Proyecto Integrado DAM

WIVA – Make a wish

01/01/2016

Rubén Abarca Jiménez
Francisco Pleguezuelos Lorente
Pablo José Herrera Marabotto

Index

Introducción

Nuestro proyecto integrado consiste en una “Whish List”, es decir, una aplicación para almacenar y compartir objetos que queremos tener pero no podemos obtener por un motivo u otro.

Para esto, la aplicación utilizará la cámara para tomar fotos de las cosas que queremos o hemos visto y nos han llamado la atención. Tras tomar la foto, le añadiremos datos como el precio, categoría, que tanto lo quieres, etc, aparte de obtener la localización y la fecha del momento. Una vez guardado el deseo, se añadirá a una de nuestras listas de deseos. Cada usuario podrá tener varias listas de deseos las cuales pueden ser públicas o no. Cuando un deseo sea “cumplido”, se marcará como tal en la lista y se moverá a una lista de deseos completados.

Existirá un buscador para buscar personas y poder añadirlas a amigos. De esta forma, dos usuarios que sean amigos entre sí podrán ver sus listas de deseos públicas.

Además todos los productos de los deseos se almacenarán en una base de datos donde todos los usuarios podrán ver y buscar productos, saber dónde fueron vistos y añadirlos como un deseo propio a una de sus listas.

Objetivo del proyecto

Wiva es una aplicación orientada al almacenamiento de datos, en este caso, objetos que quiere una persona. Con la funciones de amistad se volverá mucho más sencillo tratar con días especiales tales como cumpleaños o aniversarios ya que podrán consultarse que desea la persona. También avisa cuando se acerca el cumpleaños de un amigo.

Wiva podrá además ser un estímulo para los comercios debido a la fácil forma de crear los deseos, a que guarda la localización de donde se tomó la foto, ya que el producto es visible por todos los usuarios de la aplicación en la lista total de productos, y además, cada producto tiene un sistema de votaciones y visitas.

Requisitos

Wiva requerirá de una serie de servicios para poder funcionar en su estado más básico. Los principales serian el uso de la cámara, el acceso a internet y tener el GPS habilitado. Hablando de los permisos del móvil, aparte de los mencionados arriba, también son necesarios la escritura y lectura en la memoria externa.

Diseño del servidor

Nuestro servidor ha sido programado en Symfony2, un framework de desarrollo basado en PHP, en el que se ha implementado toda la funcionalidad programándola por capas y de forma ordenada según la documentación de Symfony.

La capa de datos la vemos implementada mediante distintos bundles que contienen las distintas entidades de nuestra aplicación haciendo que nuestra base de datos sea orientada a objetos.

Esto lo conseguimos gracias al uso de las librerías de Doctrine ORM creando la capa de abstracción entre nuestra base de datos y la lógica de negocio, y no siendo necesario crear nuestra base de datos mediante scripts SQL ni uso de Phpmyadmin.

Aquí tenemos una pequeña parte del código de nuestra entidad Producto donde podemos ver las anotaciones que usamos para que Doctrine pueda usarlas para generar nuestra base de datos.

```php
use Doctrine\ORM\Mapping as ORM;

/**
 * producto
 *
 * @ORM\Table(name="producto")
 * @ORM\Entity(repositoryClass="Web\productBundle\Repository\productoRepository")
 */
class producto
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="nombre", type="string", length=255)
     */
    private $nombre;

    /**
     * @var string
     *
     * @ORM\Column(name="descripcion", type="string", length=255)
     */
    private $descripcion;
```

Gracias a esto, cuando queremos tener acceso a los datos de nuestra base de datos para consultarlos, editarlos, borrarlos o crear nuevos, solo necesitamos usar los métodos de Doctrine para llamar al repositorio de nuestra entidad y así crear un Entity Manager que nos servirá para obtener una instanciación de nuestra entidad ya rellena con los datos almacenados en nuestra base de datos.

La capa de negocio o lógica de negocio se encuentra en nuestros controladores, donde recogemos en distintos métodos las peticiones que nos llegan al servidor.

En estos controladores aplicamos todas las comprobaciones necesarias para tratar los datos que nos llegan en la petición, validar estos datos con él resto de nuestros datos ya existentes (Foreing keys, datos de control, etc…), adaptarlos a la forma necesaria para adecuarlos a nuestro servidor y persistirlos mediante los métodos que Doctrine pone a nuestra disposición gracias a la capa de abstracción creada en nuestra capa de datos.

Los métodos de nuestros controladores se basan en recoger los datos de la petición, gestionarlos y posteriormente devolver los datos consultados o insertados al usuario en formato JSON.


```
{
  "errorCode": 0,
  "message": "",
  "Productos": [
    {
      "id": 1,
      "nombre": "Ps4",
      "descripcion": "Videoconsola domestica",
      "fecha": "2016/06/21 00:00",
      "codigoBarras": "",
      "likes": 0,
      "numeroVistas": 5
    },
    {
      "id": 2,
      "nombre": "Gps TomTom",
      "descripcion": "Apartato gps portatil para el coche",
      "fecha": "2016/06/04 16:26",
      "codigoBarras": "",
      "likes": 0,
      "numeroVistas": 0
    },
    {
      "id": 3,
      "nombre": "Tableta Grafica Wacom",
      "descripcion": "Tableta para diseñadores",
      "fecha": "2016/06/04 16:26",
      "codigoBarras": "",
      "likes": 0,
      "numeroVistas": 0
    },
    {
      "id": 4,
      "nombre": "Camiseta Futbol Granada",
      "descripcion": "Camiseta de los jugadores de futball del equipo Granada C.F.",
      "fecha": "2016/06/04 16:28",
      "codigoBarras": "",
      "likes": 0,
      "numeroVistas": 0
    },
    {
      "id": 5,
      "nombre": "Iphone 5",
      "descripcion": "Movil ultima generación",
      "fecha": "2016/06/07 00:00",
      "codigoBarras": "",
      "likes": 2,
      "numeroVistas": 6
    }
  ]
}
```

```php
class BaseMobileController extends Controller
{
    public function baseJsonResponse($array, $errorCode, $message) {
        $response = array("errorCode" => $errorCode,
            "message" => $message);
        foreach ($array as $key => $value) {
            $response[$key] = $value;
        }

        return new Response(json_encode($response));
    }

    public function successResponse($array) {
        return $this->baseJsonResponse($array, 0, "");
    }

    public function failureResponse($errorCode, $message) {
        return $this->baseJsonResponse(array(), $errorCode, $message);
    }
}
```

Para las respuestas hacemos uso de un controlador base para las respuestas, y de esta forma tener uniformidad en las respuestas que enviamos al cliente, siempre vamos a devolver una respuesta JSON con la siguiente estructura.

Como podemos ver en la imagen anterior, siempre devolvemos un JSON Array que contiene el objeto que estamos consultando, un mensaje de error y un código de error.

Para codificar el JSON Array que devuelve los datos necesarios hacemos uso de métodos auxiliares para codificar las entidades y del método “json_encode()” como podemos ver en la siguiente imagen.

```php
public function encodeProducto($producto) {
    $arrayproducto = array(
        "id" => $producto->getId(),
        "nombre" => $producto->getNombre(),
        "descripcion" => $producto->getDescripcion(),
        "fecha" => $producto->getFecha()->format(W3C),
        "codigoBarras" => $producto->getCodigoBarras(),
        "likes" => $producto->getLikes(),
        "numeroVistas" => $producto->getNumeroVistas()
    );
    return $arrayproducto;
}

public function encodeComentario($comentario) {
    $arraycomentario = array(
        "id" => $comentario->getId(),
        "comentario" => $comentario->getComentario(),
        "fecha" => $comentario->getFecha()->format(W3C),
    );
    return $arraycomentario;
}

public function encodeUbicacion($ubicacion){
    $arrayubicacion = array(
        "id" => $ubicacion->getId(),
        "latitud" => $ubicacion->getLatitud(),
        "longitud" => $ubicacion->getLongitud(),
        "precio" => $ubicacion->getPrecio(),
    );
    return $arrayubicacion;
}
```

En el caso de que la operación haya salido exitosa, devolveremos el mensaje de error vacío, y nuestro código de error a 0.

En caso de que se produzca algún error, se nos devolverla un mensaje de error con una breve descripción del problema y un código de error para que el administrador sepa en qué parte del código se encuentra el fallo.

Todo esto es posible gracias a un minucioso control de errores llevado a cabo de los controladores, y en la asociación de cada error a una serie de constantes numéricas definidas en el BaseMobileController que podemos ver a continuación.

```
define("WRONG_USERNAME", 2);
define("WRONG_PASSWORD", 3);
define("WRONG_IDDEVICE", 4);
define("UNPARSEABLE_RESPONSE", 6);
define("INVALID_USER", 7);
define("NO_POST_REQUEST", 8);
define("NO_RESULTS_AVAIlABLE", 9);
define("EMPTY_TITLE", 10);
define("ISSUE_NOT_FOUND", 11);
define("EMPTY_PARAMETER", 12);
define("EMPTY_DESCRIPTION", 13);
define("INVALID_NUMBER_PARAMETER", 14);
define("INVALID_STATUS", 15);
define("EMPTY_STATUS", 16);
define("EMPTY_COMMENTARY", 17);
define("EMPTY_IDINCIDENCIA", 18);
define("USER_ALREADY_EXIST", 19);
define("EMPTY_IDLISTA", 20);
define("EMPTY_IDPRODUCTO", 21);
define("WISH_NOT_FOUND", 22);
define("PRODUCT_NOT_FOUND", 23);
define("WISHLIST_NOT_FOUND", 24);
define("CATEGORY_NOT_FOUND", 25);
define("USER_NOT_FOUND", 25);
define("EMPTY_NAME", 27);
define("EMPTY_EMAIL", 28);

define("W3C", "Y/m/d H:i");
```

Aquí podemos ver uno de los métodos más básicos del controlador de productos, este método accede a nuestra base de datos mediante “$this->getDoctrine()->getRepository()” al cual le pasamos la entidad de la cual queremos consultar ‘productBundle’ y usamos posteriormente el método “->findAll()” para que nos devuelva todos los datos de la tabla productos.

Posteriormente, comprobamos que nos han llegado los productos, para así codificarlos en formato JSON y devolvérselos al dispositivo que nos ha hecho la petición

```
public function listarProductosAction() {
    $productos = $this->getDoctrine()
            ->getRepository('productBundle:producto')
            ->findAll();

    if (!$productos) {
        $this->failureResponse(NO_RESULTS_AVAIlABLE, "No se han encontrado productos");
    }
    $arrayproductos = array();
    foreach ($productos as $producto) {
        array_push($arrayproductos, $this->encodeProducto($producto));
    }

    $arrayJson['Productos'] = $arrayproductos;
    return $this->successResponse($arrayJson);
}
```

Los métodos findAll() o find() no son los únicos que podemos usar, también podemos añadir nuestras propias consultas en el repositorio asociado a nuestra entidad, definiendo un nuevo método como findByCategory().

```
class productoRepository extends EntityRepository
{
    public function findByCategory($cat){
        $em = $this->getEntityManager();
        $queryBuilder = $em->getRepository('productBundle:producto')->createQueryBuilder('a');

        $where = "a.categoria = '$cat'";
        $queryBuilder->where($where);

        return $queryBuilder->getQuery()->getResult();
    }
}
```

Los métodos de los controladores son usados para todos las acciones de la aplicación, desde listar una serie de datos hasta eliminarlos pasando por la creación y edición de los mismos.

La mayoría de métodos intentas ser simples y lo más concisos posibles, siempre validando al máximo para reducir los posibles fallos y corrupción de datos.

```
public function eliminarDeseoAction() {
    $peticion = $this->container->get('request_stack')->getCurrentRequest();
    if ($peticion->getMethod() == 'POST') {
        $id = $peticion->request->get("id");
        return $this->borrarDeseoById($id);
    }
}
```

Estos métodos llaman a su vez a otros métodos para separar la parte en la que recibimos los datos de la respuesta como hacemos en el método anterior, mientras que validamos y accedemos a la base de datos en un método auxiliar como vemos a continuación.

```
public function borrarDeseoById($id) {
    if ($id == null) {
        return $this->failureResponse(EMPTY_PARAMETER, "Falta algun parametro en la funcion");
    }

    $em = $this->getDoctrine()->getManager();
    $deseo = $em->getRepository('wishBundle:deseo')->find($id);

    if (!$deseo) {
        return $this->failureResponse(WISH_NOT_FOUND, 'No se ha encontrado el deseo con la id ' . $id);
    }
    $em->remove($deseo);
    $em->flush();
    return $this->successResponse(array());
}
```

Como podemos ver, existen en nuestro servidor, desde métodos muy simples, hasta métodos más complejos, en los que es necesario hacer uso de varias entidades a la vez y persistirlos en nuestra base de datos de forma ordenada para respetar todas las relaciones que existen entre las distintas tablas.

En el siguiente método podemos observar como para crear un deseo nuevo, debemos crear el registro del producto básico al cual hace referencia el deseo, para posteriormente añadir un objeto ubicación que guarde las coordenadas y el precio, para por último, crear el deseo y asociarlo a una lista de deseos existente de nuestro usuario.

```
public function crearDeseoAction() {
    $peticion = $this->container->get('request_stack')->getCurrentRequest();

    if ($peticion->getMethod() == 'POST') {
        $fecha = new \DateTime("now");
        $idLista = $peticion->request->get("lista");
        $idProducto = $peticion->request->get("producto");
        $nivel = $peticion->request->get("nivel");

        //Variables Localizacion Precio
        $latitud = $peticion->request->get("latitud");
        $longitud = $peticion->request->get("longitud");
        $precio = $peticion->request->get("precio");

        $em = $this->getDoctrine()->getManager();

        if ($idLista != null && $idProducto != null && $nivel != null) {
            $lista = $em->getRepository('wishBundle:ListaDeseo')->find($idLista);
            if ($lista != null) {
                $producto = $em->getRepository('productBundle:producto')->find($idProducto);
                if ($producto != null) {
                    if ($idLista != null) {
                        if ($idProducto != null) {
                            $deseo = new \Web\wishBundle\Entity\deseo();
                            $deseo->setFechaCreacion($fecha);
                            $deseo->setNivel($nivel);
                            $deseo->setLista($lista);
                            $deseo->setProducto($producto);

                            if($latitud != null && $longitud != null && $precio != null){
                                $ubicacion = new \Web\productBundle\Entity\ubicaciones();
                                $ubicacion->setLatitud($latitud);
                                $ubicacion->setLongitud($longitud);
                                $ubicacion->setPrecio($precio);
                                $ubicacion->setProducto($producto);

                                $em->persist($ubicacion);
                            }

                            $em->persist($deseo);
                            $em->flush();
                            $arrayDeseo = $this->encodeDeseo($deseo);
                            $arrayResponse = array("Deseo" => $arrayDeseo);
                            return $this->successResponse($arrayResponse);
                        } else {
                            return $this->failureResponse(EMPTY_IDPRODUCTO, "La peticion no contiene el campo 'producto'");
                        }
                    } else {
                        return $this->failureResponse(EMPTY_IDLISTA, "La peticion no contiene el campo 'lista'");
                    }
                } else {
                    return $this->failureResponse(PRODUCT_NOT_FOUND, "No se ha encontrado el producto al que se hace referencia");
                }
            } else {
                return $this->failureResponse(WISHLIST_NOT_FOUND, "No se ha encontrado la lista a la que se hace referencia");
            }
        } else {
            return $this->failureResponse(EMPTY_PARAMETER, "La peticion no contiene valores");
        }
    } else {
        return $this->failureResponse(NO_POST_REQUEST, "La peticion no es POST");
    }
}
```

Por cada acción que es requerida por la aplicación existe un método que responde directamente al servidor mediante una petición.

Todos estos métodos responden a una serie de archivos de enrutamiento escalonados en Symfony gracias a los cuales asociamos todas las acciones de los controladores a una dirección http.

Gracias a esto la dirección http://server.com/peticiones-moviles/productos/listar-productos/

sabe que método tiene que ejecutar.

Primer Archivo:

```yaml
mobile:
    resource: "@mobileBundle/Resources/config/routing.yml"
    prefix:   /peticiones-moviles
```

Segundo Archivo:

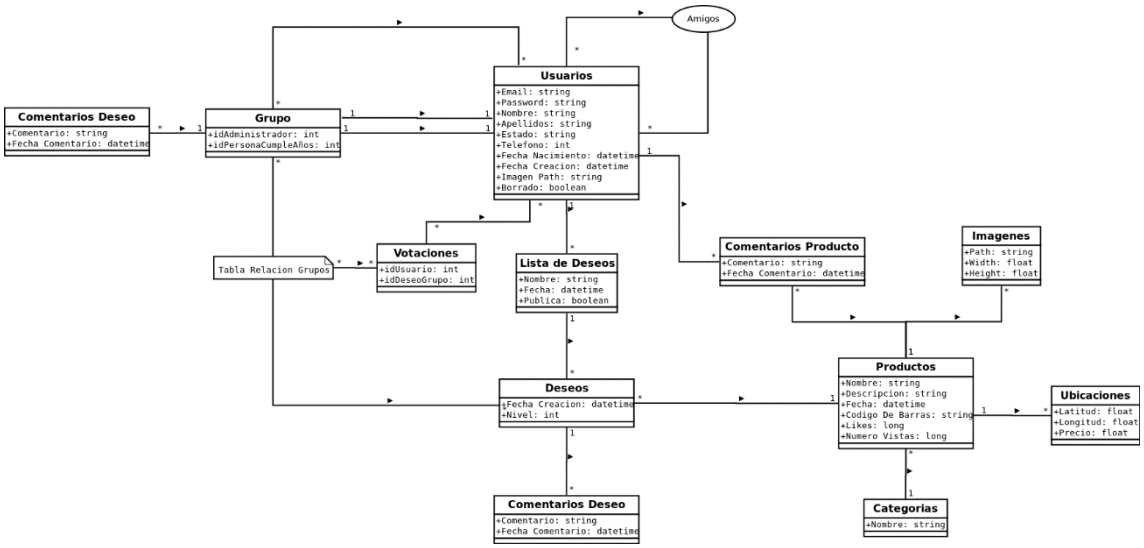
```yaml
mobile_productos:
    prefix:     /producto
    resource: "@mobileBundle/Resources/config/routing/producto.yml"

mobile_categoria:
    prefix:     /categoria
    resource: "@mobileBundle/Resources/config/routing/categoria.yml"
```

Tercer Archivo:

```yaml
producto_listar_productos:
    path: /listar-productos/
    defaults: { _controller: mobileBundle:Producto:listarProductos }

producto_listar_productos_categoria:
    path: /listar-productos-categoria/
    defaults: { _controller: mobileBundle:Producto:listarProductosCategoria }

producto_crear_comentario_producto:
    path: /nuevo-comentario-producto/
    defaults: { _controller: mobileBundle:Producto:crearComentarioProducto }

producto_listar_comentarios:
    path: /listar-comentarios/
    defaults: { _controller: mobileBundle:Producto:listarComentariosProducto}

producto_crear_ubicacion_producto:
    path: /nueva-ubicacion-producto/
    defaults: { _controller: mobileBundle:Producto:crearUbicacionProducto }

producto_listar_ubicaciones:
    path: /listar-ubicaciones/
    defaults: { _controller: mobileBundle:Producto:listarUbicacionesProducto}
```

Diseño de la base de datos

La base de datos ha sido implementada como una base de datos orientada a objetos y sigue el siguiente esquema UML donde podemos ver todas las relaciones que existen en la base de datos.



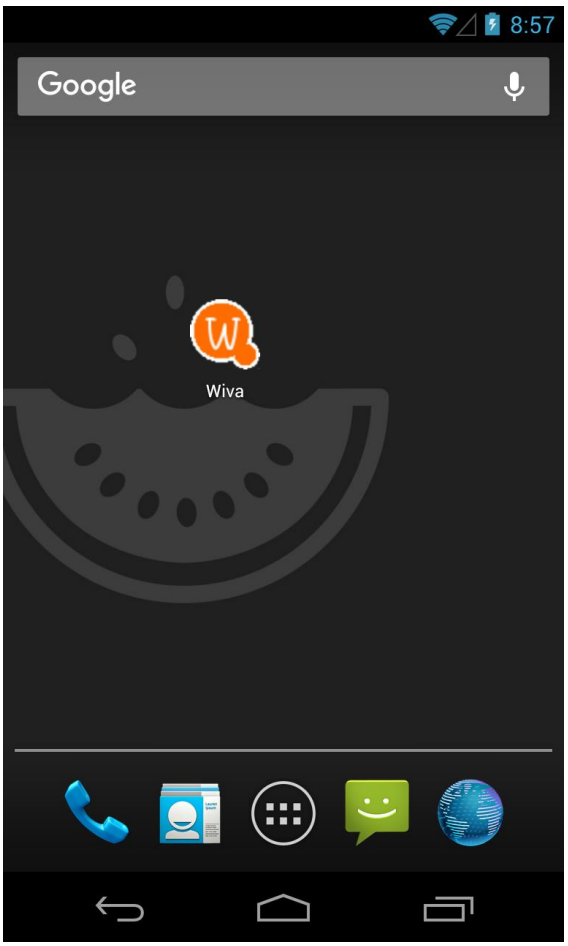
Como se puede ver en el diagrama se crearían las siguientes tablas mediante el comando de Symfony2 "$ php app/console doctrine:schema:update --force" ya que como hemos explicado en la sección anterior, las entidades de nuestro servidor contienen las correspondientes anotaciones orientando así los objetos a las base de datos y pudiendo generarse todas la estructura según nuestro código.

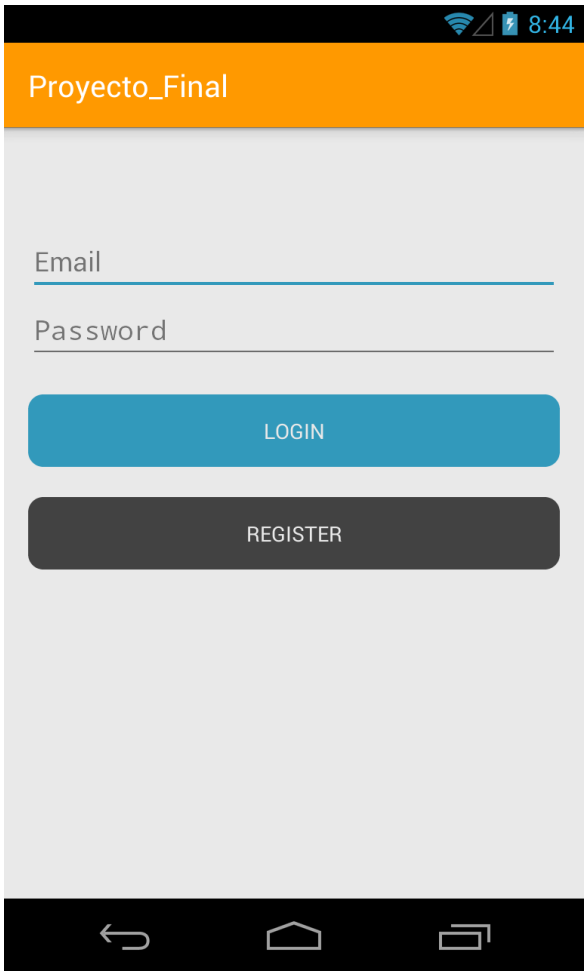
amigos	Tabla resultante de la relación recursiva de muchos a muchos entre la tabla usuarios.
categoria	Tabla que almacena las categorías de los productos y su relación es de uno a muchos con la tabla producto. Cada producto lleva una categoría asignada.
comentarios_deseo	Esta tabla almacena todos los comentarios que se hacen a un deseo, este deseo puede ser tanto propio como de un amigo. Está relacionada con la tabla deseo de muchos a uno y también aunque no esté indicado en el diagrama con una relación de muchos a uno con la tabla usuario.
comentarios_producto	Esta tabla almacena todos los comentarios que se hacen a un deseo, este deseo puede ser tanto propio como de un amigo. Está relacionada con la tabla deseo de muchos a uno y también aunque no esté indicado en el diagrama con una relación de muchos a uno con la tabla usuario.

deseo	La tabla deseo almacena deseos que están relacionado con una lista de deseos, cada deseo está relacionado de muchos a uno con la tabla producto, en la cual se almacenan todos los detalles del objeto en sí, haciendo que distintos deseos de distintas personas puedan apuntar al mismo producto.
imagen	Esta tabla almacena las rutas de las imágenes de cada producto, además de almacenar también el ancho y el alto de la imagen que puede sernos útil a la hora de hacer uso de ella.
lista_deseo	Esta tabla almacenara todas las listas de deseos de cada usuario, está directamente relacionado con el usuario a la que pertenecen.
producto	Esta tabla almacena todos los productos, estos se crean cuando hacemos un deseo nuevo sin tomar como base un producto ya existente, intentamos limitar las duplicidades de productos mediante el código de barras que se escanea mediante la aplicación, ya que queremos que los productos sean únicos.
ubicaciones	Esta tabla almacena las coordenadas de un producto y se ha separado en una tabla independiente ya que queremos que cada producto pueda tener varias localizaciones con precios distintos asignados, para así comparar y buscar el mejor precio.
usuario	La tabla usuario almacena los distintos usuarios y está relacionada con los grupos y consigo misma en el caso de la tabla de amigos.
usuarios_grupos	La tabla grupo almacenara el grupo que se crea automáticamente cuando se acerca el cumpleaños de un usuario, tiene varias relaciones con la tabla usuario ya que debe contener el usuario que cumple años, automáticamente se agregaran también todos los usuarios que son amigos y un usuario que será el administrador del grupo.

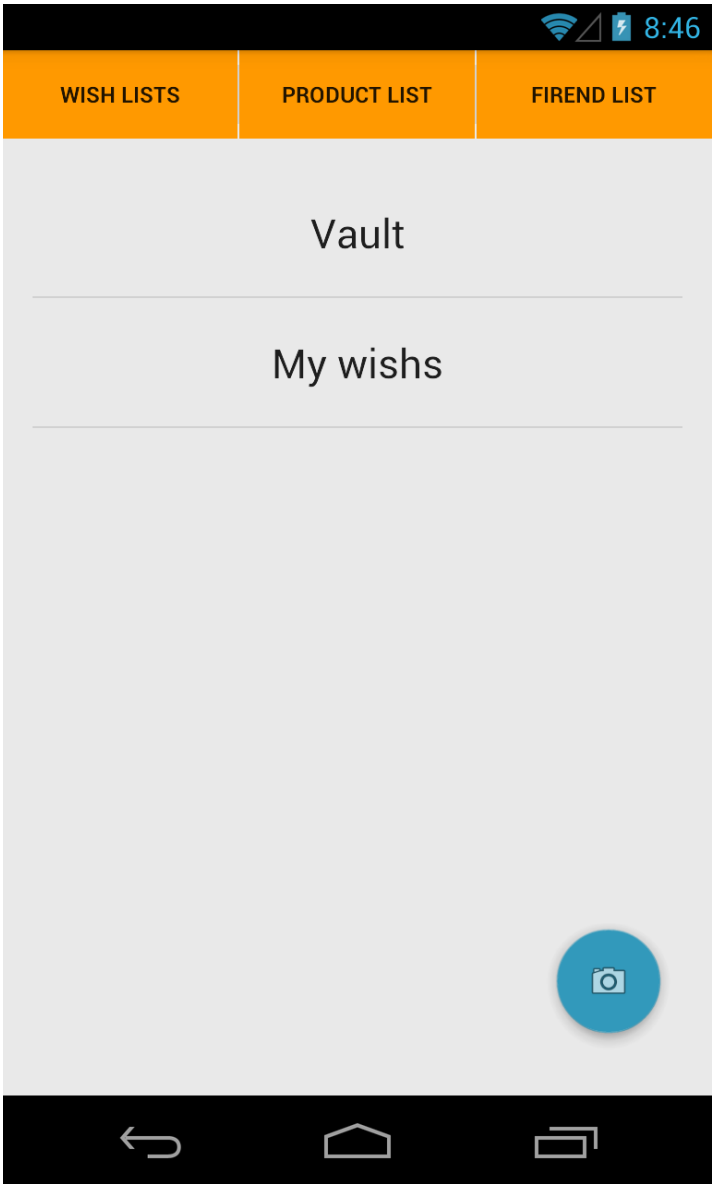
Diseño de las interfaces

La primera interfaz que se abre al iniciar la aplicación es una página de login. En ella el usuario debe introducir su email y su contraseña para poder acceder su cuenta. También si es un nuevo usuario podrá registrarse pulsando en el botón de registro, el cual lo llevará a un formulario para crear su cuenta. La cuenta se crea al momento, así que no hay que esperar para disfrutar de la aplicación.

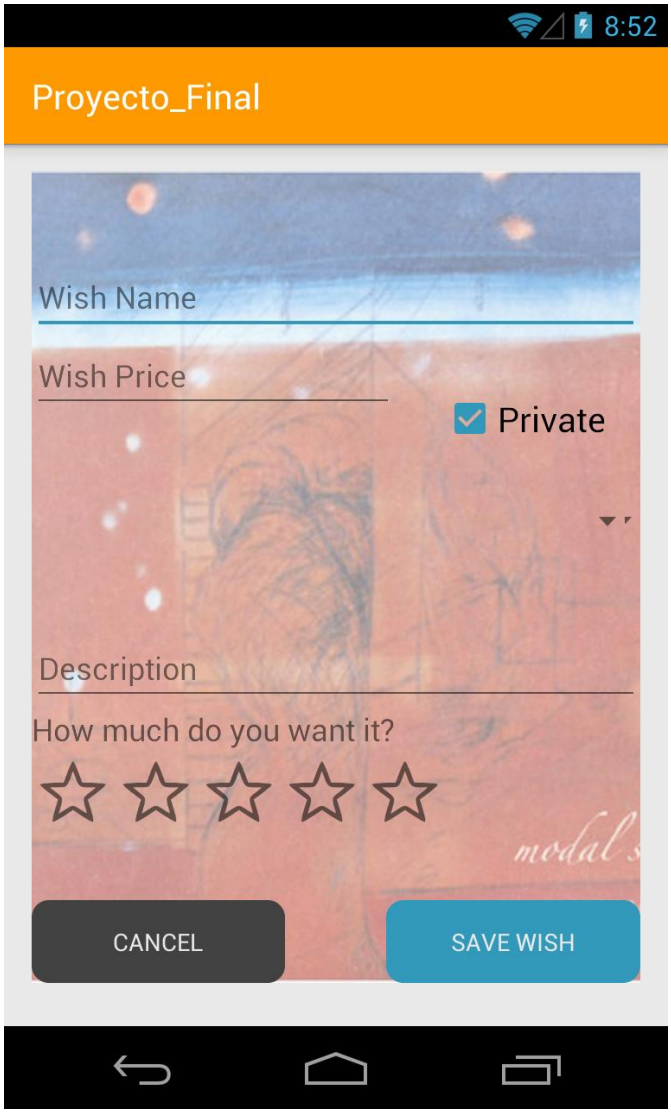




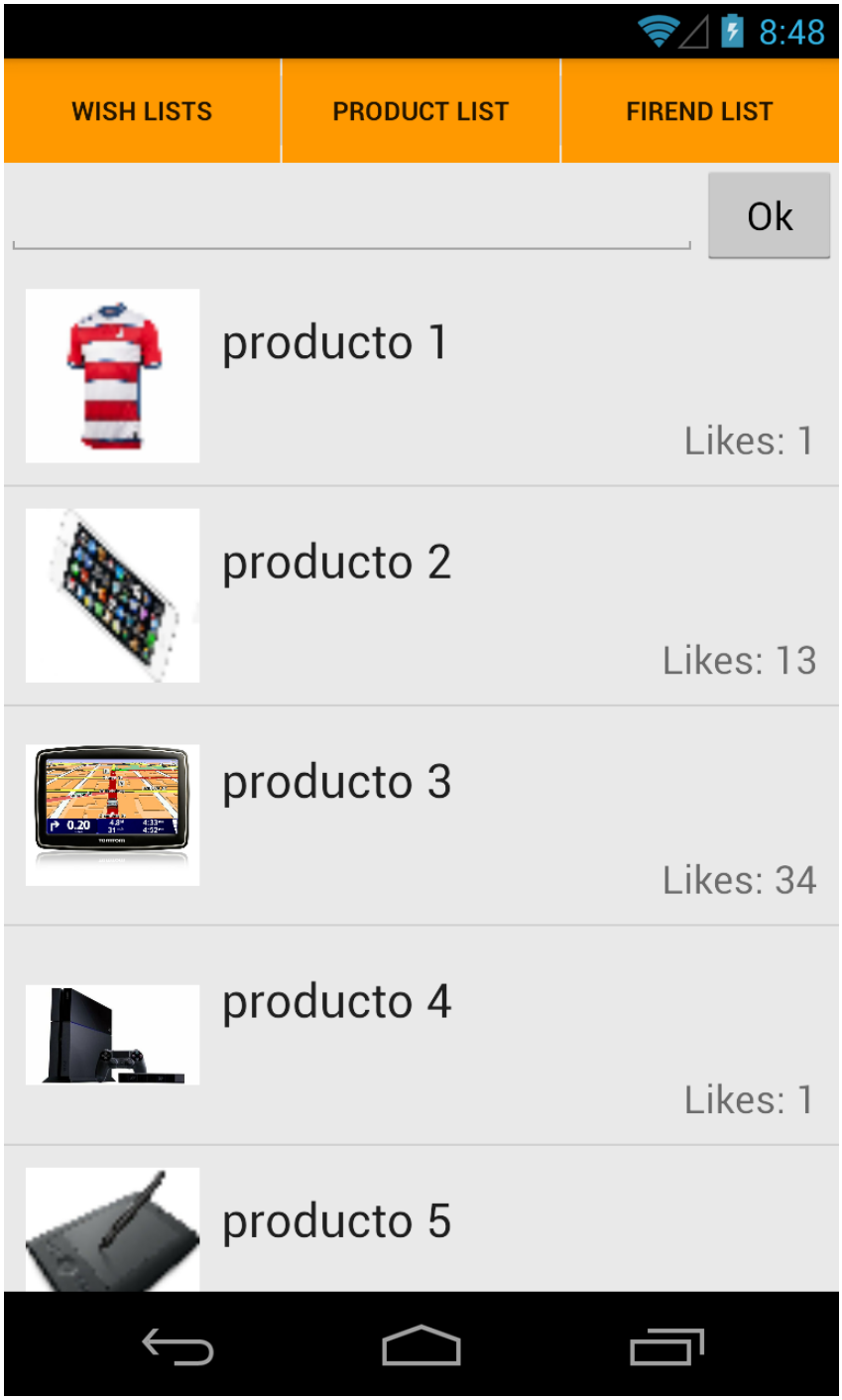
Tras logearse, el usuario verá una página con 3 pestañas. En la primera, la cual se muestra por defecto, contiene la lista de deseos del usuario. El usuario podrá navegar por sus listas de deseos y ver los deseos que ha ido guardando. También podrá modificar algunos aspectos del deseo, como el nivel de deseo.



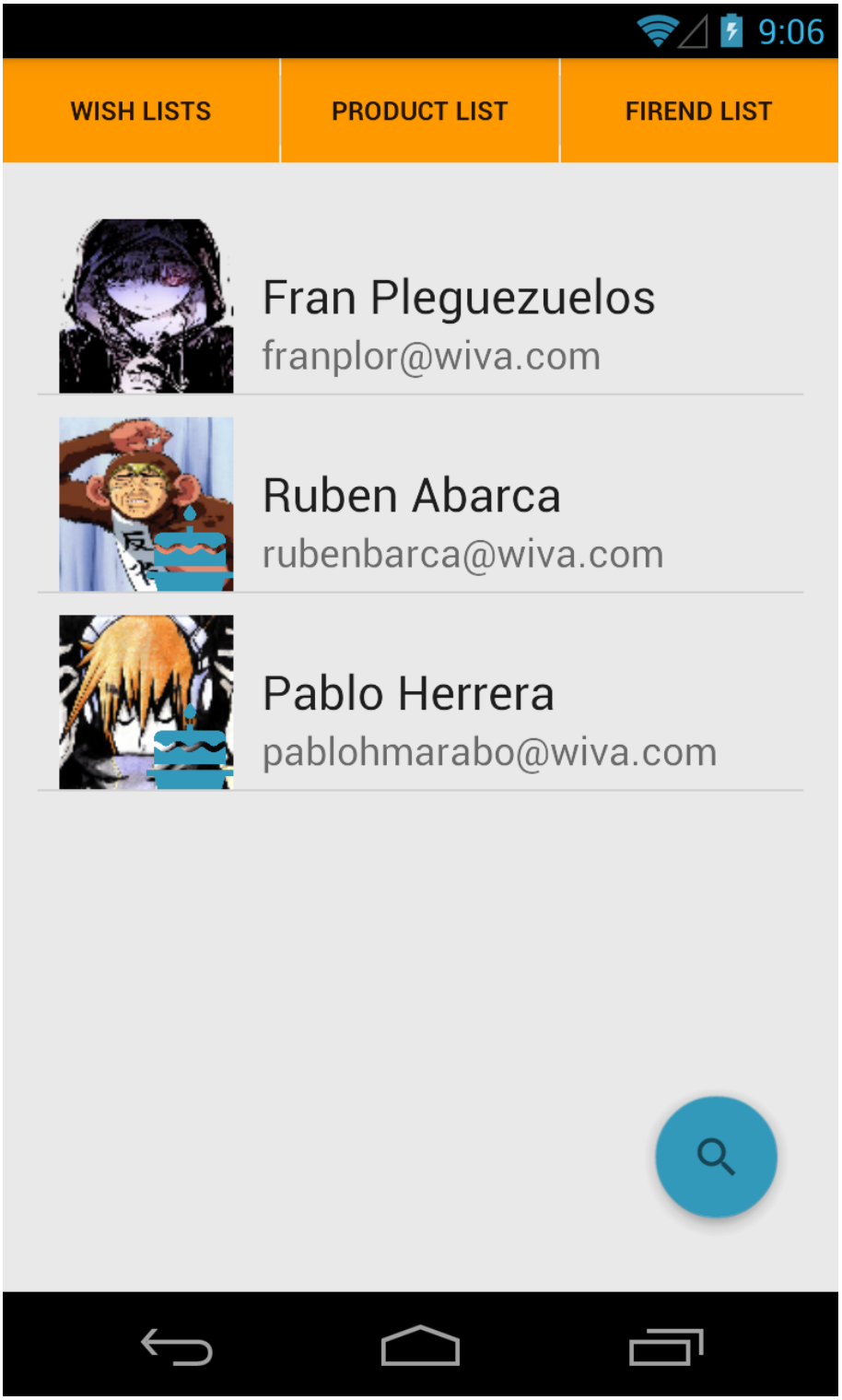
Esta pestaña posee un botón flotante el cual permite crear un deseo ya sea tomando una foto desde la cámara o abriendo una imagen de la galería. Tras tomar la foto o seleccionar la imagen de la galería se abrirá un formulario para rellenar los datos del deseo. Una vez completos el deseo se generará en la lista indicada y el producto será subido a la base de datos de todos los productos.



En la segunda pestaña aparece una lista con todos los productos que todos los usuarios han creado como deseo. Al abrir un producto podrán verse sus detalles y podrá ser añadido a una de las listas de deseos del usuario. Dentro existe un botón el cual abre un mapa con las localizaciones almacenadas del producto. También podrán añadirse datos extra como precio o localizaciones donde hayas visto tal producto para compartirlo con todo el mundo. Además, podrán comentarse los productos para poder dejar tus opiniones sobre el producto. Esta pestaña contiene un buscador para buscar productos.

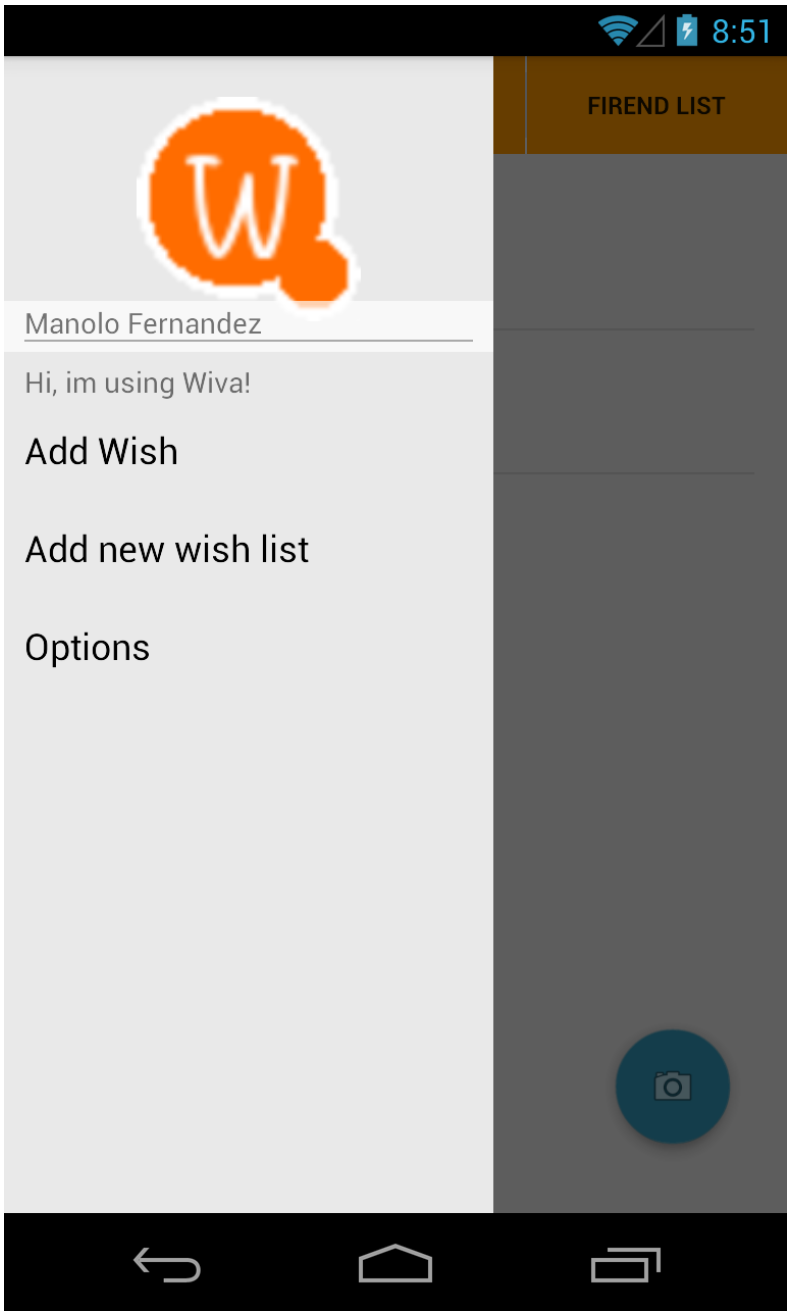


En la tercera pestaña se podrá ver los amigos del usuario. Al seleccionar uno, se abrirá una pantalla con los detalles del usuario, y sus listas públicas de deseos. Esta lista posee una funcionalidad extra que avisa cuando el cumpleaños de dicho amigo este a menos de una semana con un icono en su imagen de perfil, tanto en la lista como en los detalles del amigo.

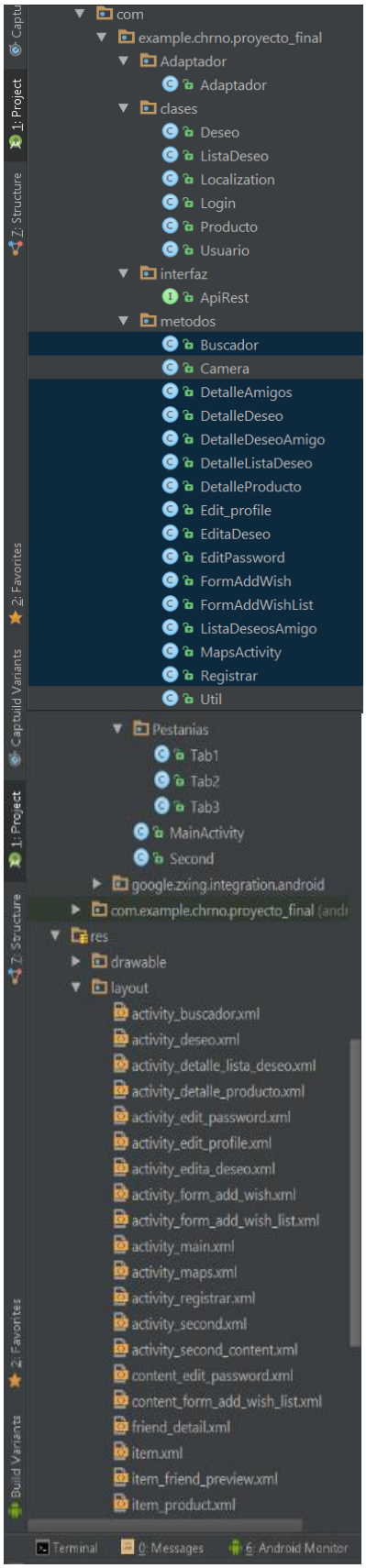


Esta pestaña también posee un icono flotante el cual abre un buscador de usuarios. Introduciendo una cadena en el buscador se buscarán entre todos los usuarios aquellos cuyo email contenga la cadena deseada. Una vez encontrado al usuario deseado, se podrá enviar una solicitud de amistad. Una vez el otro usuario la acepte os habréis registrado como amigos de tal forma que podréis acceder a las listas de deseos públicas de ambos. Cuando desees borrar a alguien de tu lista de amigos podrás eliminarlo a través del menú contextual.

En cualquiera de las pestañas el usuario puede abrir un menú deslizante situado en la izquierda. Desde este menú se pueden añadir deseos, añadir nuevas listas de deseos, editar tu perfil y manejar las peticiones de amistad.



Clases



Adaptador
Contiene un switch con el cual diferenciamos que listview cargar.

Buscador
Clase que contiene el algoritmo para buscar. Tambien utilizamos un switch para diferenciar si buscamos un producto o a un usuario.

ListaDeseosAmigo
La lista de deseos de una de las listas de deseos del usuario registrado como amigo

DetalleDeseo
Muestra los datos del deseo seleccionado. (Editables)

DetalleDeseoAmigo
Muestra los datos del deseo seleccionado de un amigo (No editables)

DetalleListaDeseo
Muestra la previsualizacion del deseo en una lista

DetalleAmigos
Muestra los datos del usuario seleccionado

DetalleProducto
Muestra los datos del producto seleccionado.

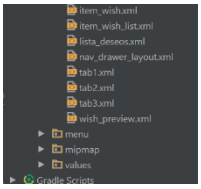
Edit_Profile
Permite modificar algunos campos del usuario

EditPassword
Permite cambiar la contraseña

FormAddWish
Formulario para crear un añadir un nuevo deseo

Registrar
Formulario para crear una cuenta nueva

Deseo
Clase POJO de un deseo



ListaDeseo

Clase POJO de una lista de deseos

Localization

Clase POJO de la localización y el precio de un producto

Producto

Clase POJO de un producto

Usuario

Clase Pojo de un usuario

ApiRest

Clase de retrofit

Camera

Clase para manejar la cámara

Util

Clase que contiene métodos genéricos usados por toda la aplicación

Tab1

Pestaña que contiene las listas de deseos

Tab2

Pestaña que contiene los productos

Tab3

Pestaña que contiene la lista de amigos del usuario

MainActivity

Clase encargada del login

Second

Clase contenedora de las pestañas

Anotaciones

Debido a que el servidor no ha podido subirse a un hosting online gratuito ya que ninguno ofrece acceso ssh, para configurar correctamente la caché de symfony y el entorno de producción, hemos implementado datos de prueba en la parte de android para que pueda relizarse una pequeña demo sin necesidad de la base de datos. Los métodos para acceder al servidor siguen en el código aunque comentados para que no falle el proyecto.

Futuras actualizaciones:

Grupos

Tenemos pensado implementar un sistema de grupos, que se crearían automáticamente dos semanas antes del cumpleaños.

Todos los amigos del usuario recibirían una petición para entrar en dicho grupo.

Una vez dentro del grupo los amigos podrán votar el precio máximo del producto, una vez votado el precio los amigos podrán elegir de entre todos los deseos del usuario con un precio menor o igual al votado.

Finalmente los usuarios se tiene que poner de acuerdo en el chat del grupo para comprar el producto elegido.

Peticiones amigos

Mejorar las peticiones de amistad, implementando notificaciones.

Bibliografía

- Stack Overflow
- http://square.github.io/retrofit/
- https://symphony.com/