DESARROLLO DE APLICACIÓN MULTIPLATAFORMA DE BÚSQUEDA Y GESTIÓN DE RECETAS

Carlos Martínez Gutiérrez

(Grado en Ingeniería del Software)

Alejandro Montero Roldán

(Grado en Ingeniería Informática)

Javier Vicente Cano

(Grado en Ingeniería del Software)

UNIVERSIDAD COMPLUTENSE MADRID

Trabajo de Fin de Grado

Director: Manuel Montenegro Montes

Agradecimientos:

Le damos las gracias a Manuel Montenegro, nuestro director de proyecto, por su entera disposición a ayudarnos siempre ante cualquier necesidad, así como su flexibilidad durante todo el desarrollo.

Índice

Resumen

El objetivo del presente trabajo de fin de grado es el desarrollo de una aplicación multiplataforma que recomiende recetas de cocina en función de los ingredientes que el usuario disponga en ese momento. Esta aplicación permitirá al usuario conocer recetas nuevas e interesantes pero siempre teniendo en cuenta el hecho de que se puedan realizar en ese momento, sin necesidad de comprar ingredientes adicionales.

Para usar esta aplicación, el usuario debe, en primer lugar, seleccionar un ingrediente principal para su receta, tras lo cual la aplicación mostrará al usuario una serie de ingredientes adicionales. Para cada uno de estos últimos, el usuario deberá indicar si dispone o no de dicho ingrediente. Mientras realiza esto, la aplicación mostrará aquellas recetas que sean realizables en función de la disponibilidad de ingredientes, y descartará las no realizables. Mientras tanto, el usuario podrá consultar la receta deseada y obtener toda la información necesaria para su preparación.

Con este proyecto hemos obtenido una aplicación funcional que ayuda en una tarea sencilla como elegir la comida del día ofreciendo al usuario una amplia variedad de posibilidades viables con los ingredientes disponibles. Además, es una buena base para la incorporación de nuevas funcionalidades y en un futuro cercano poder competir en el mercado.

Palabras clave

Receta, aplicación multiplataforma, aplicación móvil, Android, iOS, ingredientes, alimentación, cocina, API, Cordova.

Abstract

The purpose of this Final Degree Project is to develop a multi-platform application that recommends cooking recipes, based on the ingredients that the user accepts or discards depending on whether they are available or not. This application will allow the user to discover new and interesting recipes, but always taking into account the fact that they can be made at that time, without the need to buy additional ingredients.

To use this app, the user must first select a main ingredient for his recipe, after which the app will show several additional ingredients to the user. For each of these ingredients, the user must indicate whether the user has this ingredient or not. While doing this, the app will select those recipes that are feasible depending on the availability of ingredients, and discard those that are not. As soon as all the ingredients necessary for the preparation of a recipe are available, the user can consult the desired recipe and obtain all the necessary information for its preparation.

With this project we have obtained a functional app that helps in a simple task like choosing the meal of the day offering the user a wide variety of viable possibilities with the available ingredients. In addition, is a good base for the inclusion of new features and in the near future to be able to participate in the market.

Keywords

Recipe, multi-platform app, mobile app, Android, iOS, ingredients, food, cooking, API, Cordova.

1. Introducción

En este primer capítulo se explicará la motivación que nos ha llevado a desarrollar este proyecto, así como todos los antecedentes de otras aplicaciones similares. También se describen los objetivos y el plan de trabajo seguido.

1.1 Antecedentes y motivación

Cada vez existen más aplicaciones de recetas que podemos instalar en nuestro teléfono o *tablet* y, aunque muchas de ellas ofrecen características y funcionalidades interesantes, ninguna aborda el problema del usuario que cuenta con pocos ingredientes; a pesar de acceder a muchas recetas, la limitación de ingredientes le impide encontrar la adecuada.

A veces, el usuario está condicionado por los ingredientes que tiene disponibles. Otras veces, está limitado por el ingrediente principal y tiene a su disposición otros muchos ingredientes secundarios. En ambos casos, las aplicaciones existentes no ofrecen un método sencillo de búsqueda que facilite la selección de la receta más conveniente.

Algunas aplicaciones en las que hemos encontrado estas limitaciones son: Recetix[1], Recetas de cocina gratis[2], Cookpad Recetas[3], Cocina Tradicional (Recetas)[4], Recetas de cocina casera gratis[5] y Nestlé Cocina[6]. Estas aplicaciones disponen de un amplio catálogo de recetas, pero sus funciones de búsqueda son bastante restrictivas, permitiendo simplemente buscar una receta por nombre, o por categoría.

En este proyecto se pretende desarrollar una aplicación multiplataforma que resuelva todos estos problemas ofreciendo la mejor experiencia para aquel cocinero que busca elaborar la mejor receta posible sin necesidad de

[1] http://www.recetix.com/
[2] https://play.google.com/store/apps/details?id=com.recetasgratisnet.recetasdecocina&hl=es
[3] https://cookpad.com/es
[4] https://cocina-tradicional.es/
[5] https://craftlog.com/es/cocina/
[6] https://www.nestlecocina.es/app-nestle-cocina

perder la oportunidad por falta de ingredientes, proponiendo alternativas con algún ingrediente adicional.

1.2 Objetivos

Debido a todo lo anteriormente explicado, el objetivo de este proyecto es realizar una aplicación móvil multiplataforma que, de una manera rápida, intuitiva y amigable, proporcione recetas basándose en ciertos ingredientes seleccionados y descartados.

Para definir el alcance del proyecto hemos identificado ciertos objetivos clave que debemos alcanzar:

- **Desarrollar un algoritmo de búsqueda dinámica de recetas**: La base de la aplicación es la obtención de recetas a partir de ciertos ingredientes. Para ello hay que desarrollar un algoritmo que, teniendo en cuenta los ingredientes seleccionados y descartados, la limitación de consultas al catálogo de recetas y los tiempos de espera, obtenga una lista de posibles recetas para el usuario. Decimos que la búsqueda es dinámica en tanto que los criterios de búsqueda de recetas se van refinando a medida que el usuario interactúa con la aplicación.
- **Seleccionar una base de datos de recetas:** Para acelerar el desarrollo de la aplicación vamos a usar una base de datos externa de recetas de cocina, accesible vía API a la que consultaremos las posibles recetas con los ingredientes seleccionados.
- **Diseñar la arquitectura de la aplicación de búsqueda de recetas:** Es necesario definir los módulos de la aplicación y sus relaciones que posibiliten ofrecer una interfaz de usuario multiplataforma (Android, iOS, web...), recoger los ingredientes seleccionados o descartados, consultar a la base de datos de recetas, aplicar el algoritmo de selección y mostrar los resultados al usuario.
- **Diseñar la interfaz de usuario:** Queremos crear una interfaz interactiva en la que el usuario elija un ingrediente principal y la aplicación vaya proponiendo ingredientes adicionales en un proceso iterativo.

1.3 Plan de trabajo

Una vez definidos los objetivos del proyecto planteamos un plan de trabajo en cinco grandes bloques:

[T1] Definición de una metodología de desarrollo y trabajo.

Dado que somos tres alumnos, es necesario definir el rol que desempeñamos dentro del equipo de trabajo, la responsabilidad de cada uno y ciertos objetivos dependiendo de nuestra disponibilidad.

Como algunos empezamos las prácticas en empresa y otros tenemos durante el primer cuatrimestre alguna asignatura, estimamos que esta tarea se realizará en un máximo de dos semanas.

[T2] Estudio y familiarización con las tecnologías a utilizar: APIs, frameworks, etc.

Para alcanzar nuestros objetivos debemos investigar diversas APIs externas para la búsqueda de recetas y frameworks para el desarrollo de una app multiplataforma y para la creación de vistas. Dado que durante la carrera hemos utilizado diferentes herramientas y lenguajes, suponemos que esta tarea no nos llevará más de un mes. Este punto está desarrollado en el capítulo 3.

[T3] Diseño del algoritmo, de la lógica de la aplicación y de las vistas.

Antes de empezar a desarrollar, debemos especificar el funcionamiento del algoritmo y calcular su posible coste sin entrar en su implementación (capítulo 5). Una vez claro su funcionamiento, empezaremos a diseñar los módulos en los que se dividirá la aplicación (capítulo 6) y diseñaremos bocetos de la interfaz de usuario (capítulo 6.3).

Dado que coincidiremos con las vacaciones de Navidad y con exámenes por parte de algunos integrantes, estimamos dos meses para la realización de esta fase.

[T4] Implementación del algoritmo y de la aplicación, e integración de las dos partes.

La fase de desarrollo es la más extensa de las cinco y la que más esfuerzo nos llevará. Se tratará de crear el algoritmo en un entorno de

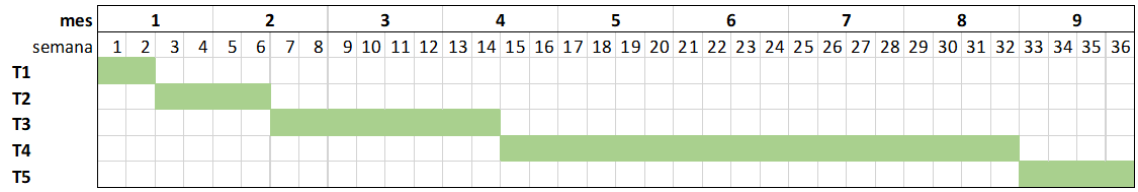
pruebas y, una vez desarrollado, crearemos la aplicación (lógica y vistas) y se integrará con el algoritmo. (capítulo 6)

Al estar libres de exámenes y asignaturas, esta fase se podrá terminar antes de junio.

[T5] Plan de pruebas y corrección de errores.

Los errores son comunes en todo proyecto software por lo que, después del desarrollo de la aplicación, vemos oportuno reservar unas semanas para comprobar que todo funciona correctamente antes de su entrega.

A continuación, se muestra un cronograma indicando de forma visual la duración de los cinco bloques:



2. Introduction

In this chapter we will explain the motivation that led us to develop this project as well as all the background of other similar apps. The objectives and the work plan followed are also described.

2.1 Motivation

There are more and more recipe apps that we can install on our phone or tablet and although many of them offer interesting features and functionality, none of them address the problem of the user who has few ingredients; despite having access to many recipes, the limitation of ingredients prevents him from finding the right one.

Sometimes, the user is limited by the ingredients available. Other times, the user is limited by the main ingredient and many other secondary ingredients are available. In both cases, existing apps do not offer a simple method of searching to facilitate the selection of the most convenient recipe.

Some apps where we have found these limitations are: Recetix[7], Recetas de cocina gratis[8], Cookpad Recetas[9], Cocina Tradicional (Recetas)[10], Recetas de cocina casera gratis[11] and Nestlé Cocina[12]. These apps have an extensive catalogue of recipes, but their search functions are quite restrictive, allowing you to simply search for a recipe by name, or by category.

This project pretends to develop a multi-platform application that solves all these problems by offering the best experience for the chef who seeks to prepare the best possible recipe without having to miss the opportunity due to lack of ingredients, proposing alternatives with some additional ingredient.

[7] http://www.recetix.com/
[8] https://play.google.com/store/apps/details?id=com.recetasgratisnet.recetasdecocina&hl=es
[9] https://cookpad.com/es
[10] https://cocina-tradicional.es/
[11] https://craftlog.com/es/cocina/
[12] https://www.nestlecocina.es/app-nestle-cocina

2.2 Objectives

The purpose of this project is to create a multi-platform app that provides recipes based on certain ingredients selected and discarded in a fast, intuitive and user-friendly way.

To define the scope of the project we have identified certain key objectives that we must achieve:

- Develop a dynamic recipe search algorithm: The basis of the app is obtaining of recipes from certain ingredients. For this purpose, an algorithm must be developed to obtain a list of possible recipes, taking into account the selected and discarded ingredients, the limitation of queries to the recipe catalogue and the waiting times for the user. We say that the search is dynamic because the search criteria for recipes are refined as the user interacts with the application.
- Select a recipe database: To speed up the development of the application we will use an external database of recipes, accessible via API, where we will consult the possible recipes with the selected ingredients.
- Design the architecture of the recipe search application: It is necessary to define the modules of the application and their relationships that make it possible to offer a multiplatform user interface (Android, iOS, web...), collect the selected or discarded ingredients, consult the recipe database, apply the selection algorithm and show the results to the user.
- Designing the user interface: We want to create an interactive interface in which the user chooses a main ingredient and the application proposes additional ingredients in an iterative process.

2.3 Work plan

Once the objectives of the project had been defined, we proposed a work plan in five large blocks:

[T1] Definition of a development and working methodology:

Since we are three students it is necessary to define the role we play within the work team, the responsibility of each one and certain objectives depending on our availability.

As some of us start our internships in a company and others have some subjects during the first four-month period, we estimate that this task will be completed in a maximum of two weeks.

[T2] Study and familiarization with the technologies to be used: APIs, frameworks, etc.

To achieve our objectives we must investigate various external APIs for the search of recipes and frameworks for the development of a multi-platform app and for the creations of views. Since we have used different tools and languages during the course, we assume that this task will take no more than a month. This point is explained in chapter 3.

[T3] Design of the algorithm, app logic and views.

Before we start to develop, we must specify how the algorithm works and calculate its possible cost without entering into its implementation (chapter 5). Once its operation is clear, we will begin to design the modules in which the app will be divided (chapter 6) and we will design sketches of the user interface (chapter 6.3).

Since we will coincide with the Christmas holidays and with exams by some members, we estimate two months for the completion of this phase.

[T4] Implementation of the algorithm and the app, and integration of both parts.

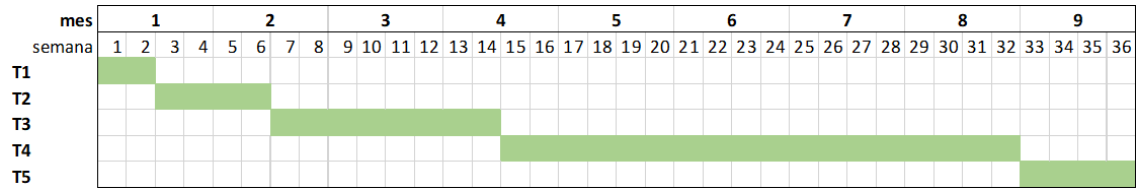
The development phase is the longest of the five and the one requiring the most effort. We will try to create the algorithm in a test environment and, once developed, we will create the application (logic and views) and integrate it with the algorithm (chapter 6).

Being free of exams and lessons, this phase can be completed before June.

[T5] Test plan and error correction

Errors are common in all software projects, so after the development of the app, it is advisable to reserve a few weeks to check that everything is working correctly before delivery.

Below there is a timeline showing visually the duration of the five blocks:



3. Selección de herramientas y tecnologías

Antes de comenzar con el desarrollo de la aplicación, se realizó una evaluación de la tecnología y de las herramientas más apropiadas para su desarrollo e implementación. Para realizar una selección fueron utilizados criterios como la madurez de las herramientas, la existencia de experiencia previa en su manejo, el *feedback* de compañeros y profesores y la existencia de manuales de consulta.

A continuación, se mostrarán los servicios externos contemplados, continuando con las tecnologías, lenguajes y *frameworks* a destacar, siguiendo con los gestores de bases de datos y terminando con las herramientas adicionales.

3.1. Servicios externos: APIs

Uno de los objetivos más importantes del proyecto fue el uso de una API externa que proporcione el mayor número de recetas posible. Se realizó un trabajo extenso en la búsqueda de APIs, intentando conseguir aquella que se adaptase mejor a nuestros requisitos. Se exploraron las siguientes:

3.1.1. Edamam [1]

Edamam es una *startup* neoyorquina fundada en 2011 con financiación privada que posee no más de 50 empleados. Su principal servicio consiste en ofrecer datos sobre recetas y nutrición en tiempo real para ahorrar tiempo y dinero a sus clientes, entre los que se encuentran Nestlé, Samsung y The New York Times, entre otros.

Los servicios aportados por esta API beneficiosos para el proyecto eran los siguientes:

- Incorpora recetas en español.
- Posee una documentación extensa y bien especificada.
- Contiene un amplio catálogo, con 1,5 millones de recetas.

- Mediante una cuenta gratuita se pueden realizar hasta 5000 peticiones al mes. No obstante, contactamos con la empresa, con el fin de informarles sobre nuestro proyecto, y nos concedieron un mayor número de conexiones mensuales sin coste extra.

El principal inconveniente encontrado fue:

- La lista de ingredientes de una receta no está normalizada. En algunos casos se mezclan, dentro de la misma cadena de texto, el ingrediente y la cantidad a utilizar en la receta. Esto dificulta claramente nuestro objetivo de realizar búsquedas basadas en ingredientes, y por ese motivo fue descartada.

3.1.2. Yummly [2]

Yummly se fundó en el 2009 como una aplicación móvil y sitio web que proporcionaba recetas de cocina. En 2013 hicieron pública su API para que los desarrolladores puedan usar sus recetas en sus servicios.

Los pros contemplados fueron:

- Contiene más de 1 millón de recetas
- Usada por grandes entidades como Google, Unilever, la universidad de Stanford y United HealthCare.
- Documentación extensa y bien especificada.
- Permite la exclusión de ingredientes en la búsqueda de recetas.

Los inconvenientes contemplados fueron:

- Solo disponible en inglés.
- El plan estándar cuesta 500$/mes (permite 250 mil peticiones al mes más 0’002$ extra por cada petición adicional). Hay posibilidad de un plan para estudiantes (gratuito, 30 mil llamadas a la API dándoles atribución).

La API de Yummly, debido a todas las ventajas que aportaba al proyecto, su plan académico y su extensa popularidad fue la API seleccionada para la aplicación del proyecto.

3.1.3. Food2Fork [3]

Food2Fork es un sitio *web* de recetas que combina los ingredientes y la búsqueda por título. Las recetas de Food2Fork son de cocineros y *chefs* con amplia experiencia en la cocina.

Los pros contemplados fueron:

- La versión gratuita permite 500 peticiones al día.

Los inconvenientes contemplados fueron:

- Solo devuelve un máximo de 30 recetas por llamada.
- Documentación muy pobre y poco extensa.
- Solo disponible en inglés.

3.1.4. BigOven [4]

BigOven fue iniciado por un veterano del software al que le encanta cocinar y deseaba poder buscar las recetas de su familia, así como disponer de un amplio libro de recetas para aprovechar al máximo lo que era más fresco ese día.

Los pros contemplados fueron:

- Ofrece 350.000 recetas.
- Muy popular y con una alta calidad en sus datos

Los contras contemplados fueron:

- Solo disponible en inglés.
- Documentación dispersa y pobremente explicada.
- No dispone de planes gratuitos.

3.2. Tecnologías, lenguajes y frameworks

A continuación, se muestran todas las tecnologías, lenguajes y *frameworks* necesarios para el desarrollo del proyecto.

3.2.1. Apache Cordova [5]

Con la necesidad de encontrar un *framework* que permitiese crear una aplicación móvil, Apache Cordova ofreció la posibilidad de realizar la aplicación evitando los lenguajes y herramientas nativos de cada plataforma móvil.

Apache Cordova permite, mediante el uso de tecnologías *web* estándar como HTML5, CSS3 y JavaScript, desarrollar una aplicación móvil multiplataforma. De esta manera se realizó la aplicación en un entorno web, en el cual los integrantes del proyecto estamos más familiarizados. Además, la aplicación se diseñó según las pautas de diseño *web* adaptable, de modo que puede ser ejecutada tanto en teléfonos móviles como en *tablets*.

Hay varios componentes en una aplicación de Cordova. A continuación, se muestra una descripción de los principales, basada en la documentación oficial de Cordova [6]. El siguiente diagrama muestra una vista de alto nivel de la arquitectura de una aplicación Cordova.

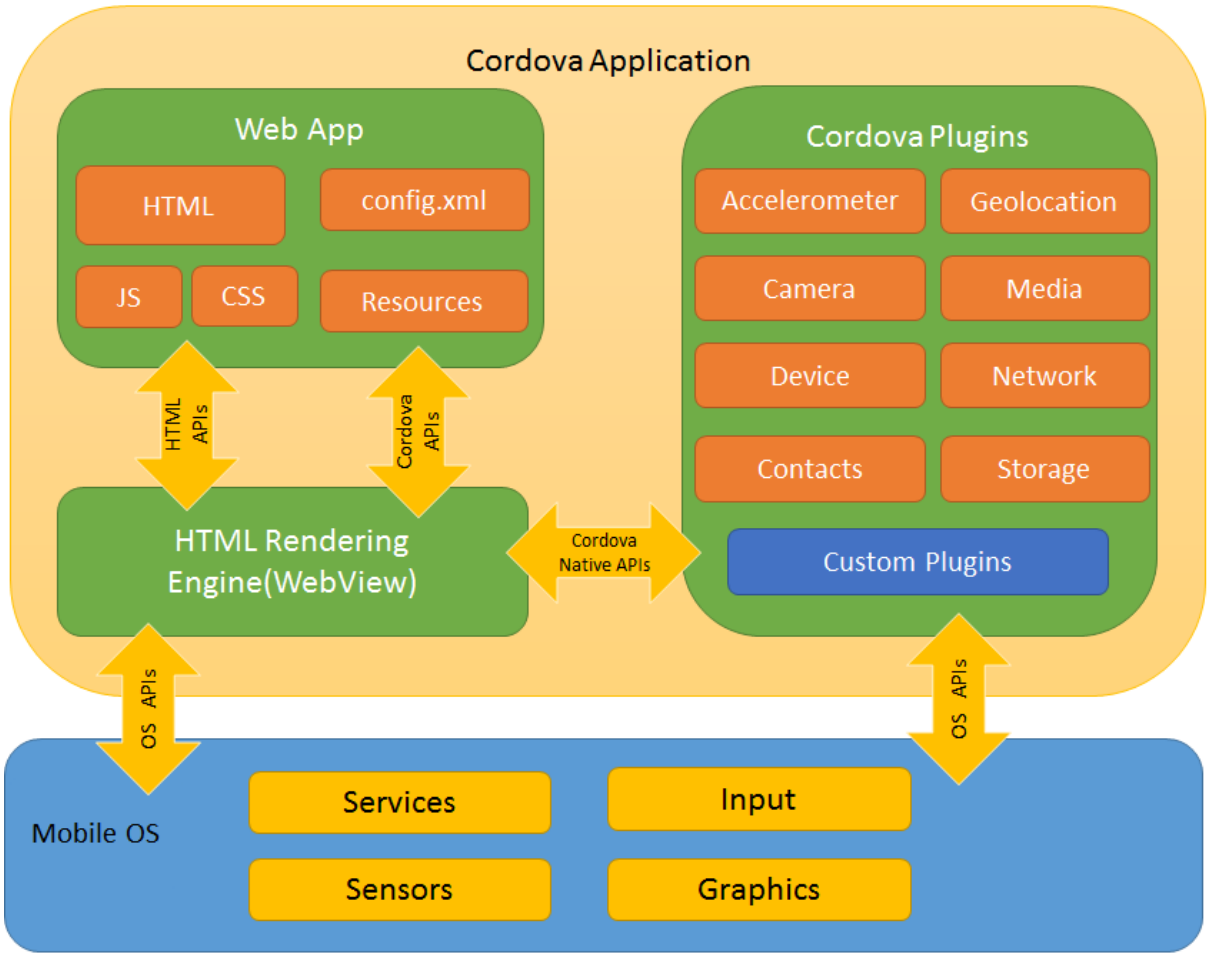


Imagen obtenida de la página oficial de Apache Cordova [6]

El módulo *Web App* contiene el código de la aplicación. La aplicación se implementa en un archivo local llamado *index.html*, el cual hace referencia a hojas de estilo CSS, código JavaScript, imágenes, archivos multimedia y otros recursos necesarios.

La aplicación se ejecuta en un componente *WebView*, que no es más que un navegador ejecutándose a pantalla completa dentro del dispositivo móvil.

Este contenedor contiene un archivo muy crucial, el archivo *config.xml* que proporciona información sobre la aplicación y especifica los parámetros que afectan su funcionamiento, como, por ejemplo, si esta responde a los cambios de orientación.

Los añadidos (*plugins*) proporcionan una interfaz para poder comunicarse con los componentes nativos, como batería, cámara, contactos, etc. También se pueden desarrollar nuevos *plugins*, aunque no hemos hecho uso de esta característica.

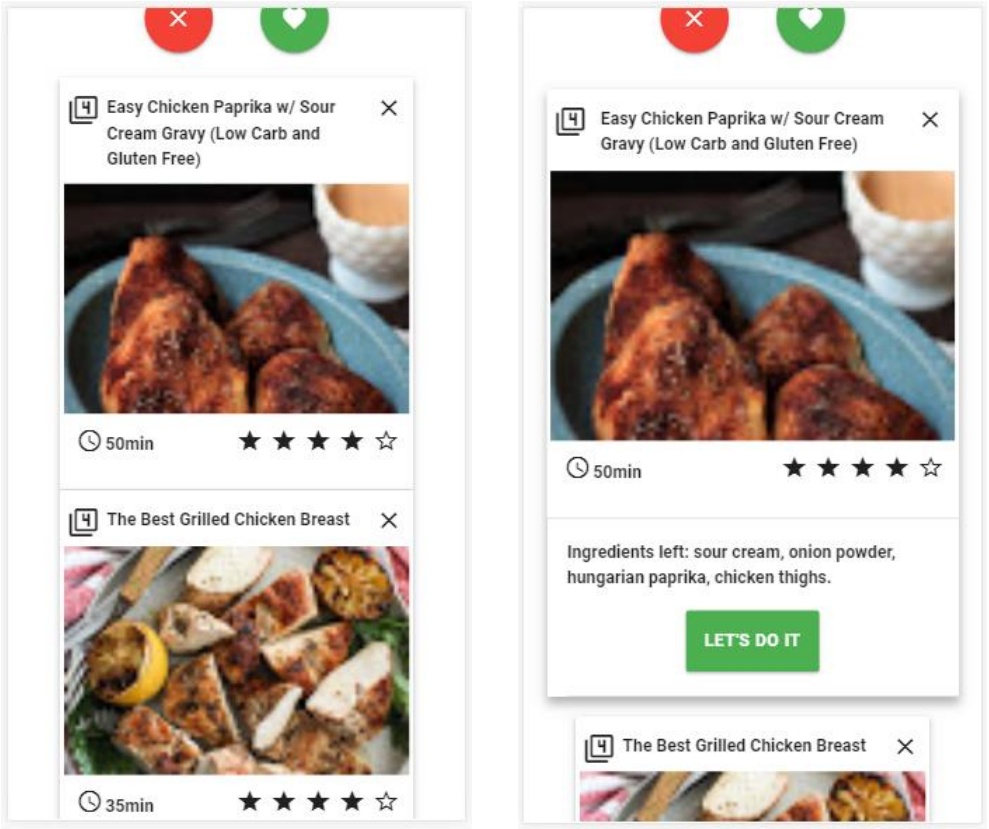
Se generaron una serie de archivos siguiendo la documentación de Cordova para construir un proyecto. Además de crear el proyecto multiplataforma, Cordova permite ejecutar dicho proyecto dentro de un emulador.

3.2.2. HTML 5 [7]

HTML (*HyperText Markup Language*) es el lenguaje de marcado utilizado para la elaboración y estructuración de la página web utilizada por Cordova. Fue utilizado para la construcción de los componentes estáticos de la interfaz de la aplicación, así como su uso en conjunto con JavaScript para la creación de los elementos dinámicos.

3.2.3. CSS 3 [8]

CSS (Cascading Style Sheets) es el lenguaje encargado de dar estilo a la página web, en este caso al documento HTML. Trabajando junto con Materialize se han conseguido elementos como, por ejemplo; mostrar una lista de recetas posibles con la información limitada (título, foto, número de ingredientes restantes, tiempo y puntuación) y al pulsar sobre un elemento de esta lista obtener el nombre de los ingredientes restantes y mostrar un botón para acceder a la página donde se muestra toda la información de la receta.



3.2.4. Materialize [9]

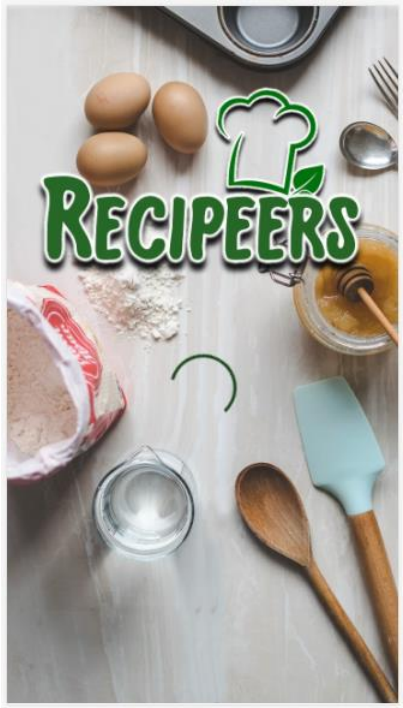
Buscando cómo crear elementos, encontramos Materialize, que nos ha proporcionado el código necesario para crear los elementos e interacciones mencionadas anteriormente.

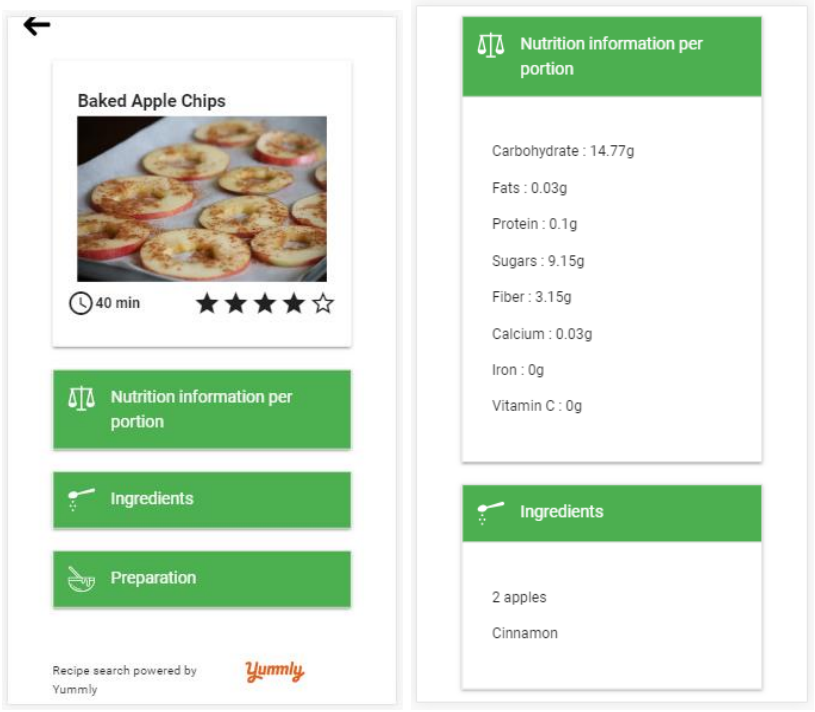
Creado y diseñado por Google, Material Design es un lenguaje de diseño que combina los principios clásicos del diseño con la innovación y la tecnología. El objetivo de Material Design es desarrollar un sistema de diseño que permita una experiencia de usuario unificada a través de todos sus productos en cualquier plataforma, en nuestro caso para móvil y *tablet*.

Adaptándose correctamente a la página *web*, Materialize permite asignar estilos y comportamientos a algunos componentes de la aplicación, que de otro modo hubiesen sido demasiado complejos de implementar para los miembros del proyecto. Además, proporciona varios componentes gráficos que facilitan la concepción y el diseño de la interfaz gráfica.

Ha sido un éxito trabajar con Materialize porque contiene una lista muy extensa de elementos que pueden ser incorporados en la interfaz

gráfica de una aplicación, obteniendo ideas tan buenas como el *loader* circular o los *collapsibles*.





3.2.5. JavaScript [10]

Lenguaje de programación interpretado, usado principalmente en navegadores web. Ha sido esencial para crear la lógica de la aplicación.

3.2.6. Node.js [11]

Es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano (pues no requiere programación multihebra) y eficiente. Posee un registro de librerías (*NPM Registry*), que puede ser accedido desde la línea de comandos mediante el gestor de paquetes de Node: npm. El ecosistema de paquetes de Node.js es el ecosistema más grande de librerías de código abierto en el mundo.

Para instalar Cordova Apache fue necesario Node.js porque la línea de comandos de Cordova se ejecuta en bajo este entorno.

3.2.7. RequireJS [12]

RequireJS es un cargador de archivos y módulos javascript. Está optimizado para el uso en el navegador, pero se puede usar en otros entornos de JavaScript, como Rhino y Node.js. El uso de un gestor de *scripts* modular como RequireJS mejora la velocidad y la calidad del código, ocupándose de las dependencias de los módulos.

Desde el principio nos encontramos con el problema de dividir el código en módulos ya que Javascript, hasta sus últimas versiones, no disponía de un sistema que lo permitiese. Con RequireJS conseguimos dividirlo y crear una arquitectura similar al patrón Modelo Vista Controlador, de forma que sea más fácil actualizar, leer y mantener el código en el futuro.

3.2.8. Spring boot [13]

Durante el desarrollo de Recipeers tuvimos la necesidad de crear un servicio conectado a la base de datos que nos proporcionase imágenes de ingredientes para mejorar la experiencia de usuario y a la vez aligerar el

peso de la aplicación al no tener que guardar todas las imágenes localmente. Para ello decidimos crearlo con Spring Boot.

Spring Boot presenta características como crear aplicaciones sencillas con un servidor *Tomcat* integrado, configuración automática siempre que sea posible y adaptable a varios entornos y externalizada. Todo ello evita generar más código del necesario y tener que crear una configuración XML.

3.3. Gestores de bases de datos

3.3.1. MySQL [14]

Durante el desarrollo de la aplicación fue necesaria la administración de una base de datos. Nuestra base de datos es una simple lista de ingredientes con su categoría y foto asociada.

MySQL es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation y está considerada la como la base datos de código abierto más popular del mundo, sobre todo para entornos de desarrollo web.

MySQL es usado en sitios como Wikipedia, Google (aunque no para búsquedas), Facebook, Twitter, Flickr, y YouTube.

Proporciona una API para poder conectarse con distintos lenguajes, está disponible en distintas plataformas y sistemas, permite seleccionar el mecanismo de almacenamiento, utiliza una conexión segura, utiliza replicación y realiza una búsqueda e indexación de campos de texto.

MySQL fue seleccionada por su fácil uso, además de por el hecho de que todos los miembros del proyecto lo han aprendido a utilizar durante la carrera. Se quería una base de datos simple y rápida de crear y MySQL cumplía todos esos requisitos.

3.4. Herramientas adicionales

3.4.1. Github [15]

Al empezar un proyecto a realizar entre tres personas, se necesitó una herramienta capaz de proporcionar un servicio para que todos los miembros del proyecto trabajasen sobre el mismo código, ayudando a administrar las distintas versiones del proyecto.

El servicio utilizado se basa en el sistema de control de versiones Git. Trata de un sistema distribuido, en el que cada usuario tiene una copia del repositorio entero, incluyendo todas las versiones. Cada versión se crea cuando un integrante realiza cambios en su código local, ya sea generando, modificando o eliminando, y lo sincroniza con el repositorio que comparte el equipo realizando un *COMMIT* y *PUSH*. Un *commit* consolida dentro del repositorio local los cambios realizados, obteniendo como resultado una nueva versión. Un *push* envía las versiones al repositorio "maestro" (compartido entre todos los usuarios). Con ello el resto de integrantes puede realizar un PULL sincronizando su código local.

Se utilizó esta herramienta durante todo el proceso de desarrollo de la aplicación y no se tuvieron problemas, pues todos los miembros del equipo ya contaban con conocimientos sobre Git.

El portal que almacenaba nuestro repositorio fue GitHub. Elegimos esta web porque ya la habíamos usado antes y nos permitía crear un repositorio privado, de forma que protegería nuestro código de terceros.

3.4.2. Google Drive [16]

Frente a la necesidad de compartir archivos entre los miembros del equipo y también con el tutor, se utilizó la herramienta Google Drive

Google Drive proporciona un servicio de almacenamiento de archivos en la nube, permitiendo a los usuarios acceso en cualquier momento y en cualquier sitio. Los usuarios de una carpeta compartida pueden acceder y modificar los archivos ubicados en ella.

Esta tecnología nos ayudó al principio con las actas de las reuniones, los bocetos y los distintos avances en cuanto a la redacción de la

funcionalidad, del algoritmo, etc. permitiendo el acceso a todo el equipo. En el tramo final fue clave a la hora de redactar la memoria. En particular, la funcionalidad de poder modificar el mismo documento por varias personas al mismo tiempo permitió hacer más sencilla esta tarea.

En definitiva, Google Drive fue la tecnología que usamos para compartir todos los archivos del proyecto excluyendo los relacionados con código fuente de la aplicación.

3.4.3. Trello [17]

Con el propósito de gestionar el proyecto, marcar objetivos, organizar el trabajo de cada semana y planificarse adecuadamente se utilizó la herramienta de administración de proyectos Trello.

Esta herramienta proporciona un tablero que contiene una serie de listas, cada una de ellas compuesta por varias tarjetas. El tablero puede ser compartido por un equipo, o puede ser utilizado por una única persona. Cada lista fue utilizada para gestionar las acciones del equipo, describiendo en cada tarjeta las características de una acción y su estado: pendiente, en proceso o completada. De este modo, añadiendo *checklists*, etiquetas, fechas de vencimiento y otros elementos a las tarjetas se conseguía que cada miembro identificase claramente sus tareas a desarrollar.

Trello es una herramienta de la que hemos hecho uso desde el principio del proyecto. Cada miembro ha participado en el tablón planificando su trabajo y el del resto si era posible. Cualquier acción a realizar solicitada por el tutor era clasificada y adjudicada a la persona adecuada. Este gestor de proyectos ha resultado ser de gran ayuda a la hora de valorar y revisar el trabajo de cada miembro.

3.4.4. Editores de texto: Sublime Text [18] y Atom [19]

Con la necesidad de editar código para la aplicación, las herramientas utilizadas por el equipo fueron Sublime Text y Atom.

Sublime Text fue más utilizada que Atom por tener más familiaridad con ella. Todos los integrantes han trabajado con ella durante la carrera. Aunque algunos miembros del equipo utilizaron Atom debido a ventajas

como: compartir el espacio de trabajo y editar el mismo código en tiempo real con el resto de miembros, mejoras en la integración del lenguaje y trabajar con Github creando nuevas ramas, *commits*, realizar *push* y *pull*, resolver conflictos de combinación, ver solicitudes de extracción y más.

3.4.5. PuTTY [20]

Para gestionar los ingredientes de la aplicación era necesario un cliente SSH para conectarse a la base de datos y para ello se utilizó PuTTy.

PuTTy es un cliente SSH que funciona por consola y que nos permitió gestionar, de manera remota, la base de datos almacenada en un servidor. En particular, utilizamos esta herramienta para añadir manualmente entradas a la tabla que contiene las imágenes de los ingredientes manejados por la aplicación.

3.4.6. Sketch [21]

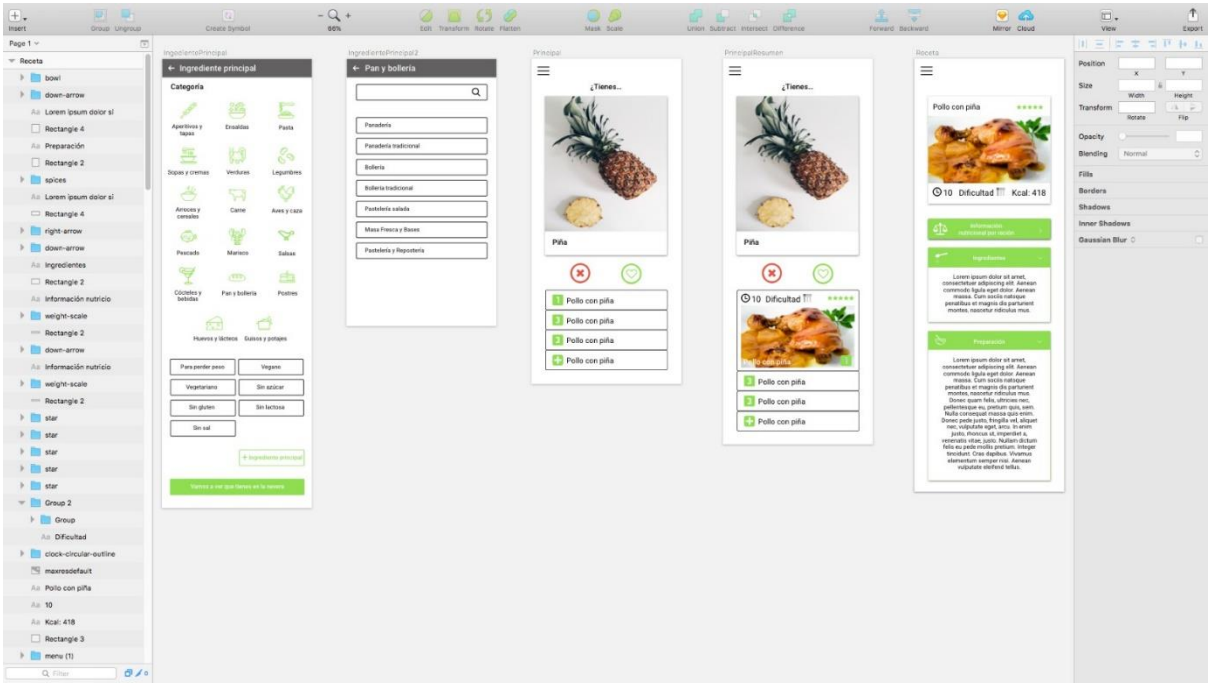
Tras terminar el proceso de investigación, conseguimos reunir y concretar todas las ideas propuestas para realizar los prototipos de las vistas de la aplicación.

Sketch es un conjunto de herramientas de diseño vectorial, orientado principalmente a interfaces de usuario. En el proceso de prototipado de la aplicación se utilizó esta herramienta para realizar bocetos, obteniendo a partir de ellos las vistas que definen la interacción del usuario con nuestra aplicación.

Utilizando herramientas de diseño para la generación de prototipos de interfaces de usuario podíamos tener una idea más clara de los requisitos que tenía que cumplir la aplicación, y la funcionalidad que se debía desarrollar, ya que desde el principio se tuvo una idea sobre cómo interactúa el usuario con la aplicación, sin necesidad de esperar a implementar las funcionalidades.

La interfaz de Sketch es muy intuitiva para un usuario inexperto. Entre otras cosas, permite disponer los distintos elementos en una rejilla, lo cual se adecua a la disposición de una interfaz de usuario móvil. Además, proporciona una función para exportar código HTML y CSS.

A continuación se muestra una captura que contiene la interfaz de Sketch, donde la barra superior contiene las principales herramientas para generar contenido y la barra lateral derecha muestra las características del elemento seleccionado.



Sketch fue seleccionada porque uno de los miembros del equipo realizaba prácticas en una empresa en la que se trabajaba con esta herramienta y contaba con el apoyo de expertos para conseguir los prototipos obtenidos.

3.4.7. VPS [22]

Para alojar el servicio que accede a la base de datos y la propia base de datos decidimos usar un VPS que ya teníamos contratado para uso personal de la empresa OVH.

Mediante los servidores virtuales privados (VPS) es posible disponer de varios servidores virtuales dentro de un único servidor físico, cada uno de ellos ejecutando un sistema operativo. En nuestro proyecto hemos utilizando un servidor virtual en el que ejecutaba el gestor de bases de datos MySQL, proporcionando servicios de acceso a nuestra base de datos de ingredientes y permitiendo, con ello, que nuestra aplicación realizase

peticiones a la base de datos. En el VPS también alojamos todas las imágenes de los ingredientes.

4. Acceso a la API de Yummly

Tal y como mencionamos en el capítulo anterior, en nuestro proyecto hemos decidido utilizar la base de datos proporcionada por Yummly para realizar la búsqueda de recetas. Es posible acceder a las recetas contenidas en esta base mediante una API pública. En este capítulo se describirán las características de esta API.

Yummly permite la búsqueda y filtrado de recetas basado en:

- Preferencias dietéticas

Se puede establecer la dieta que un usuario sigue (vegetariano, vegano, paleolítica, etc.), alergias que pueda tener (huevo, gluten, soja, etc.). También permite que el usuario vete determinados ingredientes, para evitar que aparezcan en los resultados.

- Tiempo de preparación

Es posible especificar el tiempo máximo del que se dispone para realizar la receta.

- Tipo de cocinas

También se puede indicar el estilo gastronómico de la receta: comida americana, china, hawaiana, de barbacoa o para niños.

- Plato

Con este filtro es posible indicar si se prefieren recetas de aperitivos, postres, ensaladas, snacks, bebidas, etc.

- Valores nutricionales

También dentro del ámbito de las restricciones dietéticas, es posible establecer restricciones sobre el aporte calórico de una receta, grasas, carbohidratos, colesterol, etc.

- Sabores

Cada receta tiene indicadores para saber cuánto es de salada, picante, dulce, etc. Es posible establecer un filtro sobre esta cualidad.

- Técnicas de cocinado

Mediante este filtro puede excluirse, por ejemplo, recetas que puedan requerir de un horno en caso de que el usuario no disponga de uno.

El servicio que usamos de Yummly es su API, que proporciona más de un millón de recetas, obtenidas a partir de los blogs más populares de comida y webs de recetas.

La API intenta ajustarse a los principios de REST. Es decir, las peticiones HTTP carecen de estado; se basa en operaciones sencillas como GET y POST, y en el uso de hipermedios. Los datos son devueltos en formato JSON y soporta JSONP como codificación alternativa. Gracias a JSONP pudimos superar el problema de XSS al realizar las llamadas a la API desde un JavaScript en un navegador. Los navegadores bloquean el uso del objeto XMLHttpRequest (XHR) al hacer una petición AJAX a dominios diferentes del de la propia página, por lo que al ser yummly un dominio externo tuvimos que utilizar JSONP que, en lugar de hacer una llamada con el objeto XHR, encapsula en una función javascript el objeto JSON.

Por ejemplo, en lugar de devolver un objeto del tipo `{receta:X}`, devuelve la llamada callback({receta:X}). Esa llamada se inserta en el DOM dentro de una etiqueta script y de esa forma podemos ejecutar la función *callback* con el JSON como parámetro.

El *endpoint* al que llamamos para usar la API es https://api.yummly.com/v1 y posee dos llamadas distintas para obtener información sobre recetas: *Get Recipe,* para obtener los datos de una receta específica y *Search Recipes* para buscar y filtrar recetas.

La principal condición para usar el servicio de la API de Yummly es aceptar los términos y condiciones del mismo, lo cual solo se garantiza creando una cuenta de desarrollador en su página y afirmando que se han leído, comprendido y aceptado.

Una vez creada la cuenta, para poder llamar a la API hay que añadir en la URL o en la cabecera HTTP unos identificadores facilitados por Yummly tras realizar la inscripción en cualquiera de planes que ofrece. Nosotros solicitamos una cuenta para estudiantes gratuita limitada a 30 mil

peticiones que hemos estado usando durante todo el desarrollo del proyecto.

Dado que estamos usando los servicios de Yummly y tanto en sus términos y condiciones como en la documentación de la API así lo pide, estamos obligados a mostrar atribución a la hora de mostrar las recetas. Las respuestas proporcionadas por la API incorporan, precisamente, etiquetas HTML y un archivo de imagen del logotipo de la empresa para facilitarnos el proceso.

Esta API solamente ofrece contenidos en inglés. Al principio del proyecto teníamos la intención de realizar una aplicación en castellano debido a que la API inicial que pretendíamos usar (Edamam) ofrecía también los contenidos en español. Otra razón es que muchos de nuestros familiares y amigos (no angloparlantes) nos pidieron, al conocer el proyecto, que cuando la aplicación estuviese lista se la facilitásemos, porque les interesaba para su día a día.

Al final, tras tomar la decisión de utilizar Yummly, quisimos evitar tener que traducir recetas, ingredientes y diversa información extra como categorías, datos nutricionales y textos de la aplicación. Por tanto, pensamos que sería mejor centrarse en la funcionalidad del producto y no dedicar una cantidad excesiva de tiempo a ofrecer la aplicación en castellano. No obstante, no descartamos que en un futuro la herramienta permita diferentes idiomas.

La gran ventaja que nos da esta API (aparte de su amplia gama de recetas e información relacionada) frente a otras de la competencia es que permite especificar una lista de ingredientes que no queremos que aparezcan en los resultados de búsqueda. Gracias a esto, podemos añadir en esta lista todos aquellos ingredientes que el usuario ha confirmado que no posee y, de esa manera, ninguna de las recetas que devuelve la API contiene dichos ingredientes.

Aún así, a nuestro parecer presenta algunas desventajas que nos han complicado el desarrollo, y que han hecho que nuestro algoritmo de búsqueda de recetas (que se describirá en el capítulo siguiente) sea más complejo:

- Disyunción de ingredientes admitidos

De la misma forma que la API acepta como parámetros ciertos ingredientes que no queremos que aparezcan en los resultados de búsqueda, también nos permite añadir ingredientes que sí queremos que aparezcan. Aunque el propósito sea similar (realizar un filtro por ingredientes), la forma de hacerlo difiere. El sistema devuelve aquellas recetas que contengan **todos** los ingredientes de la lista de admitidos, y que no contengan **ninguno** de los ingredientes de la lista de rechazados. Es decir, se realiza una *AND* de los ingredientes admitidos, y se realiza una *OR* de los rechazados. De forma que cuando añadimos dos ingredientes para excluir, saldrán recetas que no tienen ninguno de esos dos ingredientes, rechazando aquellas que tengan uno de ellos o los dos. Si esos dos ingredientes estuviesen en la lista de admitidos, solo saldrían recetas que incluyesen los dos a la vez. Debido a eso, tuvimos que realizar el algoritmo de que intentase maximizar la cantidad de recetas que tuviesen el mayor número de ingredientes afines al usuario, pero sin necesidad de tenerlos todos.

- Fotos de los ingredientes

Dado que la API no ofrece fotos de los ingredientes, nos vimos en la necesidad de disponer de una base de datos propia con imágenes de ingredientes, para así añadir en esta base de datos una ruta a la imagen del ingrediente y devolverla. Por otro lado, los ingredientes no están categorizados en la API, por lo que tuvimos que establecer distintas categorías de ingredientes y almacenarlas en la base de datos propia.

- Fotos con alta calidad de las recetas

Al pedir a la API una búsqueda de recetas, esta nos devuelve información básica sobre la misma como el identificador, título, ingredientes básicos y una foto, entre otros datos. Esa foto no suele ocupar demasiado y está pensada para ser mostrada en un formato reducido. Para poder mostrar una foto más grande tendríamos que hacer una consulta específica sobre cada receta obtenida, solicitando una imagen de mayor resolución, haciendo que se dispare el número de peticiones a la API. Dado que las llamadas que tenemos son limitadas, nos hemos limitado a mostrar la imagen de baja resolución.

- Falta de instrucciones de preparación de la receta

Dado que no disponemos de una lista de instrucciones por parte de la API pero sí del enlace a la receta, nos hemos visto obligados a redirigir a una página externa para poder mostrar la información de la receta, lo que se traduce en una experiencia algo más incómoda para el usuario, pues ya no es posible mostrar toda la información de una receta unificada en la misma aplicación.

Llamadas realizadas:

Como habíamos mencionado anteriormente, existen dos llamadas para el ámbito de las recetas: *Get Recipe* y *Search Recipes*. Adicionalmente, existe una llamada especial para obtener diccionarios de datos en los que muestra información sobre los tipos de cocina, ingredientes, dietas, etc. Para el desarrollo del proyecto no hemos considerado que esta última nos fuese de mucha utilidad.

Search Recipes

La llamada más utilizada por la aplicación. Su sintaxis básica sería:

```
http://api.yummly.com/v1/api/recipes?
    _app_id = nuestro_app_id &
    _app_key = nuestra_app_key &
    parametros_de_busqueda
```

El `app_id` y el `app_key` son los identificadores que nos facilitó Yummly a la hora de registrar nuestra aplicación.

De todos los parámetros de búsqueda existentes hemos usado los siguientes:

- `q`

Ingrediente principal. Aquí insertamos el ingrediente que el usuario escogió desde el menú principal de la aplicación.

- `allowed ingredient`

Array de Ingredientes que queremos que aparezcan en la receta. Se mostrarán aquellas recetas que contengan **todos** los ingredientes indicados en este array.

- `excluded ingredient`

Array de Ingredientes que no deseamos que aparezcan en los resultados. Se mostrarán aquellas recetas que no contengan **alguno** de los ingredientes especificados.

- `facetField`

Cuando se introduce este parámetro la llamada incorpora, además de los resultados de la búsqueda, un objeto llamado `facetCounts` que enumera la cantidad de ingredientes o dietas que han aparecido en la respuesta.

Realizamos esa llamada haciendo una búsqueda de todas las recetas que existen, con el parámetro `facetField` solicitando los ingredientes. De esta forma conseguimos el número de veces que aparece cada ingrediente y podemos conocer en todo momento qué ingredientes son más populares.

- `maxResult`

Máximo número de recetas que queremos que aparezcan. En nuestro caso, solicitamos 100 recetas para que el tiempo de respuesta de la API no se demore demasiado

Con estos parámetros, la respuesta de la API devuelve una lista de recetas, un contador, y un objeto con etiquetas *HTML* e imágenes para realizar la atribución de los resultados de búsqueda a Yummly.

En caso de que hubiésemos añadido el parámetro `facetField`, devuelve un objeto `facetCounts` con ingredientes y el número de recetas en las que aparece cada uno de los ingredientes.

A continuación, pasamos a describir el formato de las recetas devueltas por la llamada a `Search Recipes`. Por cada receta obtenemos:

- `imageUrlsBySize`

  Lista de enlaces a imágenes de la receta con diversos tamaños. En todas las recetas contempladas hasta el momento siempre hemos visto un único enlace a una imagen de tamaño muy reducido.

- `smallImageUrl`

  Miniatura de la receta. Muy similar a la imagen pequeña devuelta por `imageUrlsBySize`.

- `sourceDisplayName`

  Nombre del proveedor de la receta

- `ingredients`

  Lista de ingredientes de la receta

- `id`

  Identificador de la receta.

- `recipeName`

  Nombre de la receta

- `totalTimeInSeconds`

  Tiempo estimado para realizar la receta

- `attributes`

  Aquí se indica el estilo gastronómico al que pertenece, si puede contener algún alérgeno, qué tipo de plato es, etc.

- `flavors`

  Puntuación para cada uno de los sabores (salado, dulce, picante, etc.), cada una en un rango de 0 a 1.

Un ejemplo de esta llamada sería (el formato de las urls de los ejemplos han sido ligeramente modificadas para facilitar su lectura):

```
https://api.yummly.com/v1/api/recipes?
     callback = callback&
     _app_id = nuestro_app_id&
     _app_key = nuestra_app_key&
     maxResult = 1
     &excludedIngredient[] = granulated sugar&
     allowedIngredient[] = strawberry&
     allowedIngredient[] = strawberries&
     allowedIngredient[] = cool whip&
     allowedIngredient[] = honey graham crackers
```

Algunos parámetros del ejemplo son:

- `maxResult` a 1 para que solo nos devuelva una receta
- `excludedIngredient` con granulated sugar
- `allowedIngredient` con cuatro ingredientes: honey graham crackers, strawberries, strawberry y cool whip
- `callback` como nombre de función usando JSONP

La respuesta devuelta por la API se muestra a continuación:

```
callback({
   "criteria":{
      "allowedIngredient":[
         "honey graham crackers",
         "strawberries",
         "strawberry",
         "cool whip"
      ],
      "excludedIngredient":[
         "granulated sugar"
      ],
      "q":null
   },
   "matches":[
      {
         "imageUrlsBySize":{
            "90":"http://lh3.googleusercontent.com/mUH5o4d-
08W3XLSLw3-
BeJjhiw6bRryBxtRiroWr8PcHuHvLQ1rUMdR5uQ124K97KtqL8su7G5DPE-
9Ay3gnBw=s90-c"
         },
         "sourceDisplayName":"Sprinkle Some Sugar",
         "ingredients":[
```

```
            "strawberries",
            "honey graham crackers",
            "cool whip"
         ],
         "id":"Strawberry-Icebox-Cake-1105715",
         "smallImageUrls":[
            "http://lh3.googleusercontent.com/EN9UhNgtjS7Ue
VjBAcDuTnlwnzvFQZQlxOJFDZlpf178KUrxuxWlajr42B2VMmboZXKn4uYj
96fUxDDOB0j3j0A=s90"
         ],
         "recipeName":"Strawberry Icebox Cake",
         "totalTimeInSeconds":15000,
         "attributes":{
            "course":[
               "Desserts"
            ]
         },
         "flavors":{
            "sweet":0.6666666666666666,
            "piquant":0.0,
            "sour":0.8333333333333334,
            "salty":0.16666666666666666,
            "meaty":0.16666666666666666,
            "bitter":0.16666666666666666
         },
         "rating":4
      }
   ],
   "facetCounts":{
   },
   "totalMatchCount":3,
   "attribution":{
      "html":"Recipe search powered by <a
href='http://www.yummly.co/recipes'><img alt='Yummly'
src='https://static.yummly.co/api-logo.png'/></a>",
      "url":"http://www.yummly.co/recipes/",
      "text":"Recipe search powered by Yummly",
      "logo":"https://static.yummly.co/api-logo.png"
   }
});
```

Get Recipe

Nuestra aplicación solamente realizaba esta llamada si el usuario tenía interés en la receta y entraba a ver sus detalles. Su sintaxis básica es:

```
http://api.yummly.com/v1/api/recipe/id_receta?
     _app_id = nuestro_app_id &
     _app_key = nuestra_app_key
```

Dado que el único valor que realmente varía es el id de la receta (ya incluido en la URL), no es necesario incluir ningún parámetro de entrada.

Los datos que devuelve son:

- `nutritionEstimates`

Parámetros nutricionales. Forma una lista de nutrientes y la cantidad por ración. No aparecen en todas las recetas, solo en las que Yummly tiene seguridad sobre su precisión.

- `totalTime`

Tiempo estimado en realizar la receta.

- `images`

Array de imágenes de diferentes tamaños.

- `name`

Nombre de la receta.

- `source`

Datos sobre el origen de la receta.

- `id`

Identificador de la receta formado por palabras clave de la receta separadas por guiones (-) y un número al final. Ejemplo: Crock-Pot-Bbq-Chicken-Drumsticks-2422597

- `ingredientLines`

Lista de ingredientes detallados mostrando cantidad y, en ocasiones, detalles sobre cómo se necesita preparar el ingrediente (por ejemplo, troceado)

- `attribution`

Etiquetas *HTML*, enlaces y logotipo de Yummly para la atribución.

- `yield`

Número de raciones de la receta, mostrado en formato de texto (por ejemplo, “4 raciones”)

- `numberOfServings`

Número de raciones mostrado de forma numérica (por ejemplo, 4).

- `attributes`

Al igual que en la función `Search Recipes`, aquí se indica el tipo de cocina a la que pertenece, si puede haber algún alérgeno, qué tipo de plato es, etc.

- `flavors`

Aquí también se indica la puntuación para cada uno de los sabores en un rango de 0 a 1

- `rating`

Puntuación de la receta según la web de Yummly.
Un ejemplo de uso del servicio `Get Recipe` sería:

```
http://api.yummly.com/v1/api/recipe/Jello-Parfait-2423089?
     _app_id = nuestro_app_id &
     _app_key = nuestra_app_key &
     callback = callback
```

El único parámetro que se usa aparte del nombre de la función para utilizar JSONP, es el identificador como parte de la ruta que identifica a la receta (Jello-Parfait-2423089)

La API de Yummly devuelve:


```
callback({
   "nutritionEstimates":[
   ],
   "totalTime":"20 min",
   "images":[
      {
         "hostedSmallUrl":"https://lh3.googleusercontent.co
m/Z9YkfhbVxTrg8oxy3zdIVo_2kXNF8Z5-
zQFN0iQtoPnC00PX0Nt3D6bvbuwwaqlwaDg_8f4sY7xk1aMfuGnSlA8=s90
",
         "hostedMediumUrl":"https://lh3.googleusercontent.c
om/Z9YkfhbVxTrg8oxy3zdIVo_2kXNF8Z5-
zQFN0iQtoPnC00PX0Nt3D6bvbuwwaqlwaDg_8f4sY7xk1aMfuGnSlA8=s18
0",
         "hostedLargeUrl":"https://lh3.googleusercontent.co
m/Z9YkfhbVxTrg8oxy3zdIVo_2kXNF8Z5-
zQFN0iQtoPnC00PX0Nt3D6bvbuwwaqlwaDg_8f4sY7xk1aMfuGnSlA8=s36
0",
         "imageUrlsBySize":{
            "90":"https://lh3.googleusercontent.com/BuWOY84
hpR1BhXB0_VAEOXH1TI9y37pkXEi_i5Nc17M0Zf21fSq41FnXw5lKI-
1uPTq4YJSOZ3SdTuVsrBlypA=s90-c",
            "360":"https://lh3.googleusercontent.com/BuWOY8
4hpR1BhXB0_VAEOXH1TI9y37pkXEi_i5Nc17M0Zf21fSq41FnXw5lKI-
1uPTq4YJSOZ3SdTuVsrBlypA=s360-c"
         }
      }
   ],
   "name":"Jello Parfait",
   "source":{
      "sourceDisplayName":"Virtually Yours",
      "sourceSiteUrl":"kellystilwell.com",
      "sourceRecipeUrl":"https://kellystilwell.com/festive-
delicious-jello-parfait/"
   },
   "id":"Jello-Parfait-2423089",
   "ingredientLines":[
      "1 package (3oz) Strawberry Jello-o",
      "1 cup sliced fresh strawberries (optional)",
      "1 cup thawed Cool Whip whipped topping"
   ],
   "attribution":{
      "html":"<a href='http://www.yummly.co/recipe/Jello-
Parfait-2423089'>Jello Parfait recipe</a> information
powered by <img alt='Yummly'
src='https://static.yummly.co/api-logo.png'/>",
```

```
      "url":"http://www.yummly.co/recipe/Jello-Parfait-
2423089",
      "text":"Jello Parfait recipes: information powered by
Yummly",
      "logo":"https://static.yummly.co/api-logo.png"
   },
   "numberOfServings":4,
   "totalTimeInSeconds":1200,
   "attributes":{
      "course":[
         "Desserts"
      ]
   },
   "flavors":{

   },
   "rating":4
});
```

5. Algoritmo de búsqueda de recetas

En este capítulo se describirá el algoritmo que realiza la búsqueda de recetas en función de la información que el usuario, de manera incremental, proporciona definiendo los ingredientes de los que este dispone y de los que no dispone.

Uno de los principales problemas en el desarrollo de la aplicación fue la eficiencia del algoritmo de generación de recetas a partir de ciertos ingredientes.

Desde un punto de vista funcional la aplicación proporciona ciertos ingredientes con los que el usuario interacciona indicando si dispone de ellos (like) o no (dislike). Basándose en estas decisiones el algoritmo va generando recetas que contienen los ingredientes seleccionados y no los descartados.

Por tanto el objetivo del algoritmo era obtener las recetas que se le mostrarían al usuario basándose en los ingredientes previamente aceptados y descartados y mostrar el siguiente ingrediente sobre el que usuario debe decidir.

5.1 Problemas principales

Desde el principio hubo dos grandes problemas que alargaron más de lo esperado el desarrollo de esta lógica:

- **Número limitado de peticiones a la API**: al inicio del proyecto solicitamos a los dueños de Yummly una cuenta gratuita con la que poder hacer nuestro trabajo de fin de grado. Ellos generosamente nos concedieron una cuenta gratuita con límite de 30.000 llamadas. Debido a este límite debíamos elaborar un algoritmo que limitase, en la mayor medida posible, el número de llamadas, consiguiendo al final una única llamada a la API por cada ciclo del algoritmo, a excepción de un caso en el que hay que realizar una búsqueda más exhaustiva (OrderingCalls). Entendemos que un ciclo del algoritmo comprende el procesamiento que se realiza desde que el usuario selecciona la disponibilidad o no de un ingrediente hasta que el

sistema le propone otro ingrediente distinto, y actualiza la lista de recetas.

- **Tiempo de ejecución**: Debido al límite en el número de llamadas a la API, nuestra primera opción fue obtener la máxima cantidad de información posible en cada una de las llamadas a la API para reducir el número de peticiones y así gestionar toda la información que necesitamos en tiempo de ejecución. Sin embargo, las peticiones con mucha información conllevan un elevado tiempo de espera. Por ejemplo, el tiempo necesario para solicitar 300 recetas podía estar en torno a los 3 segundos. Al estar desarrollando una aplicación para dispositivos móviles, los tiempos de espera que tiene que asumir el usuario deben ser reducidos para que la experiencia de usuario sea lo más cómoda y fluida posible.

Una vez planteados los dos problemas iniciales y definido el objetivo final empezamos a diseñar diferentes propuestas hasta que logramos construir el algoritmo actual que consigue un compromiso adecuado entre los tiempos de espera y el límite en el número de llamadas.

5.2 Formato actual del algoritmo

5.2.1 Variables utilizadas

`var nRecipes = 100;` // Número límite de recetas que la petición a la API devolverá

`var min_recipes = 10;` // Número mínimo de recetas que se deben tener siempre guardadas en la lista de recetas en tiempo de ejecución.

`var principal ='';` // *String* que contiene el ingrediente principal elegido por el usuario.

`var LSelected = [];` // Lista de ingredientes que el usuario ha seleccionado que SÍ tiene.

`var LExcluded = [];` // Lista de ingredientes descartados por el usuario.

`var LRecipeAssessted = [];` // *Array* con las recetas ya procesadas por algoritmo, tanto las que se han descartado (debido a que el usuario carece

de alguno de sus ingredientes) como las que aún son candidatas a ser realizadas..

`var HTopIng = new Object();` // *HashMap* que contiene en la clave el nombre del ingrediente y en el valor en número de recetas de la API en las que aparece dicho ingrediente.

`var LRecipesUser = [];` // Lista de recetas candidatas para el usuario en función de la disponibilidad de ingredientes, es decir, lista de aquellas recetas que no contienen ningún ingrediente que haya sido descartado por el usuario, por no disponer de él. Dentro de esta lista, las recetas están ordenadas, de menor a mayor, en función del número de ingredientes de los que aún no se conozca su disponibilidad.

5.2.2 Resumen

A continuación se explica en términos generales y sin detalles cómo está diseñado el algoritmo. En las siguientes secciones se precisará más cada una de las fases del mismo.

Primero existe una fase de preparación en la que se inicializan todas las variables, se elige un ingrediente principal y muestra al usuario el primer ingrediente con el que interactuar. Concretamente la inicialización de `HTopIng` es bastante delicada y se explicará más adelante.

Al final de esta fase de preparación se obtendrán 10 recetas ordenadas en `LRecipesUser`, el ingrediente principal seleccionado por el usuario en la variable `principal`, la lista inicial de ingredientes `LSelected` que contendrá únicamente el ingrediente principal. En `LRecipeAssessted` estarán las recetas incluidas en `LRecipesUser`, ya que son las que se han sido procesadas por el algoritmo, y se obtendrá el ingrediente más prioritario para mostrarse al usuario. El orden de `LRecipesUser` viene definido por el número de ingredientes cuya disponibilidad aún no ha sido indicada por el usuario. Se mostrará en primer lugar la receta que contenga un menor número de ingredientes por comprobar.

Una vez que se ha mostrado el ingrediente más prioritario, el usuario deberá indicar si dispone de él o no. En este punto, el algoritmo se divide en dos ramas según acepte o no dicho ingrediente el usuario.

En caso de aceptarlo, el algoritmo añade el ingrediente a los seleccionados (`LSelected`), y hace una llamada a la función `Search Recipes` de la API con los ingredientes seleccionados y excluidos actualmente, con el límite indicado en `nRecipes`. De esas recetas obtenidas, selecciona 10 y las añade a la lista de `LRecipesUser` respetando el orden, ya comentado anteriormente, basado en el número de ingredientes que queden por ser confirmados por el usuario.

En caso de que el usuario haya descartado el ingrediente, se añade a los descartados (`LExcluded`) y se eliminan las recetas que contengan dicho ingrediente de `LRecipesUser`. En caso de que esta lista haya quedado con un número de recetas inferior al mínimo establecido (`min_recipes`) se procederá a realizar una llamada a la función `Search Recipes` de la API. Si, aun añadiendo las recetas contenidas en la respuesta no se tiene el mínimo de recetas necesarias se llamará a una función llamada `orderingCalls`, cuyo funcionamiento se explicará más tarde, ya que fue una de las funciones más complicadas del algoritmo.

Una vez modificada la lista `LRecipesUser` se procede a extraer el ingrediente más "popular" de la primera receta de esta lista para mostrárselo al usuario y así volver al inicio del algoritmo.

Para terminar, existe una fase de finalización. En cualquier momento del proceso, el usuario puede seleccionar una de las recetas de `LRecipesUser` para cocinarla, y en ese momento la aplicación obtiene, mediante una llamada a la función `Get Recipe` de la API, toda la información de dicha receta y la muestra al usuario.

5.2.3 El algoritmo

En este apartado se explica en detalle cada parte del algoritmo separándolo en 3 fases: preparación, acción y finalización.

5.2.3.1 Fase de preparación

Antes de la ejecución del algoritmo existen ciertos pasos que preparan e inicializan las variables que se usarán más tarde.

Al inicio todas las variables contienen un array vacío excepto `nRecipes` y `min_recipes`, pero para el inicio del algoritmo necesitamos asignar un valor distinto a algunas de las variables.

En primer lugar, se inicializa `HTopIng`, con el fin de obtener un HashMap que utilizaremos para elegir los ingredientes más usados y así mostrar a los usuarios los ingredientes más comunes en las recetas. Al tener este mapa podremos reducir el número de descartes de ingredientes ya que se proporcionan al usuario los ingredientes más utilizados, siendo estos por tanto los más comunes en casa.

Para inicializar la variable se realiza una petición a la API (con el atributo facetField, explicado en el capítulo 4) que devuelve el número de recetas en las que aparece cada ingrediente, de manera que obtendremos un mapa con clave el ingrediente y valor su número asociado.

Una vez inicializadas las variables, se le pide al usuario que seleccione un ingrediente principal de entre los que le proporcionamos. Este ingrediente será asignado a la variable `principal`.

Ahora ya comienza la fase de acción tomando dicho ingrediente principal como el primer ingrediente que ha sido aceptado.

5.2.3.2 Fase de acción

La fase de acción determina las acciones a realizar desde que el usuario acepta o descarta un ingrediente hasta que se le muestra el siguiente ingrediente con la lista actualizada de recetas candidatas.

En función de si el usuario acepta o descarta el ingrediente se debe actuar de manera totalmente distinta. Por ello, la primera parte del algoritmo se divide en dos casos, aceptación o descarte del ingrediente, y finaliza con una parte común a ambas bifurcaciones.

A. Caso de aceptación

En este caso se insertará el ingrediente aceptado en la lista de seleccionados `LSelected`. Después actualizará el número de ingredientes cuya disponibilidad está aún por determinar en cada una de las recetas de `LRecipesUser`. A continuación se procederá a llamar a la función `Search Recipes` de la API con los ingredientes seleccionados y los descartados,

recibiendo una lista con un máximo de `nRecipes` recetas que cumplen los requisitos.

Tras obtener estas recetas de la API debemos procesarlas. Primero eliminaremos todas aquellas que ya pertenezcan a `LRecipeAssessted` ya que serían recetas que ya se han recuperado previamente, y por tanto o ya han sido descartadas o están actualmente en `LRecipesUser`. Después ordenamos las recetas restantes por el número de ingredientes que faltan por aceptar o descartar y obtenemos las 10 primeras, que añadimos a `LRecipesUser` y a `LRecipeAssessted`. Entonces ya sólo queda ordenar `LRecipesUser` y ya tendríamos actualizada la lista de recetas que se muestran al usuario.

B. Caso de descarte

En este caso tenemos como entrada el ingrediente que el usuario ha descartado, el cual incluimos en la lista de ingredientes descartados `LExcluded`. A continuación borramos de `LRecipesUser` las recetas que contienen ese ingrediente.

Si, tras la eliminación, se tiene un número de recetas superior al mínimo (`min_recipes`), se llama a la API igual que en el caso de aceptación, con los ingredientes seleccionados y descartados, y se procesa la respuesta de la manera indicada anteriormente.

En caso de no tener un mínimo de recetas pasamos a una función llamada `orderingCalls()` que se encarga de obtener ese mínimo en caso de ser posible .

`OrderingCalls(nLlamadas, callback)`*:* Esta función recursiva recibe un valor numérico, que permite delimitar el número de llamadas recursivas, y una función *callback*. Entonces selecciona de manera aleatoria un ingrediente perteneciente a `LSelected`, comprueba que esta lista contenga más de un ingrediente, además del seleccionado (en caso contrario el ingrediente seleccionado hubiera sido el principal), y que ese ingrediente seleccionado aleatoriamente no sea el principal. Después suma uno a `nLlamadas` y realiza una petición a la API con esos dos ingredientes cono seleccionados únicamente y todos los descartados. En caso de haber un solo ingrediente realizaría la llamada únicamente con ese ingrediente,

que sería el principal. Tras procesar la respuesta como siempre, si no existe el mínimo de recetas requerido y `nLlamadas` aún no han llegado al máximo establecido (10 actualmente), volverá a llamar a la misma función, `orderingCalls(nLlamadas, callback)`.

```
function orderingCalls(nLlamadas, callback){
  var ingredient = _.sample(LSelected);
  var select = [];
  var maxLlamadas = 9;
  if (LSelected.length > 1){
    while (ingredient === principal) {
      ingredient = _.sample(LSelected);
    }
    select = [principal, ingredient]
  }
  else{
    select = [principal]
  }
  callAPI(select,function () {
    if (LRecipesUser.length >= min_recipes || nLlamadas == maxLlamadas){
      callback(getNextIngredient(LRecipesUser));
    } else {
      nLlamadas++;
      orderingCalls(nLlamadas, callback);
    }
  })

}
```

C. Parte final (Común a ambos casos)

Una vez que hemos obtenido las recetas (`LRecipesUser`), debemos obtener el ingrediente que mostraremos al usuario. Para ello se obtiene la primera receta de la lista, que al estar ordenada será la que esté más cerca de ser aceptada, siendo la que menos ingredientes que faltan por aceptar o descartar le quedan. Observamos dichos ingredientes y de ellos mostramos al usuario aquel que tiene un mayor valor en `HTopIng`, ya que, al ser el ingrediente más usado, existe una mayor probabilidad de que el usuario disponga de él.

5.2.3.3 Fase de finalización

La fase de finalización empieza en el momento en el que el usuario se decide realizar una de las recetas que se le ofertan. Entonces el algoritmo realizará una llamada a la API mediante la función `Get Recipe` con el Id de

dicha receta, obteniendo toda la información que se le mostrará al usuario: Información nutricional, ingredientes y sus cantidades, enlace a la web donde se detalla la preparación.

5.3 Intentos previos del algoritmo

Desde que comenzamos a pensar en el algoritmo tuvimos muy presente el hecho de tener limitado el número de llamadas a la API. Nuestro objetivo desde el principio era conseguir que el usuario llegase a una receta de su gusto lo antes posible, y eso se podía conseguir con un mayor procesado de recetas que diese prioridad a aquellas recetas que contengan un mayor número de ingredientes aceptados por el usuario. Por eso se pensó inicialmente que lo mejor era obtener, con un número reducido de llamadas, la mayor cantidad posible de datos de la API para así ser procesados en ejecución.

Tras realizar diferentes pruebas sencillas de peticiones de búsqueda basándonos en diferentes ingredientes que teníamos y que habíamos descartado nos dimos cuenta de dos aspectos muy importantes que condicionaron la manera de diseñar el algoritmo. Por un lado, cuando se solicitaba un máximo de 800 recetas, el tiempo de respuesta de la API se disparaba, haciendo inviable intentar obtener un volumen elevado de datos para ser procesado. Por otro lado, si intentábamos hacer una búsqueda muy restrictiva, es decir, con muchos ingredientes requeridos y descartados, no obteníamos ninguna receta que cumpliese estos términos.

Llegados a este punto nos dimos cuenta de que el desarrollo de el algoritmo dependía muchísimo de las limitaciones de la API, teniendo como problemas principales el no poder hacer ni un gran número de llamadas ni hacer llamadas que devuelvan gran cantidad de datos.

Al estar atascados en este problema, decidimos intentar otro enfoque en el diseño del algoritmo y nos centramos en determinar cómo obtener el siguiente ingrediente que se mostraría al usuario. Nos dimos cuenta de que realmente siempre hay ciertos ingredientes más comunes en una cocina que otros, y curiosamente estos mismos ingredientes se utilizaban en muchísimas recetas diferentes. Comprendimos que aquellos ingredientes que más salen en las recetas son los que se suelen tener en

casa y por tanto, eran los que teníamos que mostrar antes al usuario. Para ello elaboramos un *HashMap* (`HTopIng`) que contenía como clave el nombre del ingrediente y valor el número de recetas en las que aparece.

Tras tener definido el criterio a utilizar para obtener el ingrediente a mostrar teníamos que saber de dónde íbamos a obtener dicho ingrediente. Lo lógico era obtenerlo de la receta que menos ingredientes por determinar contuviese, ya que el objetivo es conseguir que el usuario pueda realizar una receta cuanto antes. Por tanto, debíamos tener las recetas ordenadas de menor a mayor por el número de ingredientes que les faltasen por seleccionar.

Ahora entra en cuestión cómo íbamos a guardar y obtener esas recetas. Nuestra primera idea implicaba que el algoritmo construyese un árbol donde cada nodo correspondería a una lista de recetas como resultado de una llamada a la API, teniendo en las hojas las llamadas de un solo ingrediente y en la raíz la llamada con todos los ingredientes seleccionados hasta el momento, y siempre teniendo en cuenta que cada llamada a la API debía estar restringida por 100 recetas.

El funcionamiento era el siguiente: tras una llamada se guardaban las 100 recetas en el nodo correspondiente y se obtenían las 10 “mejores”, es decir, las 10 con el menor número de ingredientes por aceptar o descartar, que íbamos comparando con las que ya habíamos obtenido como mejores antes y actualizaríamos esa lista, que sería la lista mostrada al usuario. De esa lista obtendríamos el siguiente ingrediente y si el usuario lo rechazaba hacíamos la misma llamada anterior con uno más en descartados sobrescribiendo ese nodo. En caso de aceptar el ingrediente añadíamos un nodo raíz con la lista de recetas de una nueva petición con todos los ingredientes.

Pero este algoritmo tenía un grave inconveniente, ya que, cuantos más ingredientes hubiese, menos recetas habría en la raíz del árbol y por tanto más difícil volver a rellenar la lista del usuario, ya que habría que ir recorriendo los nodos siguientes hasta encontrar recetas previamente no gestionadas.

Tras un tiempo valorando diferentes alternativas, nos dimos cuenta de que la clave consistía en tener almacenadas en la lista del usuario el

mayor número de recetas válidas posible, aunque no le mostrásemos todas. De esta manera, cada vez que el usuario aceptase o descartase un ingrediente, se actualizase esa lista con las nuevas incorporaciones de la llamada (solo una selección de las que devuelve la API) y eliminase aquellas que ya no valían en caso de contener el ingrediente descartado.

Sólo quedaba el problema que se produciría en caso de que el usuario rechazase tantos ingredientes que la lista de recetas del usuario se vaciase. En ese caso podría pensarse en volver a realizar una búsqueda de aquellas recetas que contuviesen todos los ingredientes aceptados hasta el momento, pero nos dimos cuenta que si había demasiados ingredientes aceptados la respuesta de la API sería vacía, ya que, por ejemplo, no existen recetas que tuviesen 20 ingredientes distintos. Por tanto, decidimos que era más importante conseguir recetas de manera rápida que buscar la receta con más ingredientes ya seleccionados y por ello desarrollamos la función `orderingCalls` que simplemente llamaba a la API con un ingrediente aleatorio seleccionado y el principal hasta obtener el número mínimo de recetas requerido.

6. Arquitectura de la aplicación de recetas

En este capítulo se detalla la estructura de la aplicación, haciendo hincapié en la arquitectura divida en módulos, en el desarrollo del *backend* de la base de datos y los servicios *web* y el diseño de las vistas.

6.1 Base de datos y servidor propio

Aunque el proyecto que desarrollamos es una aplicación móvil, nos vimos en la necesidad de apoyar su funcionamiento en un servidor externo (VPS alojado en OVH [22]), debido a dos situaciones a las que nos enfrentamos:

- Imágenes de los ingredientes

Para mejorar la experiencia de usuario, vimos oportuno mostrar una imagen del ingrediente por el que preguntábamos al usuario. La API de recetas de la que hemos hecho uso (Yummly) no proporcionaba dicha imagen, por lo que debíamos proporcionarla nosotros mismos. En caso de que hubiésemos almacenado esas imágenes en la propia aplicación, habría aumentado considerablemente la cantidad de espacio en disco utilizada, y el tamaño del fichero apk utilizado para distribuir la aplicación. Unificando la fuente de esas imágenes en un servidor también nos da la ventaja de que, en caso de ir añadiendo en el futuro más fotografías, el usuario podría disfrutarlas al instante, sin tener que actualizar la app a la siguiente versión.

- Categoría de los ingredientes

Nada más abrir la aplicación se le ofrece al usuario la opción de escoger un ingrediente como ingrediente principal de sus recetas. Para poder ayudar en ese proceso de selección al usuario, hemos categorizado los ingredientes más relevantes. Esta categorización tampoco viene proporcionada por Yummly.

Dado que necesitamos información sobre los ingredientes y en un futuro puede que necesitemos más, decidimos usar una base de datos con toda esa información y un servicio web para poder acceder a esos datos.

6.1.1 Base de datos

En el servidor también hemos desplegado una base de datos MySQL para alojar todos los datos que necesitemos sobre los ingredientes.

La base de datos se compone de una tabla descrita de la siguiente manera:

Field	**Type**	**Null**	**Key**	**Default**	**Extra**
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(128)	YES		NULL	
picture	varchar(128)	YES		NULL	
category	varchar(128)	YES		NULL	

El campo **id** es un campo numérico autoincremental que sirve de clave primaria.

Los campos **name** y **category** son *strings*.

El campo **picture** guarda la ruta de la imagen del ingrediente tomando como directorio raíz la carpeta `/static` dentro del proyecto *Spring Boot*.

Para poder utilizar la base de datos creamos un usuario llamado *recipeers* y le otorgamos permisos de selección, inserción y borrado de filas para poder ser utilizado por el servicio *Spring Boot*.

Para la creación y mantenimiento de la tabla, así como inserción y actualización de las filas utilizamos una cuenta diferente que posee más permisos que la de *recipeers*.

6.1.2 DB Services

DB Services es una aplicación desarrollada en *Spring Boot*. Proporciona varios servicios web que sirven de acceso a la base de datos. Ofrece una API de tipo REST para ser utilizada por la aplicación móvil con las llamadas:

- `/getall:` devuelve un JSON con todos los ingredientes de la base de datos

Tipo: GET

Parámetros: Sin parámetros de entrada

Devuelve: *Array* con los siguientes parámetros:

Campo	Tipo	Descripción	Nullable
id	Integer	Id del ingrediente	No
name	String	Nombre del ingrediente	Sí
picture	String	Ruta de la imagen del ingrediente	Sí
category	String	Categoría del ingrediente	Sí

Ejemplo de respuesta:

```
[
   {
      "id":143,
      "name":"russet potatoes",
      "picture":null,
      "category":null
   },
   {
   "id":430,
   "name":"lemon",
   "picture":"/lemon.jpeg",
   "category":"fruit"
   },
...
```

- `/getingredient`: recibe el identificador del ingrediente por parámetro y devuelve los datos del mismo

  Ejemplo: `/getingredient?id=430`

Tipo: GET

Parámetros:

Campo	Tipo	Descripción	Nullable
id	Integer	Id del ingrediente	Sí

- En caso de que no se aporte dicho id, la respuesta será null

Devuelve: JSON con los siguientes parámetros:

Campo	Tipo	Descripción	Nullable
id	Integer	Id del ingrediente	No
name	String	Nombre del ingrediente	Sí
picture	String	Ruta de la imagen del ingrediente	Sí
category	String	Categoría del ingrediente	Sí

Ejemplo de respuesta:

```
{
   "id":430,
   "name":"lemon",
   "picture":"/lemon.jpeg",
   "category":"fruit"
}
```

- `/getingredientsbycategory`: devuelve una lista de ingredientes de una categoría. Los ingredientes vienen ordenados alfabéticamente.

  Ejemplo: `/getingredientsbycategory?category=birds`

  Tipo: GET

Parámetros:

Campo	Tipo	Descripción	Nullable
category	String	Nombre de la categoría	Sí

- En caso de que no se aporte la categoría, la respuesta será un *Array* vacío.

Devuelve: *Array* con los siguientes parámetros:

Campo	Tipo	Descripción	Nullable
id	Integer	Id del ingrediente	No
name	String	Nombre del ingrediente	Sí
picture	String	Ruta de la imagen del ingrediente	Sí
category	String	Categoría del ingrediente	Sí

Ejemplo de respuesta:


```
[
    {
        "id":435,
        "name":"chicken",
        "picture":null,
        "category":"birds"
    },
```

```
    {
        "id":1113,
        "name":"chicken wings",
        "picture":null,
        "category":"birds"
    },
    {
        "id":1112,
        "name":"skinless chicken breast",
        "picture":"/chicken_breast.jpeg",
        "category":"birds"
    },
    {
        "id":948,
        "name":"turkey",
        "picture":null,
        "category":"birds"
    }
]
```


- `/getpicturebyname`: devuelve la ruta de la imagen del ingrediente solicitado

  Ejemplo: `/getpicturebyname?name=kosher%20salt`

Tipo: GET

Parámetros:

Campo	Tipo	Descripción	Nullable
name	String	Nombre del ingrediente	Sí

- En caso de que no se aporte el nombre del ingrediente, la respuesta será null

Devuelve: *String* con la ruta de la imagen

Ejemplo de respuesta: `"/kosher_salt.jpg"`

El servidor web, además de ofrecer esta API, aloja las imágenes de los ingredientes en un directorio público. Cuando la aplicación desde el cliente muestra el siguiente ingrediente al usuario hace una petición a

`/getpicturebyname` y si la respuesta no es nula (imagen no encontrada), cambia el atributo src de la imagen apuntando hacia la ruta devuelta por DB Services.

Esta API está preparada para devolver una respuesta JSONP [23] y así evitar el problema que supone no poder hacer una petición AJAX usando el objeto XMLHttpRequest (XHR) al igual que explicamos al principio del capítulo 4.

La forma de acceder a la base de datos por parte del servicio es mediante la interfaz Repository [24], que ofrece Spring, junto con el uso de entities de JPA (Java Persistence API).

Para utilizar esta interfaz, asignamos a la clase Ingredient una etiqueta `@Entity` y creamos una clase aparte llamada IngredientRepository que extiende a la clase `Repository<Ingredient, Integer>`. El primer parámetro de tipo (`Ingredient`) indica el tipo que utilizamos para representar cada fila de la tabla de ingredientes. El segundo parámetro (`Integer`) indica el tipo de la clave primaria de esta tabla.

De esta forma, Spring puede generar automáticamente métodos de consulta a la BD en cualquier clase que extiende a *Repository*. Para ello infiere el tipo de consulta a realizar a partir del nombre de dicho método. En particular, en `IngredientRepository` podíamos crear un método con el siguiente nombre:

```
List<Ingredient> findByCategoryOrderByName(String
category);
```

Este método devuelve un *ArrayList* con los ingredientes pertenecientes a la categoría recibida por parámetro, ordenados alfabéticamente por su nombre.

6.2 Aplicación móvil

La arquitectura de la aplicación vino definida por dos aspectos claves: la necesidad de una sola página HTML (modelo SPA) y una organización estructural bien definida y dividida en capas que permitiese desarrollar evolutivos de manera sencilla.

Además, al tratarse de ser una aplicación multiplataforma, decidimos usar Apache Cordova [6] para no tener que elaborar aplicaciones separadas para cada plataforma (Android, iOS, etc.). De este modo, desarrollamos la aplicación utilizando tecnologías web, y Cordova transforma nuestra aplicación web en una aplicación móvil para la plataforma correspondiente.

6.2.1 Modelo SPA (Single Page Application)

Al tratarse de una aplicación móvil programada en HTML fue necesario utilizar un modelo SPA, de manera que todos los elementos de la interfaz se cargasen al iniciar la aplicación, y se mostrasen, ocultasen o manipulasen dinámicamente en tiempo de ejecución, sin necesidad de cambiar de página web, así obteniendo una experiencia de usuario más fluida.

En una aplicación con múltiples páginas, la división entre los distintos recursos estáticos resulta natural, ya que cada página incluye solamente los recursos que utiliza. Sin embargo, en una aplicación con modelo SPA, al tener un único fichero HTML, somos nosotros los responsables de dividir el código JavaScript en módulos, donde cada uno de ellos maneja sus correspondientes recursos. Para esta división utilizamos RequireJS.

RequireJS [12] es capaz de gestionar tanto las dependencias con librerías externas como la carga de los diferentes módulos y archivos Javascript de la aplicación, permitiendo tener una estructura dividida en módulos además de ganar tiempo de carga de archivos, ya que RequireJS realiza la carga de los archivos de forma asíncrona, permitiendo la carga de varios módulos JavaScript en paralelo. Además junta los archivos en uno sólo reduciendo el número de peticiones al servidor, a la vez minifica dichos archivos haciendo que las conexiones sean más rápidas. Por otro lado, al gestionar las dependencias evita conflictos entre librerías.

Aún existen algunas librerías que no están del todo optimizadas para usar RequireJS y es necesario realizar especificaciones sobre el orden de carga de los módulos de dicha librería o realizar un "calzado" (shim) para que funcione. Es el caso de Materialize, que no está optimizado para usarse con RequireJS, lo que conllevó bastantes problemas a la hora de incluir las dependencias de Materialize, por lo que al final tuvimos que incluir Materialize directamente dentro del documento HTML mediante una etiqueta `<script>`, no pudiendo integrarlo a través de RequireJS.

RequireJS organiza los archivos en módulos, los cuales pueden depender de otros. Existe un único módulo principal (normalmente llamado `main.js` o `app.js`) en el que se especifica el nombre de todos los módulos utilizados en la aplicación. Este módulo principal es el que ha de ser especificado en el atributo `data-main` cuando se carga la librería RequireJS mediante la etiqueta `<script>`.

```
<script data-main="js/main"
src="js/lib/require.js"></script>
```

Además, en el módulo principal se especifica una función llamada `config` en la que se define la configuración de la arquitectura y sus dependencias. Dentro de este mismo módulo, existe una función llamada `requirejs` en la se especifican los módulos dependientes de *main*, y se ejecutan las acciones necesarias para el arranque de la aplicación. Esto es similar al método *main* de una aplicación escrita en Java.

```
requirejs(["front"], (front) => {
  console.log("Modules loaded");
  front.init();
});
```

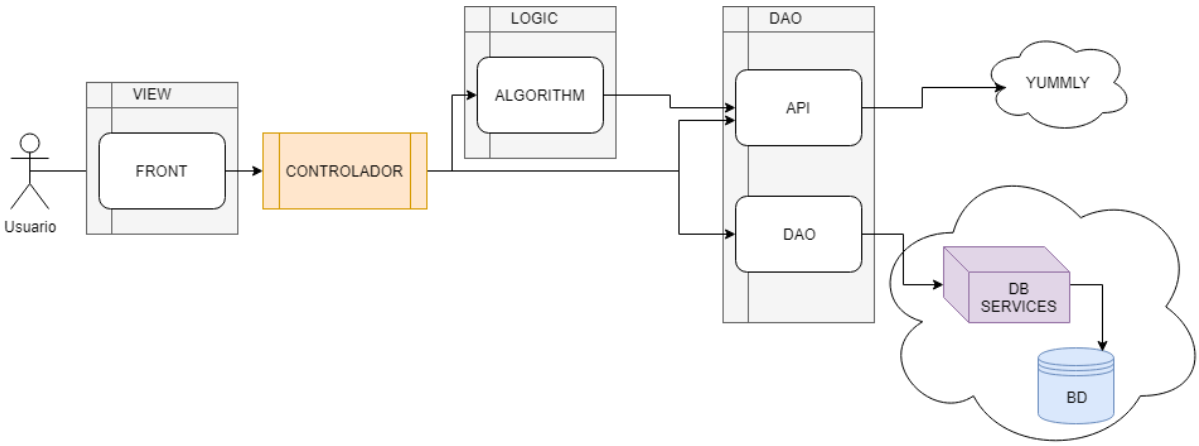
Gracias a RequireJS conseguimos una estructura de módulos en un modelo SPA que además de facilitar una carga rápida de archivos, gestiona las dependencias entre módulos, prohibiendo referencias circulares entre módulos. Por ejemplo, si el módulo controlador depende de otro llamado `algorithm`, este último no puede depender del controlador. Dividiendo la aplicación en distintos módulos se obtiene una arquitectura escalable y bien segmentada.

6.2.2 Arquitectura

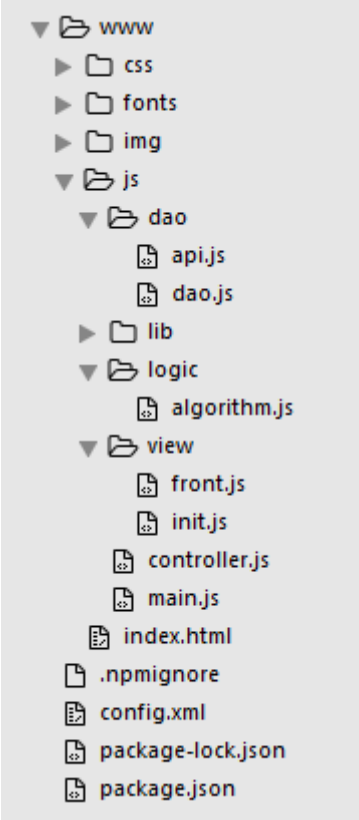
Desde el inicio del proyecto, se tuvo en mente la importancia de generar un código escalable y que permitiese realizar desarrollos incrementales de una manera sencilla. Por este motivo se definió una arquitectura dividida en cuatro capas bien definidas en la aplicación gracias a RequireJS: View, Controller, Logic, DAO, que se corresponden con las capas de presentación, controlador, negocio e integración de la arquitectura modelo vista-controlador.

La decisión de distribuir de esta manera los diferentes módulos vino principalmente motivada por la expectativa de poder incluir nuevas mejoras y funcionalidades de una manera controlada, sencilla y sobre todo que no afectase a lo ya realizado, de manera que el módulo View se encargase de las vistas y la interacción con el usuario, el módulo Controller de la conexión entre los módulos View, Logic y DAO, el módulo Logic únicamente de la lógica y gestión de los datos y el módulo DAO aportase el soporte de conexiones con la base de datos, API y el servidor.

Tenemos, por tanto, el siguiente esquema que representa la interacción entre los distintos módulos:



Debido a esta división en módulos la estructura de carpetas de la aplicación quedó de la siguiente manera:



6.2.2.1 Módulo View

Esta capa es la encargada de manejar la comunicación directa con el usuario.

Contiene dos archivos JavaScript: `init` y `front`.

El archivo `init` inicializa algunos elementos de Materialize, como por ejemplo los paneles desplegables de la vista principal al iniciar la aplicación.

Sin embargo, el archivo `front` es el encargado de manejar dinámicamente el código HTML y gestionar la interacción con el usuario. Ha sido diseñado dividiéndolo en tres partes: una para las funciones de inicialización de cada vista, otra para la inicialización de los manejadores de eventos correspondientes a los botones de la interfaz (*listeners*) y una última compuesta por funciones auxiliares.

6.2.2.2 Controlador

El controlador se encarga de establecer el flujo de llamadas entre la capa de presentación o el módulo `View` y el resto de capas. Esta capa es vital ya que establece un control y una separación entre la lógica y las vistas. Se decidió tener un controlador para estructurar una lógica que permita la elaboración de nuevos desarrollos.

Separando de esta manera las capas se establecen unas reglas de desarrollo que hace la incorporación de nuevo código y funcionalidades más intuitivas, teniendo claro en qué módulo se debe desarrollar cada parte de la nueva funcionalidad. Además, es una ayuda para todo aquel que intente entender el código ya desarrollado.

6.2.2.3 Módulo Logic

Esta es la capa donde se realizan todas las operaciones con los datos y se desarrolla la lógica de negocio. De momento sólo contiene el archivo `algorithm` ya que sólo existe una funcionalidad, pero se contempla la incorporación de nuevos archivos al módulo según se vayan ampliando las funcionalidades de la aplicación.

En el archivo `algorithm` está implementado el algoritmo expuesto anteriormente en el capítulo 5. Este archivo tiene dependencias con el archivo api.js, que implementa el módulo `DAO`.

6.2.2.4 Módulo DAO

Este módulo sirve de soporte para la lógica (módulo `Logic`) en cuanto al *backend* se supone.

Proporciona funciones de acceso a la API de Yummly en el archivo `Api.js`. Concretamente dos funciones: `getAll`, que se llama tanto para obtener `HTopIng` como para la lista de recetas basándonos ya que está preparada para realizar una petición según los datos que le lleguen, y la función `getRecipe` que devuelve la receta del id que le llega.

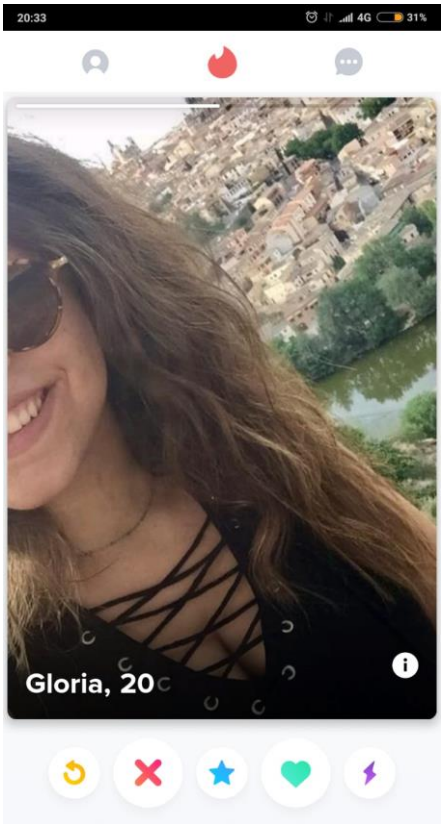
Por otro lado, el archivo `dao.js` establece la relación entre el DB Service y la aplicación siendo el punto de enlace para las peticiones

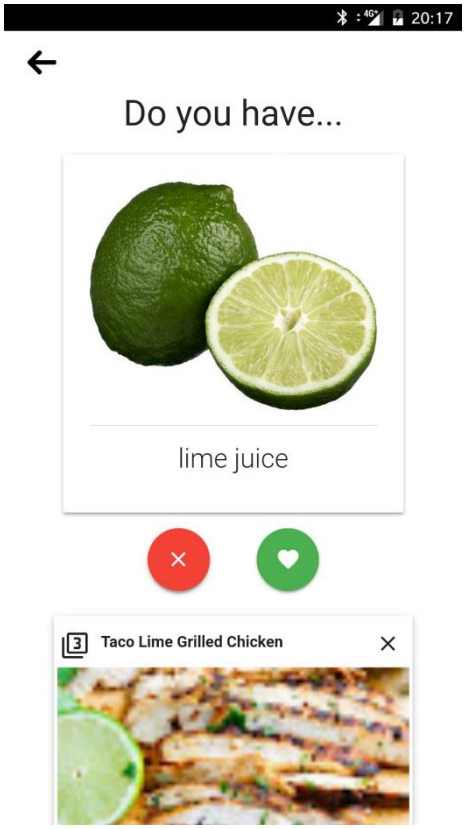
relacionadas con nuestra base de datos como la obtención de las fotos de un ingrediente o de los ingredientes pertenecientes a una categoría.

6.3 Diseño de las vistas

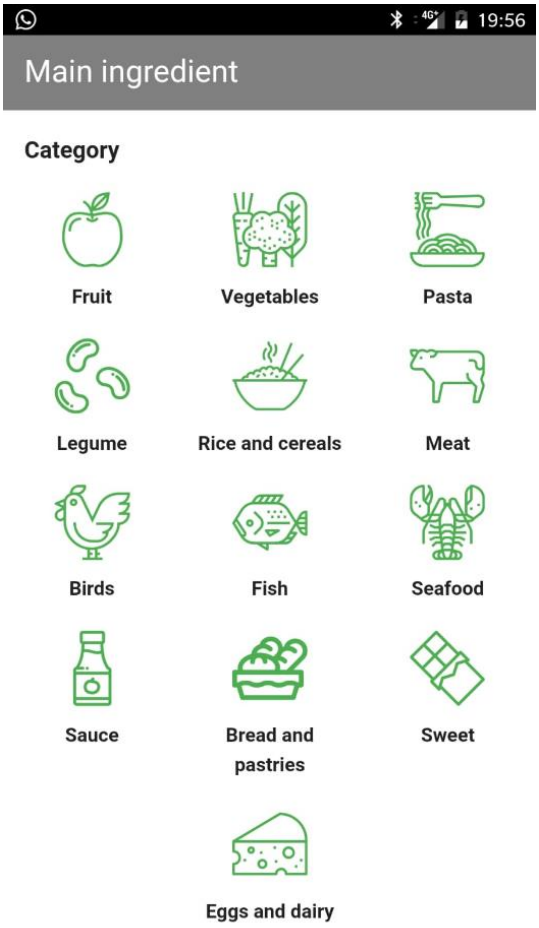
Además del requisito obvio de correcto funcionamiento de la aplicación, se prestó atención al diseño y a la usabilidad de la aplicación. Para ello, primero se realizó una extensa investigación estudiando aplicaciones móviles similares. Observamos la arquitectura de la información, funcionalidades e innovaciones que se realizaron en aquel momento. Algunos de los resultados son mostrados a continuación.

El proceso de selección de ingredientes por parte del usuario estuvo inspirado en la aplicación social de contactos *Tinder* [25] (Imagen de la izquierda), en la cual un usuario puede mostrar el interés o falta de interés en otro miembro de la red social, tal y como se muestra en la siguiente figura:

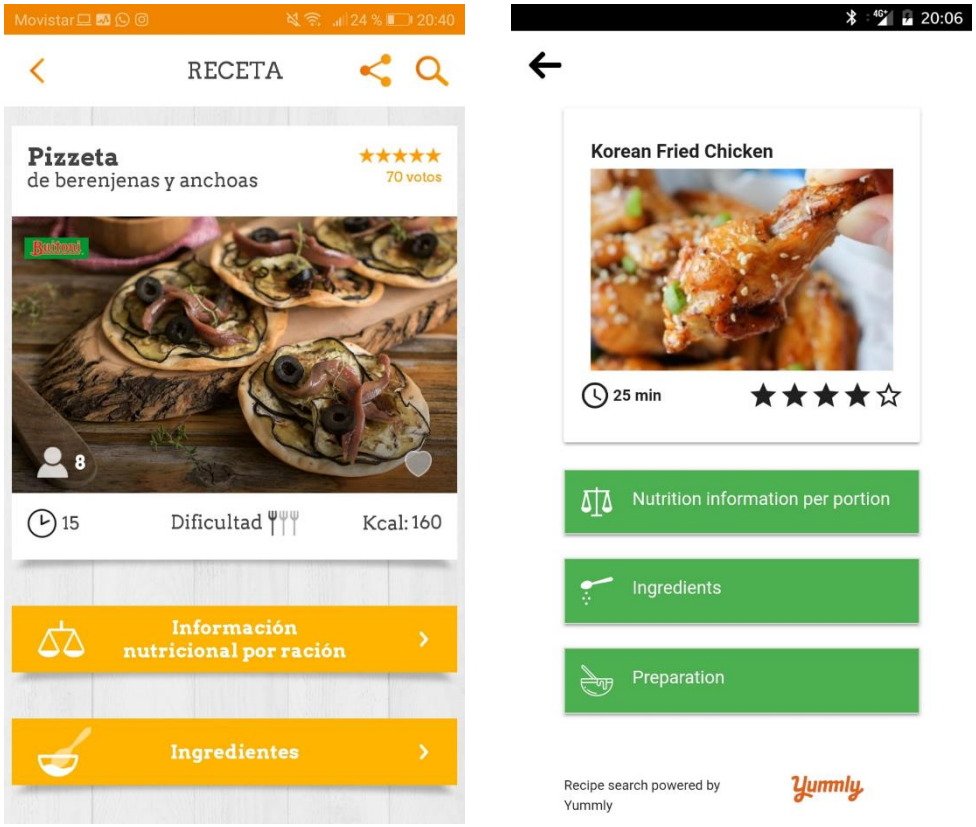




También nos inspiramos en la aplicación *Recetas Gratis* [26] para determinar el modo en el que se muestran las categorías para seleccionar el ingrediente principal.



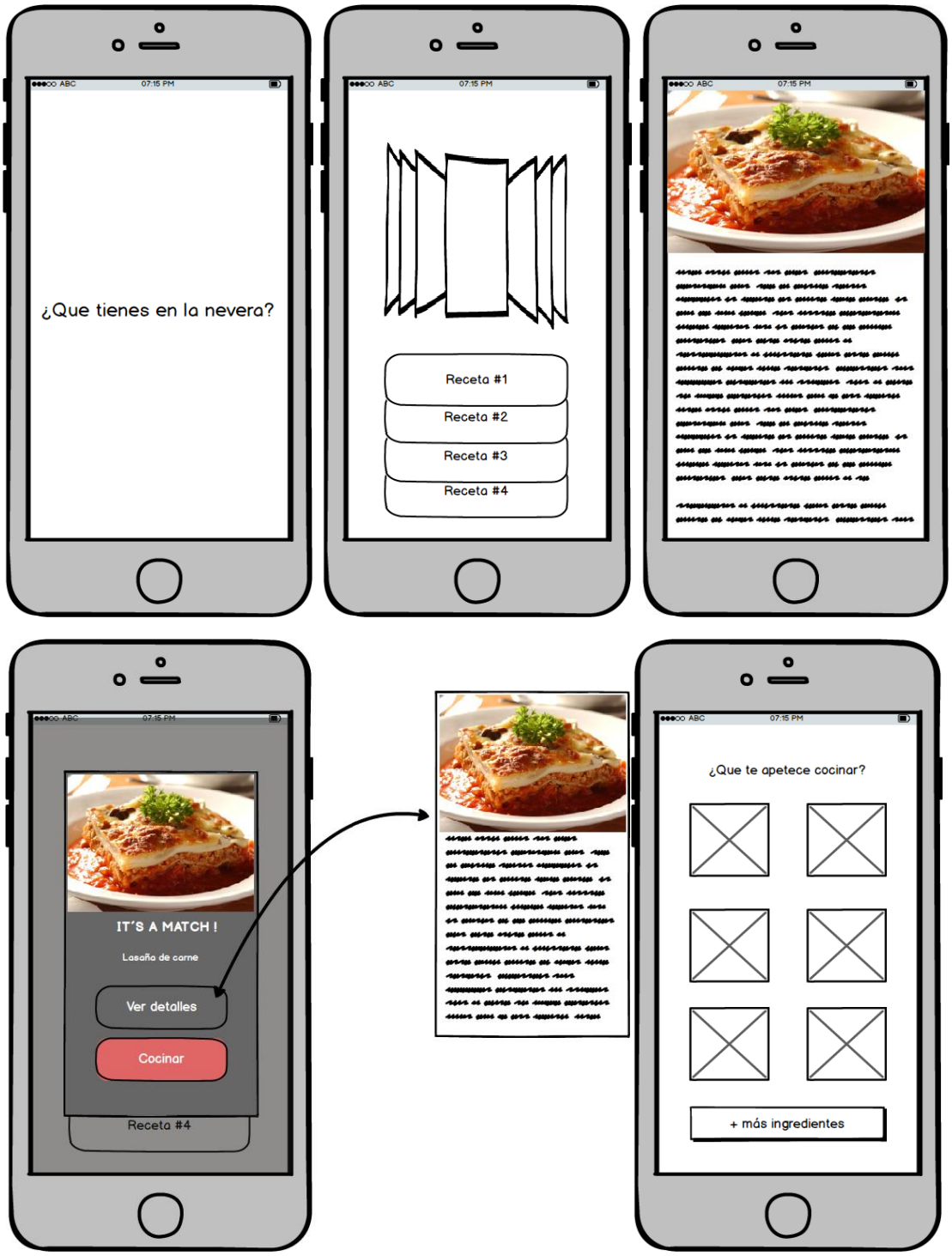
Con respecto a la visualización de la información de una receta, nos basamos en una aplicación (llamada *Nestlé Cocina* [27]) en particular en lo relativo a cómo se muestra un resumen de la receta, su puntuación, la duración, y en el hecho de que utiliza tarjetas desplegables para mostrar cada uno de los apartados de la información de la receta: información nutricional por ración, ingredientes, preparación, etc.



El siguiente proceso tras la investigación fue la fase de prototipado generando los *wireframes*, donde definimos primero el funcionamiento de la aplicación sin ningún tipo de diseño estableciendo una navegación sencilla, funcional para así detectar fácilmente los errores que pueda haber. Corregir los errores en esta etapa inicial nos ahorró las modificaciones posteriores en el diseño.

Se generaron los siguientes bocetos con la herramienta Balsamiq [28], utilizada en la asignatura “Desarrollo de sistemas interactivos” del grado en ingeniería informática.

Los *mockups* generados fueron los que se muestran en la siguiente figura:



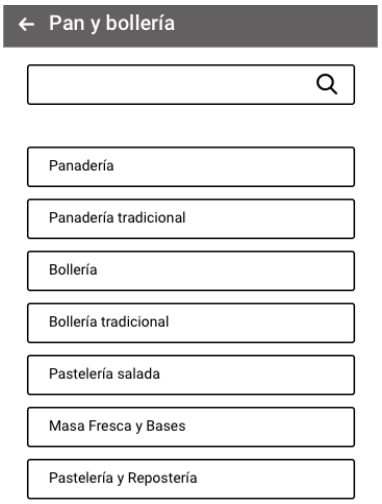
Una vez que los requisitos de usabilidad y la funcionalidad fueron cubiertos, pudimos comenzar a concretar el diseño visual de cada página individual.

En él, el color verde se convierte en el color corporativo de la aplicación. No quisimos una aplicación demasiado recargada de distintas tonalidades y colores, por lo que los elementos a destacar en esta son mostrados únicamente en color verde. Entre este tipo de elementos a destacar tenemos las recetas disponibles (al disponer el usuario de todos sus ingredientes), contenido importante de las recetas, etc.

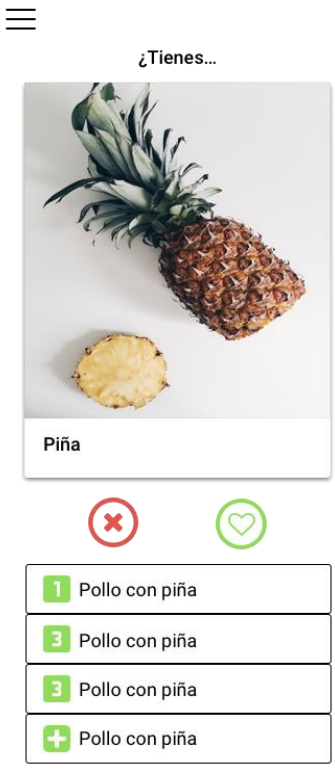
Se generaron unos bocetos más visuales con la herramienta Sketch que posteriormente serían modificaciones por decisión del equipo o limitaciones con el código de la aplicación.

A continuación, se muestra la primera vista mostrada por la aplicación, en la que el usuario puede elegir la categoría correspondiente del ingrediente que quiere utilizar como principal.

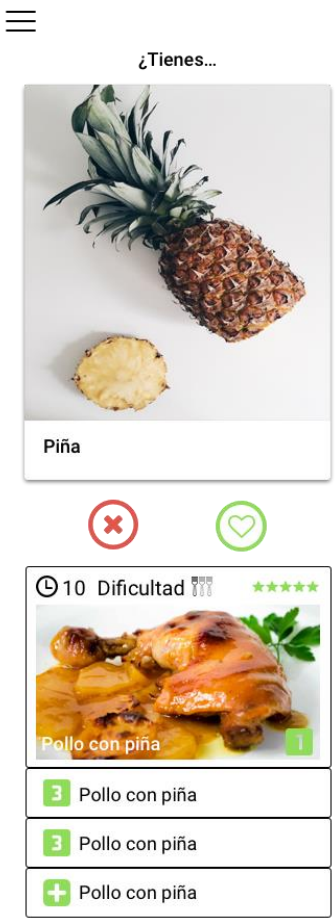
En la siguiente vista el usuario puede navegar por las distintas categorías de ingredientes para elegir el ingrediente principal de la receta.



Una vez que el usuario ha seleccionado el ingrediente que quiere utilizar como principal, la siguiente vista se encarga de mostrar una sección para preguntar al usuario por el ingrediente mostrado y debajo otra sección para mostrar la lista de posibles recetas.



Si el usuario pulsa una receta, se mostraría un pequeño resumen donde se puede ver una imagen de la receta, su puntuación, duración de la preparación y el número de ingredientes restantes para completarla.



Por último, cuando el usuario pulsa la receta deseada, se muestra la vista de la receta donde el usuario podrá consultar la información de la receta.

Las vistas finales de la aplicación (es decir, los documentos HTML y las hojas de estilo CSS) reflejan de manera bastante precisa este diseño inicial, salvo posteriores incorporaciones y cambios debidos a la propuesta de ideas nuevas durante la fase de implementación, limitaciones de Materialize, y falta de tiempo al incluir algunos mecanismos de interactividad propuestos inicialmente (por ejemplo, el deslizamiento de elementos mediante la pantalla táctil del dispositivo móvil).

7. Conclusiones y trabajo futuro

En este capítulo vamos a revisar los objetivos definidos al comienzo del proyecto valorando su grado de consecución. Adicionalmente vamos a proponer posibles mejoras del proyecto para el futuro.

7.1 Revisión de objetivos

Se han completado satisfactoriamente todos los objetivos planteados:

- Desarrollar el algoritmo de búsqueda

Este objetivo se ha conseguido satisfactoriamente obteniendo una selección inicial de recetas mediante una sola petición a la API. Solo en el caso excepcional de que queden pocas recetas disponibles con los ingredientes seleccionados y el usuario rechace suficientes ingredientes como para descartar la mayoría de ellas, el algoritmo necesita hacer más peticiones a la API para conseguir más información.

- Seleccionar una base de datos de recetas

Se consiguió encontrar una API externa que cumplía los requisitos necesarios (Yummly) y conseguimos acceso gratuito mediante una cuenta para estudiantes y así poder desarrollar este proyecto.

- Diseño de la arquitectura

Se definió una arquitectura modular que permitió alcanzar los objetivos del proyecto. Fue necesario definir una base de datos propia para enriquecer el contenido de la aplicación y un servidor web para acceder a dicha base de datos desde la aplicación.

Además, la aplicación está preparada para todas las plataformas propuestas.

- Diseño de la interfaz de usuario

Se diseñó una interfaz interactiva, amigable y adaptable a distintos dispositivos con la cual el usuario elige un ingrediente principal y recibe las propuestas de recetas disponibles a la vez que se le pregunta por el siguiente ingrediente.

Tras terminar el proyecto hemos podido observar que el resultado es una aplicación operativa que cumple con los requisitos definidos inicialmente. Durante el proceso, ciertamente hubo varias complicaciones, sobre todo relacionadas con la integración con la API, pero supimos sobrellevarlas y sobre todo aprendimos a cambiar de enfoque para no continuar atascados. Por otro lado, hemos tenido que aprender a usar herramientas totalmente nuevas como RequireJS, Cordova, Spring Boot, etc. Hemos aprendido que una correcta evaluación de riesgos puede ser clave a la hora de planificar y desarrollar un proyecto, como es nuestro caso con la API.

Nuestro objetivo es sacar la aplicación al mercado en un futuro próximo, y, aunque estamos contentos con el resultado del proyecto y pensamos que es una base muy buena desde la que empezar, también somos conscientes que nos faltan ciertas funcionalidades para poder obtener una aplicación competitiva.

7.2 Trabajo futuro

Las futuras evoluciones de este proyecto deberían incluir algunas funcionalidades adicionales como:

- Gestión de usuarios

Para cada usuario se podrían recordar las consultas realizadas, aprender de su comportamiento para mejorar la búsqueda aplicando algoritmos de aprendizaje automático, añadir interacción con redes sociales, hacer grupos de usuarios con preferencias similares, guardar recetas favoritas, etc.

- Lista de la compra

Ampliando la gestión de usuarios se podría permitir que la aplicación sugiriera ciertos ingredientes en una lista de la compra que, de tenerlos en la nevera, se amplíe mucho más la cantidad de recetas que se pueden realizar.

- Personalización

Es posible que algunos usuarios tengan ciertas restricciones alimentarias, dietas o alimentos a evitar, por lo que podemos añadir filtros

para las recetas en caso de que el usuario sea alérgico o padezca intolerancia a algún alimento, sea vegetariano, etc.

- Mejoras en la interfaz de usuario

Se pueden añadir animaciones nuevas para adornar visualmente la aplicación como por ejemplo una transición de tipo desplazamiento lateral para rechazar recetas, ingredientes, etc.

- Publicar la aplicación en tiendas de aplicaciones

Durante todo el proceso de desarrollo del proyecto mucha gente nos ha pedido que al terminarlo les proporcionásemos una dirección desde donde descargarse la aplicación. Dado que nosotros ideamos inicialmente esta aplicación pensando en resolver un problema que solemos tener frecuentemente, consideramos un buen objetivo subirla a las principales tiendas de aplicaciones para que otras personas también puedan usarla.

8. Conclusions and future work

In this chapter we will review the objectives defined at the beginning of the project, evaluating their degree of achievement. In addition, we will propose possible improvements to the project for the future.

8.1 Review of objectives.

All the objectives set have been successfully completed:

- Develop the search algorithm

This objective has been successfully achieved by obtaining an initial selection of recipes through a single API request. Only in the exceptional case that there are few recipes available with the selected ingredients and the user rejects enough ingredients to discard most of them, the algorithm needs to make more requests to the API to get more information.

- Select a recipe database

We managed to find an external API that met the necessary requirements (Yummly) and got free access through a student account so we could develop this project.

- Design of the architecture

A modular architecture was defined to achieve the objectives of the project. It was necessary to define a database to enrich the content of the app and a web server to access the database from the app. In addition, the app is ready for all the proposed platforms.

- Design of the user interface

An interactive, user-friendly and adaptable interface was designed for different devices with which the user chooses a main ingredient and receives the available recipe proposals while being asked for the next ingredient.

After completing the project, we have observed that the result is an operational app that meets the requirements initially defined. During the process, there were certainly several complications, mainly related to API integration, but we were able to cope with them and we learned to change our approach so as not to get stuck. On the other hand we have had to learn to use totally new tools such as RequireJs, Cordova, Spring Boot, etc. We have learned that a correct risk analysis can be key when planning and developing a project, as is the case with the API.

The idea behind the app is to bring it to market in the near future, and although we are happy with the outcome of the project and we think it is a very good basis from which to start, we are also aware that we lack certain configurations to be a competitive app.

8.2 Future work

Future developments of this project should include some additional features such as:

- User management

For each user the app could remember the queries made, learn from their behaviour to improve your search by applying automatic learning algorithms, add interaction with social networks, make groups of users with similar preferences, save favourite recipes, etc.

- Shopping list

Expanding user management could allow the app to suggest certain ingredients on a shopping list that, if kept in the refrigerator, would further expand the number of recipes that can be made

- Personalization

It is possible that some users may have certain dietary restrictions, diets or foods to avoid, so we can add filters for recipes in case the user is allergic or intolerant to any food, is vegetarian, etc...

- Improvements in the user interface

New animations can be added to visually embellish the app, such as a side-scrolling transition to reject recipes, ingredients, etc.

- Publish the application in app stores

Throughout the project development process many people have asked us to provide them with an URL from which to download the app. Since we initially came up with this app with the idea of solving a problem that we often have, we consider it a good objective to upload it to major app stores so that other people can use it as well.

9. Apéndice: Aportaciones de los integrantes

En este capítulo explicamos de forma detallada la participación de cada uno de los miembros del grupo y sus responsabilidades dentro del proyecto.

9.1 Carlos Martínez Gutiérrez

Desde el principio mi tarea principal dentro del equipo de trabajo ha estado enfocada hacia el desarrollo del *backend* de la aplicación.

Durante la preparación del proyecto los tres miembros hicimos un trabajo bastante similar, ya que lo hicimos de forma conjunta para poder ponernos de acuerdo en las responsabilidades que tomaría cada uno, los roles de equipo que adaptaríamos, etc.

La preparación del proyecto consistió en varias tareas:

- Definición de la metodología de trabajo

Debíamos saber todos en todo momento en qué estaban trabajando los demás e intentar comprenderlo, de forma que los tres supiésemos cómo funciona la aplicación en todo su conjunto. Eso también implica que haya tareas realizadas por más de un integrante del grupo a la vez.

- Creación del repositorio

De esta tarea me encargué yo en su totalidad. Investigué sobre páginas para alojar proyectos y Github permitía crear repositorios privados con una cuenta universitaria que ya tenía, por lo que creé el repositorio e hice participantes a Javier, a Alejandro y a Manuel, nuestro director del proyecto.

- Definición de objetivos, organización del tablón de Trello y planificación del proyecto

Estas tareas fueron realizadas por los tres, ya que debíamos estar todos de acuerdo en todas las partes, sobre todo a la hora de elegir los objetivos al ser estos poco realistas.

Después de tener claras las tareas de planificación, comenzamos las de investigación. Yo me encargué de realizar una investigación exhaustiva de APIs de recetas, ya que al principio conocía de una pero nos dio problemas con los ingredientes. Tuve que investigar nuevas, hacer una recopilación de las que encontré con sus pros y sus contras y enseñar todas las encontradas al equipo para decidir cuál usar.

También ayudé a Alejandro con su investigación sobre las aplicaciones y webs similares e investigué junto con Javier todo lo que pudimos sobre Cordova: cómo funciona, quién lo usa, qué podemos hacer con ello, hicimos apps de prueba y las instalamos en nuestro teléfono.

Una vez tuvimos claro el alcance que teníamos con Cordova, nos pusimos a desarrollar el algoritmo. Empezamos unas primeras versiones probando llamadas a la API y separando la información para acabar creando una primera versión. Comprobamos que era necesaria una mayor optimización para poder ejecutarlo en un tiempo relativamente corto, tarea que fuimos avanzando Javier y yo.

Más tarde, junto con Alejandro también, conseguimos crear un algoritmo final que solo usase una petición a la API en cada iteración.

Mientras el algoritmo era probado y se comprobaba que no tuviese fallos me puse a crear la aplicación para el proyecto con Cordova, encargándome de la configuración del proyecto y creando una vista sencilla sobre la que se iría desarrollando en el futuro el resto de la aplicación.

Al mismo tiempo que Javier y Alejandro avanzaban con la aplicación, paralelamente instalé en un VPS todo lo necesario para levantar una base de datos y un servidor web para poder usarla.

Para ello empecé instalando la propia máquina y descargando los programas principales (java y MySQL) junto con sus dependencias. Creé un usuario llamado *recipeers* para poder acceder y que no tuviese permisos de `sudo`.

En la base de datos creé las tablas y contribuí a la inserción de ingredientes ayudado en gran parte por Alejandro. También creé otro

usuario *recipeers* con los permisos básicos para mayor seguridad, ya que sería el usuario utilizado por el servidor web.

Desarrollé una aplicación web con Spring Boot para servir de API y así la aplicación pudiese conectarse con la base de datos para conseguir la información extra que Yummly no podía proporcionarnos.

Al finalizar, realizamos tareas de *testing* para comprobar que todo funcionaba correctamente y descubrir fallos. También animamos a un número reducido de conocidos para que la probasen y nos comentasen errores y posibles mejoras.

9.2 Alejandro Montero Roldán

Mi puesto dentro del equipo de trabajo ha estado enfocado hacia el desarrollo del *frontend* de la aplicación, enfocado a la experiencia de usuario.

Al tener la oportunidad de trabajar con compañeros de la carrera, donde conocemos el trabajo de cada uno y sus habilidades, nuestro objetivo se convirtió en vivir la mejor experiencia creando una aplicación agradable y aprender de su desarrollo.

Hemos trabajado en común durante el desarrollo, intentando que todos los miembros entendiéramos la aplicación por completo. Aunque hubo tareas que fueron asignadas a cada miembro del equipo, todo el equipo estaba al corriente del estado de la aplicación.

El proceso de inicio fue un trabajo en grupo, se realizaron tareas de investigación, gestión y diseño que nos facilitó el trabajo al tener todas las ideas puestas en común.

Con el avance del proyecto se definieron tres tareas principales:

- Desarrollo del *backend*, capa de base de datos, integración de la API y DB Services. Estas tareas fueron asignadas a Carlos.
- Desarrollo de la lógica de negocio e integraciones con el resto de módulos, que fueron asignada a Javier.
- Desarrollo del diseño y *frontend* y todo lo relacionado con la experiencia de usuario, desde la investigación, diseño e

implementación de las interfaces hasta ayudar con la integración con el módulo `View`. Estas tareas me fueron asignadas a mí.

Nuestros primeros pasos fueron definir cómo íbamos a trabajar y la metodología de trabajo a seguir. Con Trello conseguimos planificar nuestro trabajo, donde subíamos las tareas nuevas con asignación a miembros del equipo. Gracias a esta herramienta todos sabíamos en qué fase se encontraba cada tarea (por hacer, desarrollo o terminada) y qué tareas le correspondían a cada uno.

Tuvimos muchas ideas que incluir en la aplicación, pero nos dimos cuenta que debíamos de centrarnos en tener la base de la aplicación, con una funcionalidad principal. Para ello definimos el alcance del proyecto, intentando crear el producto lo más sencillo posible y posteriormente mejorarlo con nuestras ideas.

Con los objetivos bien definidos, realizamos una planificación para llegar a los tiempos de entrega acordados con el tutor y conseguir el mejor trabajo posible en el menor tiempo posible para tener la oportunidad de incorporar nuevas funcionalidades.

Una vez definido el proyecto comenzó la fase de investigación. En esta fase me encargué de la investigación de aplicaciones de recetas ya en el mercado. Para cada una de ellas llevé a cabo un análisis de diseño, de su usabilidad, observaba las innovaciones que introducían y si era adecuada para nuestra aplicación, definiendo todas las posibilidades que permitía la aplicación y, sobre todo, los fallos que encontraba en la misma.

Por otro lado, decidí empezar a investigar sobre Sketch, la herramienta principal, utilizada en mi equipo de trabajo en el que me encuentro de prácticas, para el desarrollo de bocetos. Estoy realizando prácticas de empresa para una empresa multinacional y pertenezco al departamento de Experiencia de Usuario.

Tras investigar y familiarizarme con Sketch acabé por entender su funcionamiento y obtener los bocetos utilizado para el desarrollo de las vistas de la aplicación.

Durante este proceso realicé un análisis (*benchmark*) comparando los productos similares que existen en el mercado, junto con el criterio de

profesionales, a partir de lo cual obtuve todo lo necesario para empezar a crear los bocetos. Una vez terminados, fueron valorados por mi equipo de trabajo, donde sugirieron mejoras y detectaron fallos que solucioné posteriormente.

La siguiente tarea que decidimos como prioritaria era el diseño del algoritmo principal. Los primeros pasos fueron llevados a cabo por el equipo en conjunto. Más adelante tuve que dejar de participar temporalmente en esta tarea para encargarme del diseño de las interfaces, de modo que los dos compañeros restantes siguieron diseñando el algoritmo principal, hasta la última versión del mismo, en el que participé con el fin de disponer de una versión final válida.

Por motivos de planificación y fechas de entrega tuve que empezar el proceso de implementación de las vistas de la aplicación. Para ello comencé con una primera fase de diseño en la que estudiaba las posibilidades de implementación con los bocetos que teníamos de la aplicación. Después siguió una segunda fase de implementación, donde generaba el código necesario para la implementación de las vistas.

Durante la fase de implementación tuve que intervenir en la parte de *frontend* de la aplicación para generar funcionalidades que necesitaba en aquel momento. Mi compañero Javier fue la persona con mayor responsabilidad en esta parte, aunque yo le serví de apoyo y desarrolle varias funciones relacionadas con el control dinámico de los datos de las vistas.

Más adelante, mi compañero Carlos y yo nos encargamos de generar contenido para la base de datos. Mediante el uso de la línea de comandos de la base de datos me encargué de categorizar los ingredientes que tienen la aplicación, añadir fotos y crear ingredientes que faltaban en la base de datos.

Por último, cuando ya teníamos nuestra aplicación lista, realicé una fase de testeo para revisar funcionalidades, navegación, control de errores, mejora de detalles, etc. Para ello conté con la ayuda de amigos y familiares que fueron sometidos a unas pruebas de usuarios con la aplicación, donde mostraba los resultados obtenidos a mis compañeros y se realizaba una toma de decisiones para posibles cambios.

9.3 Javier Vicente Cano

Desde el inicio del proyecto nuestra metodología siempre fue intentar realizar los avances en común, de manera que todo el equipo entendiese la aplicación por completo. Ciertamente, hubo repartos de tareas que se realizaron de manera casi individual, pero todo el equipo estaba al corriente del estado de la aplicación y entendía los últimos avances.

Sobre todo al inicio del proyecto, se trabajó de manera más grupal, ya que era importante poner en común las tareas propias de gestión, investigación y diseño.

Según se avanzó en el proyecto las tareas se fueron repartiendo, dividiéndose de manera genérica en tres tareas principales. Por un lado, el *backend*, que incluye la capa de base de datos, integración de la API y DB Services, se le adjudicó a Carlos. Alejandro se encargó de todo lo relacionado con la experiencia de usuario, desde la investigación, diseño e implementación de las interfaces hasta ayudar con la integración con el módulo View. Yo, por otro lado, me encargué de la lógica de negocio y las integraciones con el resto de los módulos.

Al principio del proyecto decidimos que era necesario emplear cierto tiempo en la preparación del proyecto.

Primero definimos cómo íbamos a trabajar, es decir, la metodología de trabajo. Se decidió utilizar una herramienta de gestión de tareas llamada Trello en la que cada semana subíamos las tareas nuevas y se asignaban a miembros del equipo con fechas límite. También dividimos las tareas añadiéndolas etiquetas distintas según fuese de gestión, desarrollo, investigación, etc. Gracias a esta herramienta todos sabíamos en qué fase se encontraba cada tarea (por hacer, en desarrollo o terminada) y qué tareas le correspondían a cada uno.

A continuación definimos el alcance del proyecto. Esto fue vital ya que teníamos muchas ideas que queríamos incluir en la aplicación, todas ellas expuestas en el trabajo futuro. Pero al final decidimos que lo mejor era desarrollar una buena base y desarrollar sólo la funcionalidad principal,

debido a todo el proceso de investigación y diseño que teníamos por delante.

Una vez definidos los objetivos, quedaba planificar la realización de los mismos. Se intentó realizar una planificación que tuviese en cuenta los distintos riesgos de cada fase por si surgían problemas. En caso de finalizar todo el proyecto antes de tiempo se tendría en cuenta la posibilidad de incluir una nueva funcionalidad. No obstante, decidimos no usar ninguna herramienta de planificación de proyectos, al poder incluir fechas límite en las tareas de Trello.

Una vez definida la planificación del proyecto, comenzó la fase de investigación. En esta fase se me encargó la investigación de aplicaciones de recetas que se encontrasen ya en el mercado. Para cada una de ellas llevé a cabo un análisis funcional, definiendo todas las posibilidades que proporcionaba la aplicación y, sobre todo, los fallos que encontraba. El objetivo de esta investigación era obtener una lista con todo aquello que queríamos en nuestra aplicación, ya fuese porque nos gustaba de otra o porque lo echábamos en falta.

Por otro lado, Carlos y yo decidimos empezar a investigar sobre córdova, ya que aunque sabíamos que era capaz de transformar una aplicación web en una android o Ios, debíamos aprender a utilizarla. Para ello antes me instalé el Android Studio con el fin de utilizarlo para ejecutar la aplicación Android resultante de Cordova. Tras investigar y probar acabé por entender su funcionamiento y obtener una aplicación sencilla desarrollada en html y javascript transformada en una aplicación Android.

La siguiente tarea que decidimos como prioritaria era el diseño del algoritmo principal. Los primeros pasos los fuimos llevando en conjunto, y yo me fui encargando de transcribir, no desarrollar, las versiones iniciales del algoritmo.

Según fuimos avanzando con el algoritmo nos dimos cuenta que, debido a las limitaciones de la API, la importancia de realizar un algoritmo eficiente que se adaptase a dichas limitaciones. Es por eso que la elaboración del diseño definitivo del algoritmo nos llevó más tiempo del esperado.

Una vez diseñado la versión final del algoritmo, y estando transcrita. Desarrolle un diagrama para poder visualizar las variables necesarias durante la ejecución y poder ver el algoritmo dividido en funciones simples que se pudiesen empezar a desarrollar.

Todo el equipo comenzó en desarrollo del algoritmo con las funciones individuales que serían llamadas desde el main del algoritmo. Más tarde, tras tener una versión reducida y simple del algoritmo mis compañeros se centraron en sus tareas y a mi se me encargó terminar de desarrollarlo por completo.

Una vez desarrollado, creamos el proyecto definitivo de Cordova y utilizamos RequireJS para incluir todos los módulos del proyecto: `View`, `Controlador`, `Logic`, `DAO`. Yo me encargué de la integración de los distintos módulos en el proyecto y las conexiones entre los mismos como ya está explicado en el capítulo 6.

El desarrollo del controlador fue íntegro por mi parte, y el esquema general del `front` igual, aunque Alejandro intervino a la hora de integrar la parte de control dinámica del *html*. Todo el módulo `DAO` fue desarrollado por Carlos y la integración entre el módulo `Logic` y el `DAO` y el `controlador` y el `DAO` quedó a mi cargo.

Una vez desarrollada la aplicación completa y estando en pleno funcionamiento comenzó un periodo de pruebas o testing. Durante este periodo la idea principal era intentar conseguir que la aplicación fallase de cualquier manera posible. Aunque no surgieron errores graves, si encontramos ciertos fallos de visualización o de duplicidad en la base de datos, por ejemplo. Además, durante esta fase les dejamos probar la aplicación a diferentes usuarios ajenos al desarrollo de la misma para ver si existían errores que no consiguiésemos descubrir.

Por último, el desarrollo de la memoria fue dividido en partes según lo desarrollado por cada uno, de manera que cada uno escribiese sobre aquello que realmente controlaba mejor.

Referencias:

[1] «Edamam.com» [En línea]. Available: https://www.edamam.com/. [Último acceso: 2018].

[2] Yummly, «Yummly.com» [En línea]. Available: https://www.yummly.com/. [Último acceso: 2018].

[3] Food2Fork, «Food2Fork» [En línea]. Available: https://food2fork.com. [Último acceso: 2018].

[4] Big Oven, «Bigoven.com» [En línea]. Available: https://www.bigoven.com/. [Último acceso: 2018].

[5] Apache Cordova, «Cordova.apache.org» [En línea]. Available: https://cordova.apache.org/. [Último acceso: 2018].

[6] «Cordova.apache.org» [En línea]. Available: https://cordova.apache.org/docs/en/8.x/guide/overview/index.html. [Último acceso: 2018].

[7] «W3.org» [En línea]. Available: https://www.w3.org/TR/html5/. [Último acceso: 2018].

[8] «W3.org» [En línea]. Available: https://www.w3.org/Style/CSS/. [Último acceso: 2018].

[9] «Materializecss.com» [En línea]. Available: https://materializecss.com. [Último acceso: 2018].

[10] «JavaScript.com» [En línea]. Available: https://www.javascript.com/. [Último acceso: 2018].

[11] «Node.js» 2018. [En línea]. Available: https://nodejs.org.

[12] «Requirejs.org» [En línea]. Available: http://requirejs.org/. [Último acceso: 2018].

[13] «Projects.spring.io» [En línea]. Available: https://projects.spring.io/spring-boot/. [Último acceso: 2018].

[14] «Mysql.com» [En línea]. Available: https://www.mysql.com/. [Último acceso: 2018].

[15] «GitHub» [En línea]. Available: https://github.com/features. [Último acceso: 2018].

[16] «Google.com» [En línea]. Available: https://www.google.com/intl/es_ALL/drive/. [Último acceso: 2018].

[17] «Trello.com» [En línea]. Available: https://trello.com/. [Último acceso: 2018].

[18] «Sublimetext.com» [En línea]. Available: https://www.sublimetext.com/. [Último acceso: 2018].

[19] «Atom» [En línea]. Available: https://atom.io/. [Último acceso: 2018].

[20] «Putty.org» [En línea]. Available: https://www.putty.org/. [Último acceso: 2018].

[21] «Sketch» [En línea]. Available: https://www.sketchapp.com/. [Último acceso: 2018].

[22] «Ovh.es» [En línea]. Available: https://www.ovh.es/vps/. [Último acceso: 2018].

[23] «Docs.spring.io» [En línea]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/mvc/method/annotation/AbstractJsonpResponseBodyAdvice.html. [Último acceso: 2018].

[24] «Docs.spring.io» [En línea]. Available: https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html. [Último acceso: 2018].

[25] «Tinder» [En línea]. Available: https://tinder.com/. [Último acceso: 2018].

[26] «app.recetasgratis.net» [En línea]. Available: https://app.recetasgratis.net/. [Último acceso: 2018].

[27] «Nestlé Cocina» [En línea]. Available: https://www.nestlecocina.es/app-nestle-cocina. [Último acceso: 2018].

[28] «Balsamiq.com» [En línea]. Available: https://balsamiq.com. [Último acceso: 2018].