UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Título del Proyecto:

Controlador de alimentos

Autor:

Ibáñez Paricio, Miguel

Director:

Tomás Gironés, Jesús

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles

Septiembre del 2015

Contenido

Introducción

Descripción del problema

La problemática que mi proyecto final de máster aborda, viene dada por una situación cotidiana que me encontraba a menudo en mi día a día.

Dicho problema surge cuando me disponía a realizar la compra semanal. En muchas ocasiones encontraba que en las grandes cadenas de supermercados, los alimentos frescos como carne, leche, huevos o pescado se venden en packs con cantidades pensadas para familias más grandes o con ofertas que premian a la cantidad comprada en lugar de descuentos unitarios, esto, esto se añade a la dificultad de comprar en tiendas tradicionales o mercados municipales debido a que los horarios de apertura no se ajustan normalmente a mi jornada laboral. Todo ello hace que en ocasiones sea difícil consumir todos los alimentos frescos a tiempo antes de su fecha de caducidad, como consecuencia acabo o tirando comida o guardando parte de ella en el congelador y que acabo olvidando que tengo almacenada.

Tras plantearme el desarrollo de una herramienta para gestionar el stock de comida como mi proyecto final de máster, decidí investigar la situación global de el desperdicio de alimentos para ver si realmente sería interesante/rentable desde un punto de vista de negocio desarrollar una herramienta de estas características.

La Comisión Europea estima que cada año se desaprovechan en el mundo, más de 1.300 millones de toneladas de alimentos, es decir, 1/3 de la producción mundial, de los que 89 millones de toneladas de comida en buen estado corresponden a la Unión Europea.

A continuación se indican algunos datos más específicos sobre las pérdidas y desperdicio alimentario en el ámbito de la UE:

179 kilos por habitante de alimentos desperdiciados, y ello sin contar los de origen agrícola generados en el proceso de producción ni los descartes de pescado arrojados al mar.

170 millones de toneladas equivalentes de CO2 al año.

Entre un 30% y un 50% de los alimentos sanos y comestibles a lo largo de todos los eslabones de la cadena agroalimentaria hasta llegar al consumidor que podrían ser aprovechables se convierten en residuos.

En los hogares, el desperdicio alimentario alcanza el 42% del total, en la fase de fabricación el 39%, en la restauración el 14% y en la distribución el 5%.

España es el séptimo país que más comida desperdicia (7,7 millones de t), tras reino Unido (14,4 millones de toneladas) Alemania (10,3 millones de toneladas), Holanda (9,4 millones de

toneladas) Francia (9 millones de toneladas) Polonia (8,9 millones de toneladas) e Italia (8,8 millones de toneladas).

En cuanto al ámbito de los hogares, el estudio publicado en 2013 por la Confederación Española de Cooperativas de Consumidores y Usuarios (HISPACOP) y avalado por el Instituto Nacional de Consumo (INC), ha señalado entre otros datos que el desperdicio medio por hogar (2,7 personas de media) es de 1,3 kg/semana o 76 kg/año, lo que equivale a más de medio kg de alimentos por persona y semana. Así, los hogares españoles tiran en un año 1,5 millones de toneladas de alimentos que son válidos para el consumo.

Además de estos datos a nivel europeo, estos son los países que más comida desperdician en todo el mundo: EEUU, Australia, Reino Unido, Alemania, Canadá, Noruega, Malasia, Holanda, Francia y China.

Con todos los datos propuestos, el siguiente paso para decidir sobre la necesidad/viabilidad del proyecto es ver cuál es el grado de penetración de nuevas tecnologías, y más concretamente el uso de *"smartphones"* en los países citados.

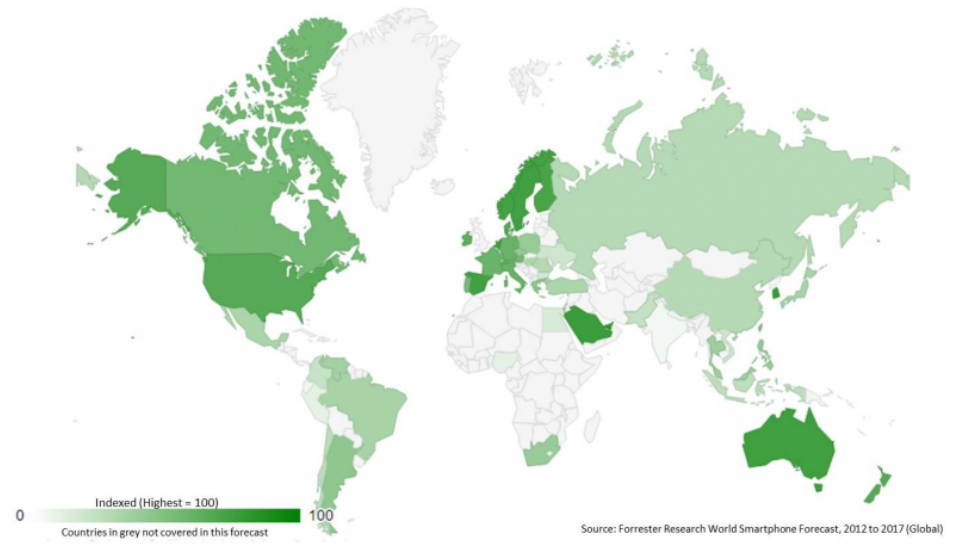


Ilustración 1 Grado de penetración de smartphones en el mundo

Es fácil observar que los países que más alimentos desperdician, son también de los que más grado de penetración de *"smartphones"* tienes, casi en todos de más del 50%, y si nos centramos en los usuarios menores de 40 años, más de un 80%. Por lo tanto cabe pensar que el sistema de control de stock de alimentos puede tener un numero de usuarios potenciales bastante alto.

Objetivos

Desarrollo de la primera versión de una aplicación de gestión de stock de alimentos. Las plataformas soportadas en la primera versión serán Android e iOS, ambas en forma nativa. Se pretende que la aplicación sea lo más fácil y cómoda de usar posible, que tenga una serie de funcionalidades suficientes para una primera versión y que sirva de punto inicial de un proyecto de más envergadura con más tipos de funcionalidades y, a ser posible con integración de comercios u otro tipo de agentes de financiación y monetización.

Motivación

Las motivaciones son tanto personales como profesionales. En el ámbito personal, la motivación es crear una aplicación que inicialmente ayude a particulares y a pequeños negocios de restauración a controlar el stock de alimentos del que disponen con el objetivo de no desperdiciar comida. En el plano profesional, a parte de plasmar los conocimientos adquiridos durante el máster, realizar una primera aplicación completa fuera de la empresa, con libertad de creación y con opción de comercialización por mi cuenta.

Modelo de negocio

Descripción de la aplicación v 1.0

La aplicación permitirá al usuario introducir el nombre y la fecha de caducidad de los productos que ha adquirido. Podrá consultar los productos que tiene en stock, así como buscar recetas con los mismos de una manera sencilla. La aplicación notificará con de manera automática y con antelación (configurable en Android, 1 día en iOS) al usuario que tiene alimentos que van a caducar. Además, el usuario podrá consultar en un mapa interactivo los establecimientos de compra de alimentos más cercanos a su posición (configurable en Android, 10 km en iOS)

La primera versión de la aplicación está orientada a usuarios particulares y a pequeños establecimientos de restauración.

Segmentos de clientes

El cliente potencial de esta aplicación en un principio es cualquier persona que disponga de un Smartphone, y que realice compras de productos de alimentación de manera regular. Con más detalle, los principales usuarios se encontrarán entre la franja de edad entre 30 y 65 años, serán usuarios concienciados con la desigualdad en el acceso a alimentos, concienciados en el ahorro energético y el medio ambiente.

Otro potencial segmento de usuarios son los pequeños restaurantes, que con la aplicación podrían realizar un control sencillo de su stock de alimentos.

Posición de valor

El objetivo es ofrecer una aplicación sencilla de usar con la cual se pueda gestionar de manera sencilla y eficiente el consumo de alimentos de un hogar o un pequeño restaurante, ofreciendo así un ahorro en gasto, un ahorro en desperdicio de comida y gestión de residuos y una sensación de ayuda y concienciación con el medio ambiente. Como aliciente, se puede ofrecer a los restaurantes la posibilidad de realizar descuentos, si tanto el restaurante como el cliente son usuarios de la aplicación, con el objetivo de atraer clientes a los restaurantes y usuarios a la aplicación. Este caso se puede extender a establecimientos de venta de productos alimenticios y así fortalecer la relación entre usuarios personales y corporativos de la aplicación, además de fidelizar al cliente.

Canales de venta

El principal canal de venta serían las tiendas de aplicaciones Google Play y App Store, se considerará la publicitación de la aplicación en redes sociales y organismos y colectivos de concienciación sobre el consumo alimenticio.

Relación con el cliente

En la primera versión la relación con el cliente se basará en la mejora y corrección de la aplicación en base a los comentarios que los usuarios propongan en los canales de venta comentados anteriormente.

Ingresos

Los ingresos de la primera versión se centrarán en la publicidad dentro de la aplicación y en una posible versión sin anuncios de pago pero (1€ aprox.), el objetivo de esta primera versión es la obtención de usuarios y observar las demandas que tienen sobre el producto.

Resursos Clave

Como recursos clave dispongo de la experiencia en desarrollo de aplicaciones móviles adquirida durante los últimos años, se estudiará la posibilidad de contratar servicios de algún diseñador o la compra de recursos gráficos (iconos, fondos, etc..) que hagan la interfaz de la aplicación más atractiva.

Actividades Clave

- Desarrollo robusto, sencillo de usar y atractivo visualmente.
- Dar a conocer la aplicación mediante acciones en redes sociales o publicidad.
- Hacer hincapié en el aspecto social de la aplicación (presentación para colectivos de consumo responsable)
- Soporte post-publicación, corrección de bugs y mejora continua.
- Explotar la relación Coste aplicación(gratuita con publicidad, un euro sin publicidad)/Posible ahorro en gasto de alimentación
- Fortalecer la relación usuarios personales-usuarios profesionales de la aplicación, mediante descuentos y otras promociones.

Colaboradores clave

Como colaboradores clave, se buscaría lo siguiente

- Expertos en marketing digital, para asesoramiento sobre publicidad en redes sociales, etc..
- Restaurantes de pequeño tamaño dispuestos a colaborar con la aplicación.
- Establecimientos y asociaciones pro consumo responsable.

Estructura de costes

Fijos:

- Conexión a internet.

Equipos:

- Ordenador iMAC para desarrollo Android e iOS.
- Teléfonos iPhone y Android para pruebas y demostraciones.

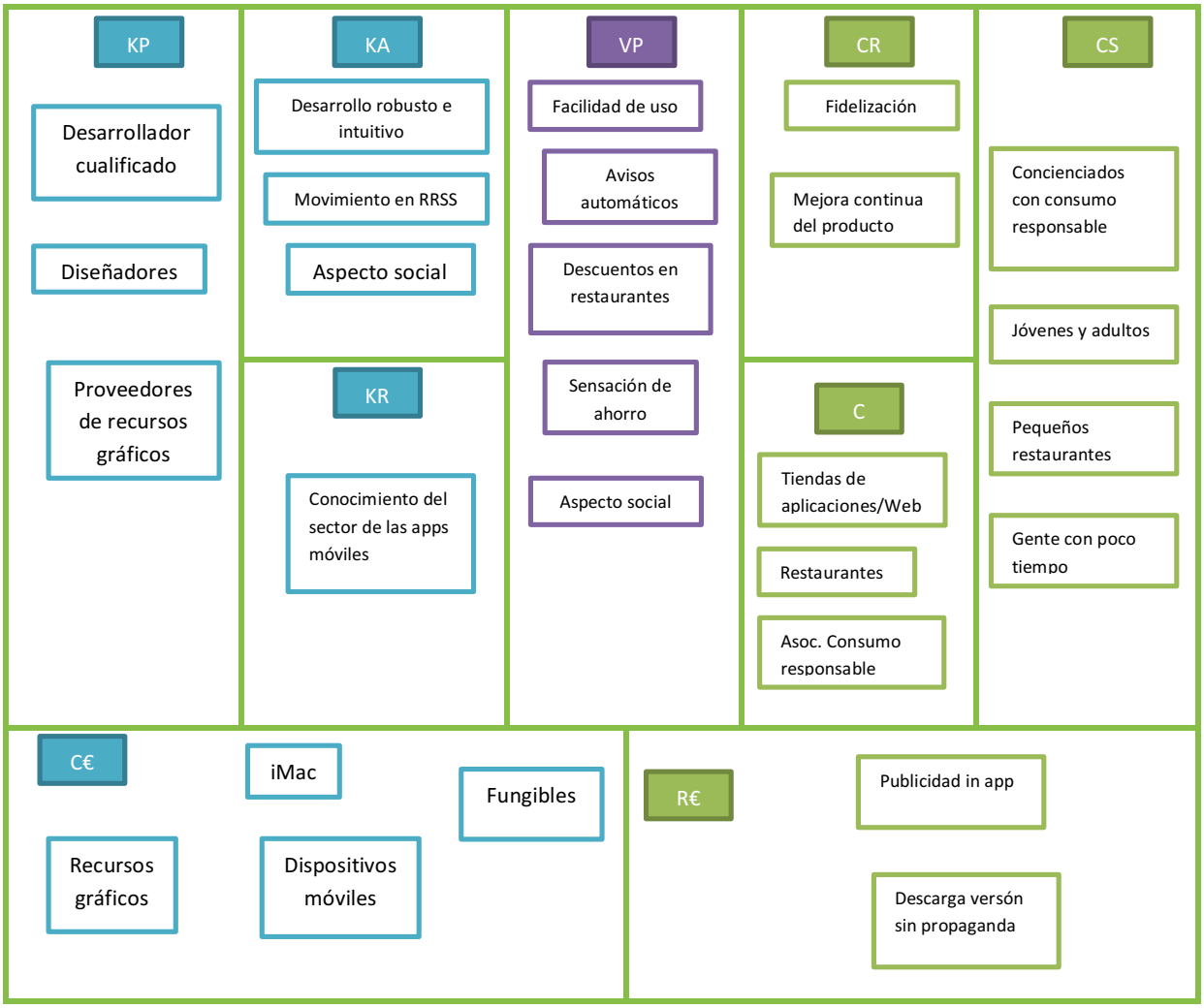
Fungibles:

- Material de oficina en general.

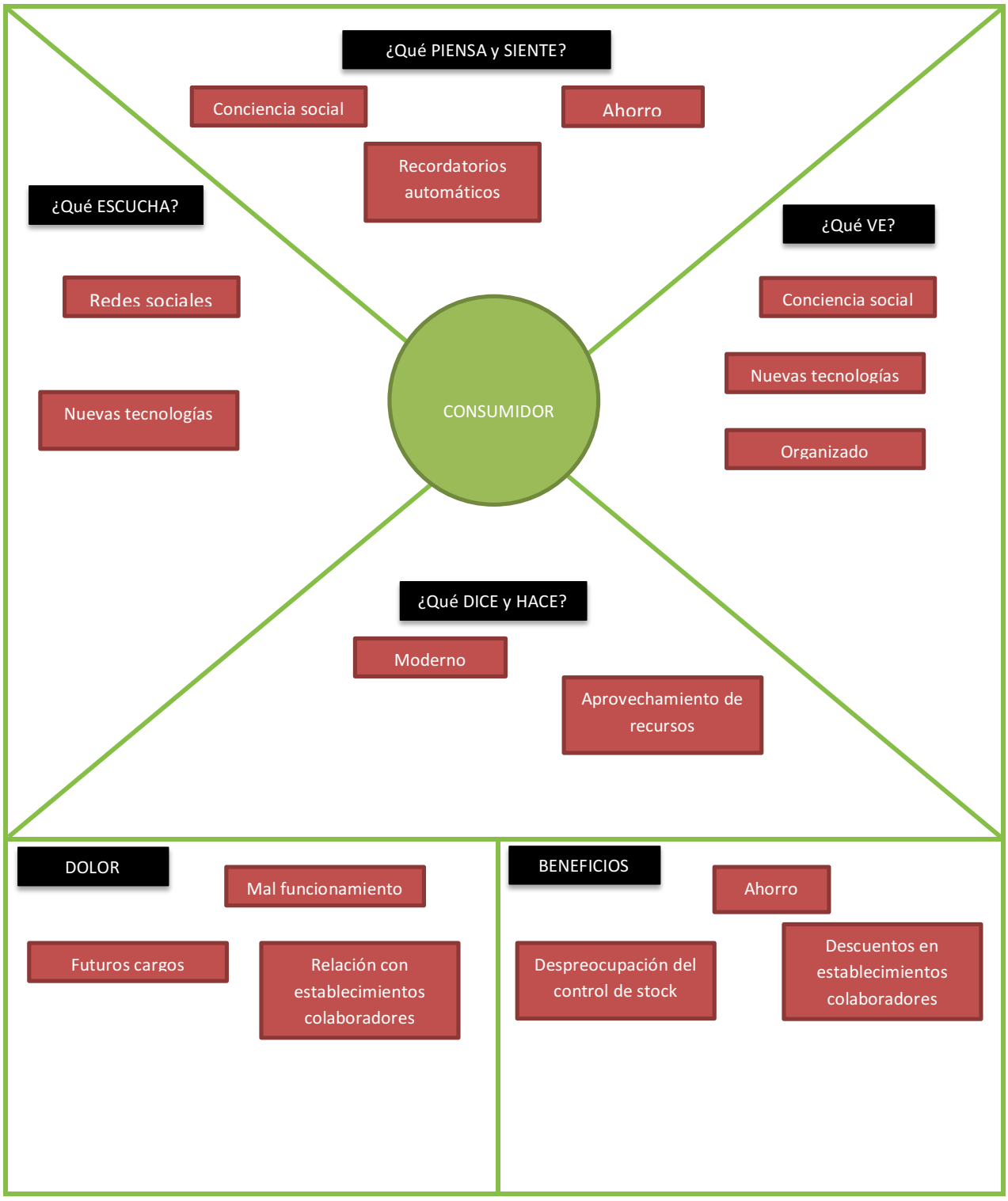
Subcontrataciones:

- Diseño de interfaz o compra de recursos online (iconografía etc..)
- Publicidad en canales online.

Lienzo del modelo de negocio



Mapa de empatía del consumidor

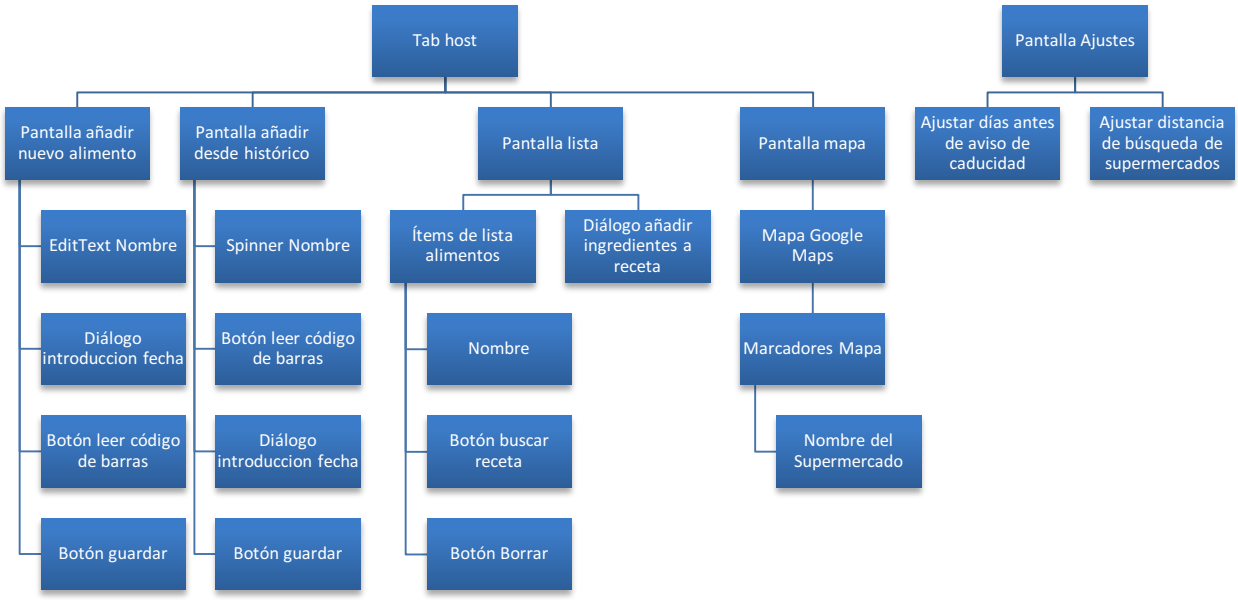


Arquitectura de la aplicación

Esquema del diseño

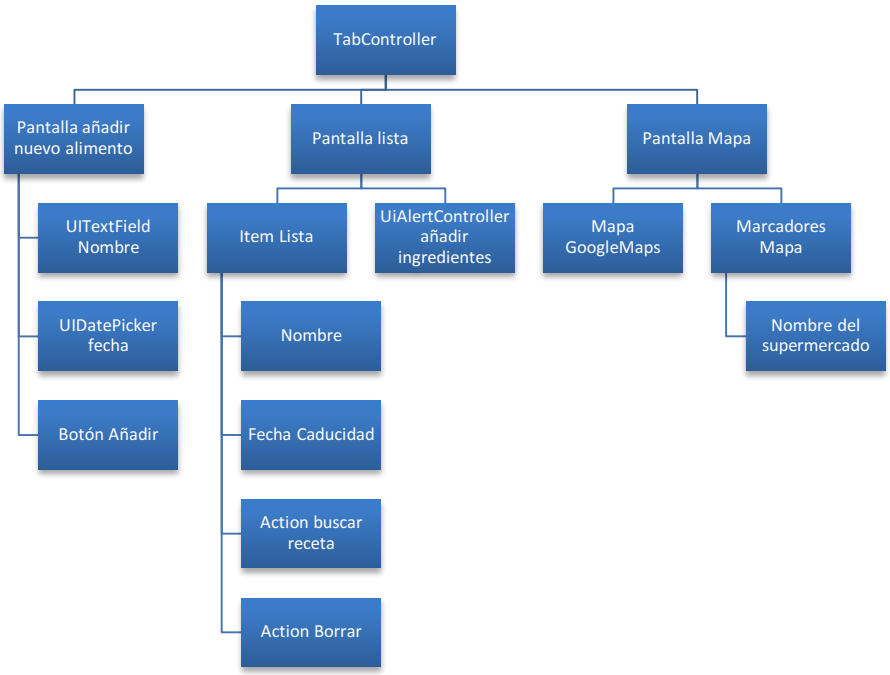
La base del diseño tanto para la aplicación de Android como iOS ha sido un esquema en forma de tabs, con una pantalla base sobre la que se cargan las diferentes sub-pantallas de la aplicación.

Esquema ANDROID



El tab host responde al componente ViewPager de Android, sobre el que se cargan el resto de pantallas en forma de fragment, la navegación entre pantallas se realiza mediante slide horizontal, el tab host incluye una barra indicadora con el nombre de la pantalla que estamos visualizando en cada momento. El acceso a la pantalla de ajustes se hace desde el típico icono de 3 puntos incluído en el toolbar que apareció en Material Design y que está disponible en el resto de versiones gracias a la librería de compatibilidad AppCompat. Para la pantalla de preferencias se ha optado por una Preferece Activity, que carga el contenido de las preferencias desde un fichero xml. Los avisos de funcionamiento de la aplicación se hacen mediante Toast. El diseño de las notificaciones es el típico de el SO Android, con un icono de la aplicación y un texto explicativo.

Esquema iOS



El esquema de iOS es similar al de Android, sin la pantalla de añadir desde histórico y la de preferencias. La navegación en este caso se realiza mediante la barra inferior del Tab Controller, al que se han añadido los nombres e iconos representativos de cada pantalla. Los mensajes que envía la aplicación al usuario se gestionan mediante UIAlertController. Para realizar la pantalla del mapa se ha tenido que agregar el sdk de Google Mas para iOS siguiendo las instrucciones que se encuentran en la página oficial de Google Maps. Se ha optado por Google Maps por ser una herramienta ya conocida de Android, pero se estudiará para versiones posteriores usar el SDK nativo de iOS, ya que al incluir el SDK de Google Maps el tamaño del proyecto y a consecuencia de la aplicación ha crecido considerablemente. Las notificaciones, al igual que en Android siguen el esquema estándar de diseño, con el icono de la aplicación y el texto explicativo

Modelo de datos

Para la primera versión se ha optado por un esquema de base de datos sencillo y basado en SQLite.

Android

Tabla actual, tabla histórico:

TEXT PRIMARY KEY	LONG	TEXT
nombre	date	barcode

La base de datos en Android cuenta con 2 tablas, con los mismos campos. Esto es así porque con la tabla actual se gestiona el stock acual de alimentos, se inserta una nueva fila cuando se añade un alimento y no existe uno con el mismo nombre (en este caso se avisará al usuario) y se eliminará cuando se borre el alimento en la pantalla de lista. Respecto a la tabla de histórico se usa para rellenar el spinner de la pantalla de añadir desde histórico, facilitando así al usuario la introducción de alimentos que ya había tenido anteriormente en su stock, la adición de alimentos a la misma se produce cuando el usuario introduce un alimento, si este no está en la tabla histórico . El campo date guarda el timestamp en milisegundos de la fecha de caducidad, esto es así porque a la hora de añadir las alarmas que nos avisarán de la próxima caducidad de los alimentos es más fácil trabajar con las fechas en formato timestamp. El campo barcode es opcional, y se rellena leyendo el código de barras de cada producto mediante la cámara (requiere tener instalada la aplicación barcode scanner), se usa para poder añadir desde histórico un alimento leyendo su código de barras. Para el uso de la base de datos se ha implementado el esquema SQLiteOpenHelper visto durante el curso en el cual se han implementado los siguientes métodos.

```
   public void guardarAlimentoHistorico(String nombre, long fechaCaducidad, String codigo ){
      try {
         SQLiteDatabase db = getWritableDatabase();
         db.execSQL("INSERT INTO historico VALUES ('"+nombre+"'
,"+fechaCaducidad+",'"+codigo+"')");
      } catch (SQLException e) {
         e.printStackTrace();
      }


   }

   public int guardarAlimentoActual(String nombre, long fechaCaducidad, String codigo) {
      try {
```

```
            SQLiteDatabase db = getWritableDatabase();
            db.execSQL("INSERT INTO actual VALUES ('"+nombre+"'
,"+fechaCaducidad+",'"+codigo+"')");
            return 1;
         } catch (SQLException e) {

            e.printStackTrace();
            return 0;
         }
      }

      public ArrayList<String> getHistoricoAlimentos(){
         ArrayList<String> alimentos = new ArrayList<String>();
         try {
            SQLiteDatabase db = getWritableDatabase();
            Cursor c =  db.rawQuery("SELECT name FROM historico",null);
            while(c.moveToNext()){
               alimentos.add(c.getString(0));
            }
            c.close();
         } catch (SQLException e) {
            e.printStackTrace();
         }
         return alimentos;
      }

      public ArrayList<String []> getActualAlimentos(){

         ArrayList<String[]> alimentos = new ArrayList<String[]>();
         SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy", Locale.getDefault());
         Date date;
         try {
            SQLiteDatabase db = getWritableDatabase();
            Cursor c =  db.rawQuery("SELECT name,date FROM actual ORDER BY date ASC",null);
            while(c.moveToNext()){
               String[] alimento = new String[2];
               Long l = Long.parseLong(c.getString(1));
               alimento[0]= c.getString(0);
               date = new Date(l);
               alimento[1] = format.format(date);
               alimentos.add(alimento);
            }
            c.close();
         } catch (SQLException e) {
            e.printStackTrace();
         }
         return alimentos;
      }
      public int deleteAlimento(String name){
         try {
            SQLiteDatabase db = getWritableDatabase();
            db.execSQL("DELETE FROM actual WHERE name ='"+name+"'");
            return 1;
         } catch (SQLException e) {

            e.printStackTrace();
            return 0;
         }
      }

      public String obtenerPorCodigo(String codigo){
         String nombre = "";
         try {
         SQLiteDatabase db = getWritableDatabase();
         Cursor c =  db.rawQuery("SELECT name FROM historico WHERE barcode = '"+codigo+"'",null);
         while(c.moveToNext()){
```

```
            nombre  = c.getString(0);

         }
         c.close();
      } catch (SQLException e) {
         e.printStackTrace();
      }
      return nombre;
      }

      public boolean obtenerCaducables(){
         try {
            SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(ctx);
            int dias = Integer.parseInt(prefs.getString("dias", "1"));
            this.maxDiff = (long) (3600*dias*24*1000);
            SQLiteDatabase db = getWritableDatabase();
            Cursor c =  db.rawQuery("SELECT date FROM actual",null);
            while(c.moveToNext()){
               Calendar cal = Calendar.getInstance();
               Long  date = Long.parseLong(c.getString(0));
               Long diff = date-cal.getTimeInMillis() ;
               if(diff<this.maxDiff)
               return true;

            }
            c.close();
         } catch (SQLException e) {
            e.printStackTrace();
         }
         return false;
      }
}
```

iOS

En iOS se ha optado por utilizar únicamente la tabla actual descrita anteriormente. Para la gestión de la base de datos se ha utilizado la librería FMDB, que permite realizaro operaciones básicas sobre bases de datos de forma sencilla. Para inegrar la librería, ya que mi proyecto está escrito en Swift y la librería está realizada con ObjectiveC , se ha implementado un Bridging Header que actúa de acoplador de la librería para que pueda ser usada con sintaxis Swift. Estos son los métodos implementados.

```
let insertSQL = "INSERT INTO ACTUAL (name, date) VALUES ('\(editName.text!)', '\(timestamp)')"

let result = controladorAlimentosDB.executeUpdate(insertSQL, withArgumentsInArray: nil)

let deleteSQL = "DELETE FROM ACTUAL WHERE name = '\(alimento)'"

let result = alimentosDB.executeUpdate(deleteSQL,withArgumentsInArray: nil)

let querySQL = "SELECT name FROM ACTUAL"

let results:FMResultSet? = controladorAlimentosDB.executeQuery(querySQL,withArgumentsInArray: nil)

while results?.next() == true {

        listaNombreAlimentos.append(results!.stringForColumn("name"))
```

```
}
```

Servicios web

Para la primera versión de la aplicación únicamente se usa el servicio que pide a Google la lista de supermercados alrededor de mi posición. La llamada es la siguiente.

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=" +  latitude + "," + longitude + "radius=" + dist + "&types=grocery_or_supermarket&sensor=true&key=APIKEY
```

Donde latitude y longitude son la latitud y longitud actuales, y dist es el radio de búsqueda.

La llamada se realiza de forma distinda, según la plataforma.

En Android se usa la librería Volley, recomendada por Google para llamadas asíncronas a servidor y fácilmente integrable en la app mediante gradle. El código usado para realizar la llamada mediante Volley es el siguiente.

```
RequestQueue queue = Volley.newRequestQueue(getActivity().getApplicationContext());
JsonObjectRequest jsObjRequest = new JsonObjectRequest
      (Request.Method.GET, query.toString(), null, new Response.Listener<JSONObject>() {

         @Override
         public void onResponse(JSONObject response) {
            JSONArray results = null;
            try {
               results = response.getJSONArray("results");

            for(int i=0;i< results.length();i++){
               JSONObject objJsonPlace = (JSONObject) results.get(i);

latitudes.add(objJsonPlace.getJSONObject("geometry").getJSONObject("location").getDouble("lat"));

longitudes.add(objJsonPlace.getJSONObject("geometry").getJSONObject("location").getDouble("lng"));
               names.add(objJsonPlace.getString("name"));
            }

               pd.dismiss();
               if(names.size()>0){
                  for(int i=0;i<names.size();i++){
                     if(mapa != null)
                     mapa.addMarker(new MarkerOptions().position(new LatLng(latitudes.get(i), longitudes.get(i))).title(names.get(i)));
                  }
               }else{
                  Toast.makeText(getActivity(), "No se han encontrado lugares cercanos", Toast.LENGTH_SHORT).show();
               }

            } catch (JSONException e) {
               e.printStackTrace();
            }
```

```
            }
        }, new Response.ErrorListener() {

            @Override
            public void onErrorResponse(VolleyError error) {


            }
        });

queue.add(jsObjRequest);
```

Podemos observar como en el onResponse(), que se ejecuta cuando la llamada ha ido bien se realiza el parseo de la respuesta obtenida y así obtenemos los datos que necesitamos para añadir los marcadores en el mapa.

En iOS la llamada y el parseo de datos se realiza tal como vimos durante el curso en el ejemplo de la aplicación de El Tiempo.

Vistas

Android

Ilustración 2 Pantalla Añadir Alimento

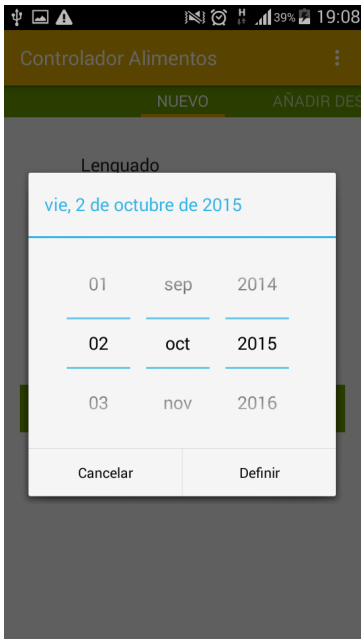


Ilustración 3 Detalle Introducción fecha

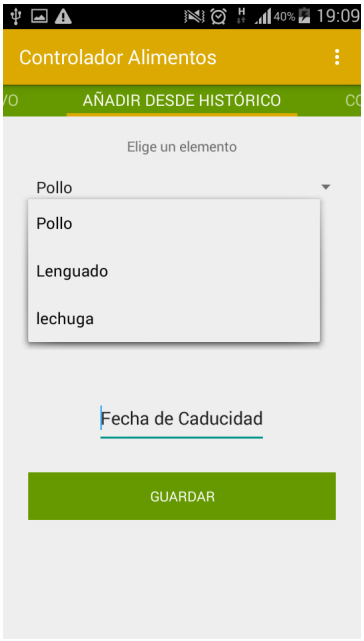


Ilustración 4 Añadir desde histórico

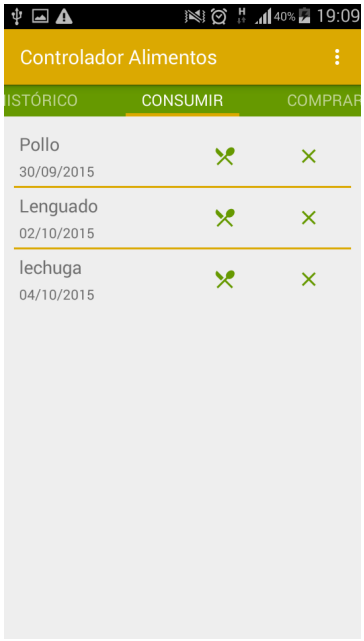


Ilustración 5 Lista de alimentos en stock

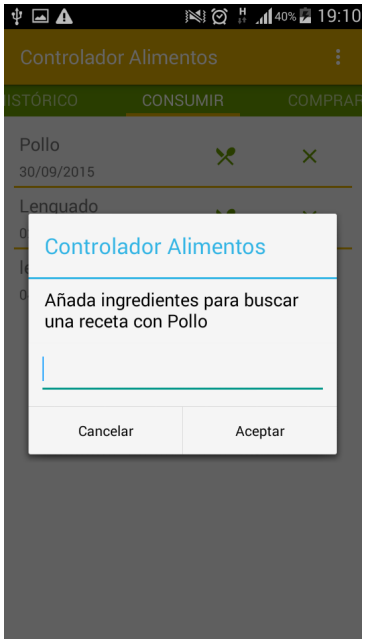


Ilustración 6 Diálogo para buscar receta

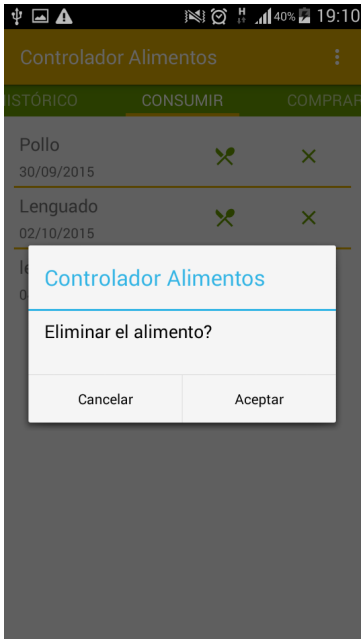


Ilustración 7 Diálogo de confirmación de borrado

Ilustración 8 Pantalla de mapa

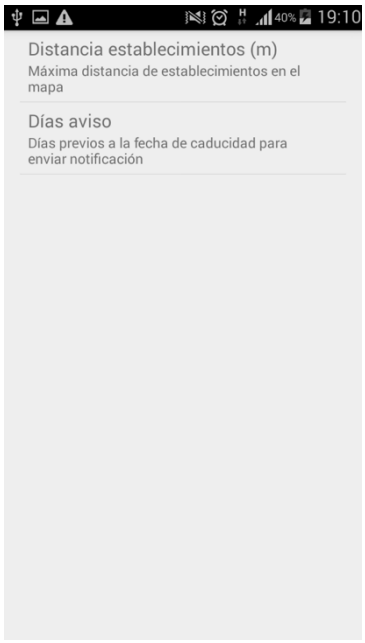


Ilustración 9 Pantalla de ajustes

iOS

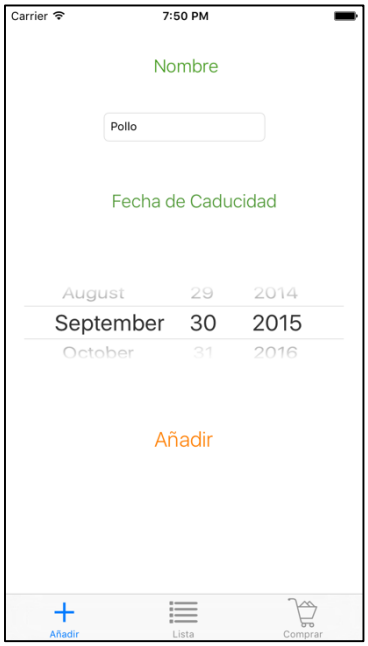


Ilustración 10 Pantalla añadir alimento

Ilustración 11 Lista de alimentos en stock

Ilustración 12 Detalle Actions de lista

Ilustración 13 AllertViewController para añadir alimentos y buscar receta

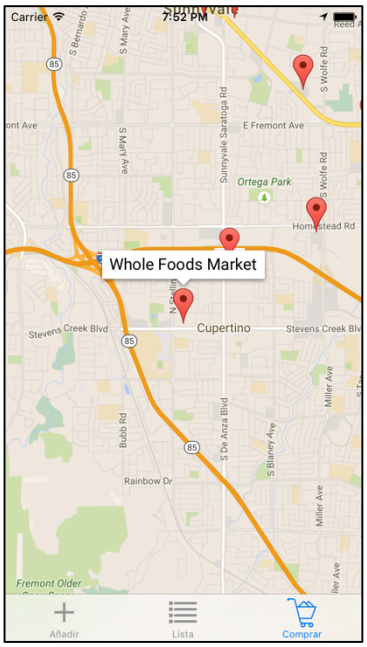


Ilustración 14 Pantalla Mapa

Capítulos adicionales

Funcionamiento de las alarmas

Android

Para que la aplicación avise al usuario de que tiene alimentos a punto de caducar se ha hecho uso de la clase AlarmManager, dicha clase se encarga de programar eventos, en nuestro caso lo que se pretende es que cada vez que se cumpla el evento de alarma se lance una notificación al usuario, para ello configuraremos el AlarmManager de la siguiente manera.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
if(!prefs.getBoolean("firstTime", false)) {
   alarmMgr = (AlarmManager)this.getSystemService(Context.ALARM_SERVICE);
   Intent intent = new Intent(this, AlarmReceiver.class);
   alarmIntent = PendingIntent.getService(this, 0, intent, 0);

alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,AlarmManager.INTERVAL_FIFTEEN
_MINUTES,AlarmManager.INTERVAL_DAY, alarmIntent);
```

```
   SharedPreferences.Editor editor = prefs.edit();
   editor.putBoolean("firstTime", true);
   editor.commit();
}
```

Este código debe ejecutarse una única vez, pues configura una alarma que una vez al día ejecutará el servicio AlarmReceiver, en dicho servicio se comprobará si hay alimentos que caducan dentro del margen que ha introducido el usuario en sus preferencias, si es así se notificará al usuario, que clickando en la notificación irá directamente a la lista de alimentos para así poder o bien buscar una receta, o bien eliminar dicho alimento. Se ha optado por el esquema que permite un menor consumo de recursos, ya que es una alarma diaria, al usar setInexactRepeating, para que así el sistema ejecute la alarma junto con otras alarmas de baja prioridad. Se ha optado por usar una alarma diaria en lugar de alarmas individuales para cada alimento porque el AlarmManager presenta un inconveniente, si el teléfono se apaga las alarmas se borran, y si programaramos alarmas para cada uno de los alimentos sería bastante complicado mantener el estado de las alarmas. Para mantener nuestra alarma diaria se ha optado por la solución clásica, añadir un BroadcastReceiver cuando el teléfono se enciende para volver a programar la alarma. El código de dicho BroadcastReceiver es el siguiente.

```
public class TurnOnReceiver extends BroadcastReceiver {
   private AlarmManager alarmMgr;
   private PendingIntent alarmIntent;
   public TurnOnReceiver() {
   }

   @Override
   public void onReceive(Context context, Intent intent) {
       if (intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
          Log.v("Controlador alimentos","Entra");
          if (alarmMgr!= null) {
                alarmMgr.cancel(alarmIntent);
          }
          alarmMgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
          Intent i = new Intent(context, AlarmReceiver.class);
          alarmIntent = PendingIntent.getService(context, 0, i, 0);

alarmMgr.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,AlarmManager.INTERVAL_FIFTEEN
_MINUTES,AlarmManager.INTERVAL_DAY, alarmIntent);
           }
   }
}
```

iOS

En iOS se ha optado por programar notificaciones un día antes de que los alimentos caduquen, ya que el sistema mantiene las notificaciones programadas y resulta más sencillo gestionarlas de esta manera. Así es como programamos una notificación al añadir un alimento.

```
let notification = UILocalNotification()

notification.alertBody = "\(editName.text!) caduca mañana" // text that will be displayed in the

notificationnotification.alertAction = "Abrir" // text that is displayed after "slide to..." on the lock screen - defaults to "slide to view"

notification.fireDate = date.date.dateByAddingTimeInterval(-60*60*24) // todo item due date (when notification will be fired)

notification.soundName = UILocalNotificationDefaultSoundName // play default sound

notification.userInfo = ["UUID": editName.text!, ] // assign a unique identifier to the notification so that we can retrieve it later

notification.applicationIconBadgeNumber++

UIApplication.sharedApplication().scheduleLocalNotification(notification)
```

Al usar este esquema, hay que tener varias consideraciones, debemos borrar la notificación si eliminamos el alimento al que hace referencia, para ello haremos uso del userInfo que hemos añadido a la notificación y que identifica cada notificación, el siguiente código se ejecutará cada vez que eliminemos un elemento de la lista de alimentos.

```
let app:UIApplication = UIApplication.sharedApplication()

        for oneEvent in app.scheduledLocalNotifications! {

            let notification = oneEvent as UILocalNotification

            let userInfoCurrent = notification.userInfo! as! [String:AnyObject]

            let uid = userInfoCurrent["UUID"]! as! String

             if uid == self.listaNombreAlimentos[indexPath.row] {

                //Cancelling local notification

                app.cancelLocalNotification(notification)

                break;

            }

        }

        self.listaNombreAlimentos.removeAtIndex(indexPath.row)

        self.listaAlimentos.deleteRowsAtIndexPaths([indexPath], withRowAnimation: UITableViewRowAnimation.Top)
```

La segunda consideración es que al ejecutar *notification.applicationIconBadgeNumber++* nos añade el típico marcador rojo en el icono de la aplicación que indica en número de notificaciones pendientes por atender, para ello añadimos la siguiente línea de código en la función applicationDidBecomeActive del fichero AppDelegate.swift.

UIApplication.sharedApplication().applicationIconBadgeNumber = 0

Y Si…?

- Los usuarios ven que la aplicación no es cómoda de usar?
- Los restaurantes no están interesados en el modelo?
- La competencia con más recursos lanza una solución más completa?
- Los anuncios y comentarios en redes sociales no van acompañados de un número elevado de descargas?
- Los usuarios no ven ninguna ventaja en la aplicación de pago frente a la gratuita pero con propaganda.
- La aplicación tiene éxito y hay que realizar avances de funcionalidad de manera rápida con los recursos actuales?
- Quieren entrar nuevos actores en el sistema, como supermercados u otro tipo de establecimiento del sector?
- Los usuarios ven la aplicación muy útil y demandan aplicaciones similares con otro tipo de escenario?

Narración del Modelo de Negocio

El usuario, con poco tiempo y que debe hacer compras semanales o con un período mayor, normalmente en grandes superficies, que premian las ofertas basadas en cantidad (por ejemplo los 3x2), debe controlar los alimentos perecederos de que dispone para así evitar comprar cantidades mayores a las que después va a poder consumir o almacenar para su conservación, y en consecuencia evitar el desperdicio de comida y dinero. Para ello tiene 2 opciones.

OPCIÓN 1

Mantener un control periódico de los alimentos que dispone para controlar el estado o la fecha de caducidad, bastante pesado, y que muchas veces se traduce en que se olvida que se dispone de ciertos productos y se echan a perder, o acaban teniendo varios alimentos que caducan pronto y que no pueden combinarse en una receta y que por falta de espacio en el congelador acaba tirando a la basura. También podemos considerar potenciales usuarios pequeños restaurantes en los cuales un buen control del stock de alimentos es muy importante para la economía del establecimiento y para mantener una calidad óptima en sus platos.

OPCIÓN 2

Usando la aplicación Controlador de alimentos, con la cual únicamente deben introducir la fecha de caducidad o día de consumo máximo de cada alimento comprado en el momento de guardarlo. Con introducirlo una vez no debe volver a comprobar si el alimento va a caducar pronto, la aplicación se encargará de recordárselo. En el caso de un pequeño restaurante se podría llevar el control de stock de alimentos de una forma automática y sencilla.

Como podemos observar, los usuarios se verían beneficiados, tanto en facilidad de control de los alimentos de que disponen como en su economía, ya que no desperdiciarían tanta comida, pues como hemos visto en la introducción la cantidad de comida desperdiciada, precisamente en los países con mayor capacidad tecnológica a nivel personal, es enorme. Otro de los beneficios para el usuario es el conocimiento de que se está actuando de forma responsable con el resto de la sociedad y con el medio ambiente al no desperdiciar alimentos. Esto mismo es aplicable a los restaurantes.

Desde el punto de vista del desarrollador, hemos ganado un clientes que al descargar la aplicación nos producen un beneficio económico, el objetivo principal es mantener al usuario activo, ya que así se mantendrán los ingresos por publicidad. Para ello una de las estrategias a seguir es conseguir que se establezca una relación entre los 2 segmentos principales de clientes, los usuarios y los pequeños restaurantes. Si se desde los restaurantes que usan la aplicación se establece algún tipo de beneficio para los usuarios por consumir en sus locales, los restaurantes aumentarán su clientela potencial, los usuarios se beneficiaran de las ofertas, y nosotros mantendremos a los usuarios activos y promocionando la aplicación.

Conclusiones

Cómo conclusión podemos indicar que se ha cumplido con el objetivo marcado para el proyecto del Máster. Se ha completado una primera versión de una aplicación funcional, y que cubre la mayoría de aspectos vistos durante el curso, tanto de Android como de iOS. Considero que se debería pulir el concepto y ampliar alguna funcionalidad para la publicación de la aplicación con fines comerciales.

Las líneas futuras del proyecto son muchas. Desde el punto de vista técnico, se deberían incluir las mismas funcionalidades en todas las plataformas, además de intentar hacer una versión de Windows Phone para cubrir todas las plataformas mayoritarias. En la versión de Android de debe investigar el uso de la nueva API de lectura de códigos de barra incluida en la versión 7.8 de Google Play Services y que evitaría la dependencia de aplicaciones de terceros.

Para la mejora del producto se debe pensar en nuevas funcionalidades que hagan más atractiva la aplicación y que permitan crear una versión de pago apetecible. Estas funcionalidades podrían incluir gestión de listas de compras, una base de datos remota con códigos de barras y alimentos para facilitar al usuario la introducción de nuevos alimentos, inclusión de información nutricional o recomendación de dietas etc… Pensando en un futuro a largo plazo se debería estar atento a los avances en electrodomésticos inteligentes, como neveras con conectividad que controlan mediante NFC los alimentos que introducimos o sacamos de la misma. Desde el punto de vista comercial se deben buscar nuevos elementos que den valor a la aplicación, como la integración con supermercados o tiendas para poder realizar compras desde la aplicación, o la promoción desde plataformas y asociaciones de concienciación con el consumo responsable.

En el aspecto personal me ha gustado mucho poder plasmar en esta aplicación lo aprendido durante el Máster y poder realizar la aplicación en 2 plataformas distintas para así poder observar las ventajas e inconvenientes que ofrecen cada una de ellas desde el punto de vista del desarrollador. Por otro lado ha sido muy gratificante poder trabajar en una aplicación fuera del trabajo, con libertad de creación y de una forma más relajada, pudiendo buscar e investigar diferentes formas de plantear soluciones y afrontar diferentes situaciones que las que suelo encontrarme día a día.

Sin más, me gustaría agradecer a todos los profesores de máster por los conocimientos adquiridos y por elaborar un programa de asignaturas con un contenido amplio, interesante y actual.