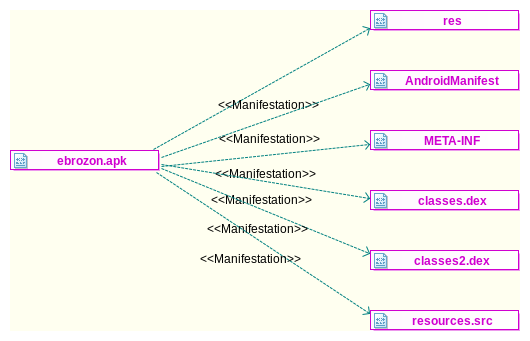
**Vista de desarrollo - BD**

****

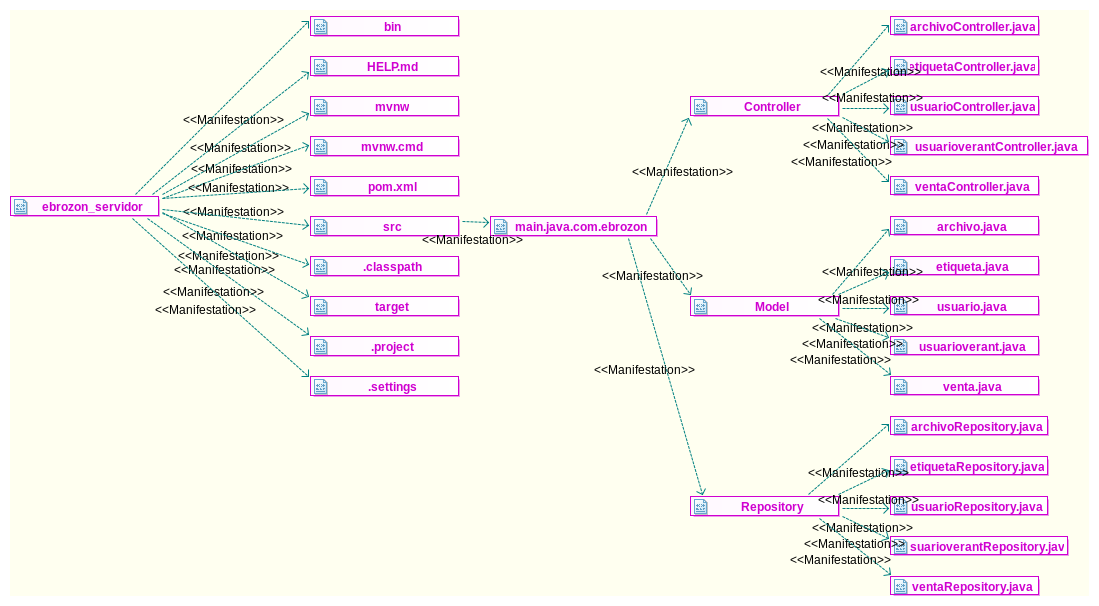
**Vista de instalación - Front End**

****

**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. |