

PROYECTO DE DESARROLLO DE  
APLICACIONES MULTIPLATAFORMA

**GuideFood**



**Pedro Molina Espinosa  
CURSO 2019/2020**

## INDICE

1. Resumen .....	3
2. Introducción .....	4
2.1. ¿Por qué Flutter?.....	4
2.2. ¿Qué es dart? .....	5
2.3. ¿Como funciona un servicio REST?.....	6
2.4. ¿Qué es y cómo utilizar un sistema de control de versiones (GitLab)?.....	6
2.5. Mongo.....	7
2.6. Firebase.....	8
3. Objetivos .....	9
3.1. Diseño .....	9
3.2. Funcionales.....	10
3.3. No funcionales .....	11
4. Hipótesis de Trabajo .....	11
4.1. Patrón de diseño .....	11
4.2. Tecnologías posibles.....	12
4.2.1. RESTDB.IO.....	12
4.2.2. Docker.....	13
4.2.3. Ideas desechadas.....	13
5. Material .....	14
5.1. Hardware .....	14
5.2. Software .....	16
6. Método.....	20
6.1. Backend inicial .....	20
6.2. Servidor de imágenes.....	22
6.3. Servidor de datos (Backend principal) .....	24
6.4. Frontend con flutter .....	28
6.4.1. Datos .....	28
6.4.2. La vista intro SPLASH .....	29
6.4.3. La vista del listado .....	31
6.4.4. La vista del selector .....	38
6.4.5. La vista del login .....	45
6.4.6. La vista de los favoritos y los valorados .....	46
6.4.7. La vista del detalle .....	50
6.4.8. El drawer .....	55
6.5. Debuggeando con flutter .....	56
6.6. Testing básico en modelos .....	58
6.7. Librerías .....	60
6.8. Utilización de recursos .....	61
7. Manual de uso .....	62
8. Bibliografía.....	71
6. Conclusiones .....	72

## 1. RESUMEN

Desarrollo de una aplicación móvil que emplea servicios de suministro de datos como RESTFUL con una api que gestiona con la autenticación, una lista de recetas gastronómicas favoritas marcadas por un usuario.

La aplicación cuenta con un sistema de vistas diseñadas con flutter para mostrar una lista de recetas que conseguiremos a través de una respuesta de nuestro servicio rest además de una vista que nos permita introducir aquellos ingredientes de los que disponemos y nos mostrará por pantalla una lista de recetas.

- Objetivos iniciales:

-Objetivos a nivel de desarrollo:

- Puesta a disposición del usuario de una lista de recetas con una explicación detallada de preparación e ingredientes a través de la interfaz de usuario.
- Sistema de añadido a favoritos con autenticación por mail.
- Creación de una api web para aglutinar nuestros datos.
- Sistema de búsqueda de recetas.
- Utilización de material design y cupertino para el desarrollo en android y iPhone.
- Es primordial emplear un sistema de control de versiones así que voy a utilizar gitlab.

-Objetivos nivel usuario:

- Poder encontrar recetas y pasos para realizarlas .
- Poder guardar una lista de sus recetas favoritas.
- Seleccionar ingredientes de una lista y aglutinar un conjunto de recetas posible en función de la cantidad de ingredientes.
- Disponer de una interfaz de usuario intuitiva y funcional.

## 2. INTRODUCCIÓN

El desarrollo de esta aplicación implica un sistema de diseño de interfaz de usuario que será lo que mostraremos a aquel que consuma datos de nuestra aplicación, además de una parte de modelo lógico que nos permita transformar los datos que pediremos a nuestro servicio rest, lo cual implica una serie de packages de diseño y de gestión de datos tales como las librerías convert de dart y el empleo del plugin past json as code para diseñar el modelo de datos.

La idea primaria es hacer un diseño usando librerías externas más allá de la versatilidad de diseño que nos permite flutter.

El uso del repositorio me permitirá desplegar un sistema de versiones para gestionar errores y estabilidad de mi aplicación.

### **2.1. ¿POR QUÉ FLUTTER?**

Google (la empresa desarrolladora de flutter) ha apostado fuerte con flutter rediseñando su infraestructura informática en torno a este para temas tan cruciales como su sistema de inversión con anuncios.

No solo voy a apoyar su uso citando al gigante informático, sino que además comentando algunas cualidades de diseño, flutter adelanta por la derecha en rendimiento a las aplicaciones desarrolladas en java a través de android studio suponiendo así un impacto en la comunidad de desarrolladores móviles, gracias a que su compilado a lenguaje nativo del dispositivo ya sea Android o iOS, le infiere una velocidad de rendimiento a tener en cuenta.



Otro tema digno de mención es lo anteriormente citado, sí, el desarrollo único de una aplicación vale para su compilación tanto en android como iOS, motivo por el cual es un sdk multiplataforma sobresaliente, con la única desventaja de que para su compilado necesitamos usar el xcode de apple del cual solo desarrolladores con mac pueden hacer uso.

Por último cabe destacar su desarrollo veloz y flexible que nos permite depurar en tiempo real una aplicación gracias a su función “hot reload” mediante la cual no tenemos que esperar decenas de segundos y hasta minutos para la instalación completa de nuestra aplicación para depurarla. Esta funcionalidad permite reinstalar rápidamente la aplicación, o recargarla en cuestión de milisegundos para apreciar los cambios, y no solo eso, sino que con solo guardar mientras está activo el debugging podemos apreciar dichos cambios.

Flutter utiliza widgets para disponer su entorno, los cuales pueden contener más widgets y cada tipo contiene ciertas funcionalidades que podemos ir alternando para conseguir los resultados que deseamos, y esto define intrínsecamente a flutter como un framework completamente modular.

Aquí un ejemplo de una vista:

```

1 class _SelectorPageState extends State<SelectorPage> {
2   final GlobalKey<ScaffoldState> _scaffoldKey = new GlobalKey<ScaffoldState>();
3
4   ApiProvider provider = new ApiProvider();
5   List<Ingrediente> listaPasada = new List<Ingrediente>();
6
7   @override
8   Widget build(BuildContext context) {
9     Size size = getMediaSize(context);
10    return Scaffold(
11      key: _scaffoldKey,
12      endDrawer: DrawerGuideFood(),
13      body: Stack(
14        children: <Widget>[
15          _body(size),
16          getAppBar(
17            context,
18            primaryColor.withOpacity(0),
19            size.width * 0.05,
20            _scaffoldKey,
21          ),
22        ],
23      ), // Stack
24    ); // Scaffold
25  }

```

## 2.2. ¿QUÉ ES DART?

Dart, originalmente conocido como Dash, es un lenguaje tipado similar a Java, es de código abierto y desarrollado y mantenido por Google.



El objetivo de Dart no es sustituir a JavaScript sino modernizar el desarrollo web, porque este fue su enfoque al principio, y puesto que su desarrollo lleva desde 2011 además de haber sido incluido en el desarrollo de flutter desde 2016/17, hay numerosos tutoriales y una documentación detallada para tener guías suficientes para emplearlo.

Ejemplo de Dart:

```

int fibonacci(int n) {
  if (n == 0 || n == 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

var result = fibonacci(20);

```

### 2.3. ¿CÓMO FUNCIONA UN SERVICIO REST?

Se trata de un sistema de petición/respuesta entre un cliente y un servidor y no manejan estados, esto quiere decir que no hay pasos intermedios que ralenticen o “burocratizan el proceso” así es más rápido. También se pueden utilizar cachés para mapear la estructura rest y agilizar las peticiones.

Las peticiones más importantes en un servicio rest son las peticiones http, las cuales se usan para interactuar con nuestro servicio pidiendo datos con GET, creando y añadiendo con POST, modificando o actualizando con PUT y eliminando con DELETE.

Este sistema siempre opera a través de direcciones web apuntando siempre a los recursos del servicio a través de la URI (dirección del dato en nuestro servicio rest):

<http://localhost:3000/alumnos/>

Estas APIs emplean JSON como lenguaje para ordenar los datos en nuestra base de datos rest.

### 2.4. ¿QUÉ ES Y CÓMO UTILIZAR UN SISTEMA DE CONTROL DE VERSIONES (GITLAB)?

La anarquía es algo repudiado por las compañías gestoras de información que antaño se quebraban la cabeza gestionando ingentes cantidades de documentos en archivos gigantescos que normalmente acababan cogiendo polvo y el hecho de consultarlos, si no llevaban un buen control de orden, podía suponer un dolor de cabeza importante, imagina en el caso de la programación.

Los controles de versiones permiten a los programadores hacer una gestión de esta información, ya sea en local o en la nube, con git podemos llevar un registro detallado de los cambios realizados por cada desarrollador que trabaja en un proyecto, permitiendo así la posibilidad de trabajar en equipo y de desarrollar en equipo un mismo código.

Los equipos de trabajo hoy empiezan a usar el sistema SCRUM que no es más que un sistema de reparto de tareas en tiempos y temáticas concretas que permiten a un equipo llevar un calendario de trabajo.

A pesar de que github es a día de hoy el sistema web de control de versiones más usado, no cuenta con la opción directa de este servicio, pero dispone de un gita, que le permite suplir esta falta.

Gitlab es un todo en uno, ya que cuenta con su tabla de SCRUM y permite un reparto y una eficiencia más adecuada al desarrollo.

Git nos permite almacenar nuestro código a través de:

Commit: son guardados de punto de control de nuestro proyecto que almacena nuestro código y que deben ir acompañados de un comentario para orientarnos en caso de necesitar consultarlos.

Branch: nos permite crear ramas a partir de las cuales desarrollar sin afectar a la rama estable o final (master), donde tenemos versiones en producción.

Checkout: nos permite saltar a una rama para desarrollarla a partir del punto en que se dejó o fuera clonada tras crearla a partir de una rama padre.

Rebase: nos permite clonar el código de otra rama a aquella en la que estamos posicionados.

Fetch: nos permite comprobar desde el repositorio local, si en el repositorio global se han realizado cambios y si es así, descargarlos para conocer estado de posibles nuevas ramas o nuevo código implementado por un compañero, lo cual nos permite hacer un merge.

Merge: utilidad que nos permite fusionar nuestro código y permitirnos discriminar entre líneas, que queremos añadir a una versión final, normalmente entre dos versiones diferentes en una misma rama, o dos versiones diferentes entre rama hija y rama padre, para añadir el desarrollo a esta última.



## 2.5. MONGO

Mongo es una base de datos no relacional y NoSQL orientada a documentos que utiliza BSON, un lenguaje binario de JSON.

Mongo es un tipo de base de datos excepcional para el desarrollo de aplicaciones con CRUD donde no necesitamos definir un esquema de datos concreto.

Es fácil de usar y permite un gran nivel de escalabilidad gracias a su función de sharding (fragmentación) que permite distribuir la base de datos en diferentes servidores añadiendo una shardkey para identificar la masa de datos que contiene.

Además se disponen de drivers para ser utilizada en múltiples lenguajes de programación, lo que la hace bastante atractiva.

El proyecto es de código abierto por lo que se puede modificar o crear drivers adaptados a los requisitos de un proyecto.



{ name: mongo, type: DB }

## 2.6. FIREBASE

Firebase es una plataforma de Google INC, que permite alojar bases de datos no relacionales en tiempo real además de una enorme serie de funcionalidades de monitorización, analíticas y despliegue en lo que a seguridad, autenticación, almacenamiento y usuarios respecta, incluidos permisos de acceso y restricciones que podrían suponer un dolor de cabeza implementar con cualquier lenguaje.

La fiabilidad en uno de sus puntos fuertes permitiendo encontrar un punto intermedio entre seguridad y rendimiento gracias al uso de tokens para identificación y uso de apis integradas en el servicio.



Existe un plan gratuito que permite disfrutar de las funcionalidades básicas y usar un servicio notable en proyectos escolares como este y en otros básicos de compañías que con un bajo precio pueden acceder a funcionalidades más avanzadas como conexiones de usuarios ilimitados, mayor cantidad de almacenamiento y número de proyectos. Es una plataforma idónea para desarrollo de aplicaciones móviles permitiendo una integración sencilla y bien documentada tanto para android como para ios además incluye la funcionalidad web para ser utilizada por servicios web tales como angular ionic react, etc.

Gracias al servicio de firebase he montado el servicio de autenticación con google.

Aquí podemos ver como el servicio está implementado y es funcional ya que ha cogido la cuenta que he utilizado para hacer login.

Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario
pedromolesp@gmail.com	Google	5 dic. 2019	14 dic. 2019	cV1mKc1sapcwXE2UBNJWfz0rW...

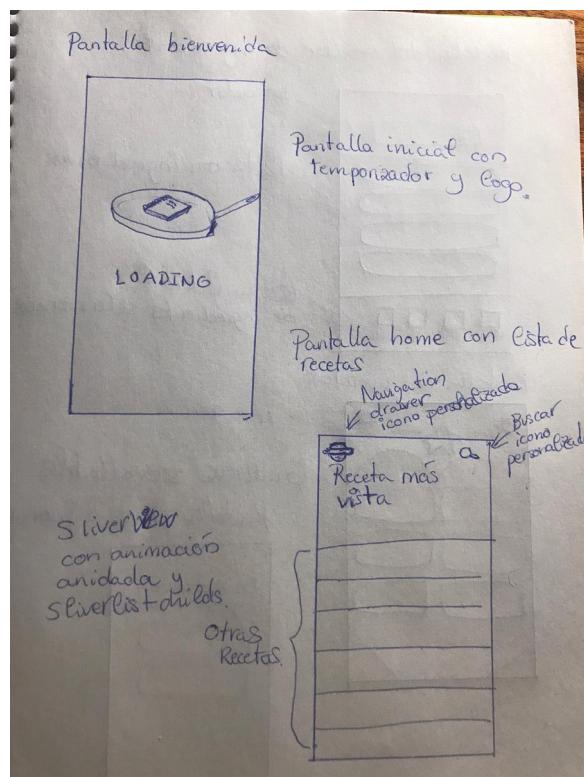
Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario
pedromolesp@gmail.com	Google	5 dic. 2019	14 dic. 2019	cV1mKc1sapcwXE2UBNJWfz0rW...

### 3. OBJETIVOS

#### 3.1. DISEÑO

Las primeras vistas de la aplicación fueron diseñadas dibujándolas en una libreta para ir estableciendo un patrón de modelos y vistas.

Estas son las vistas diseñadas al inicio:



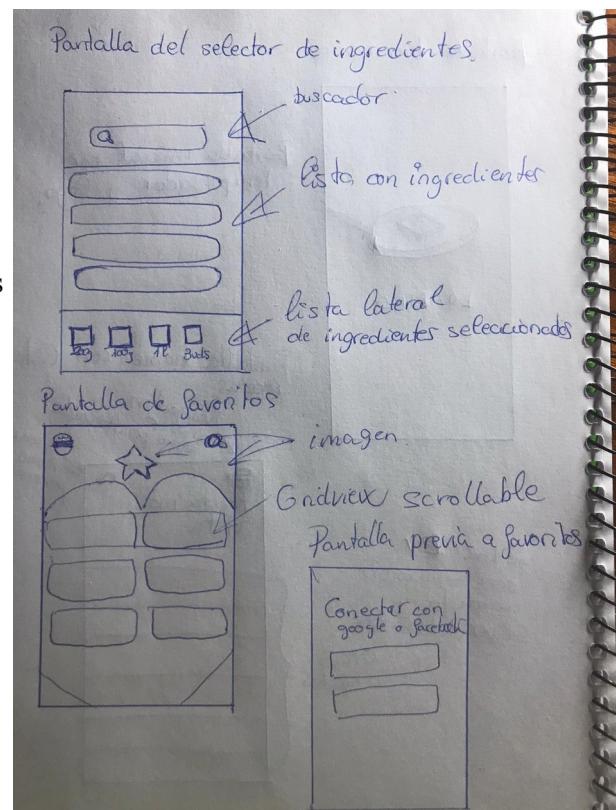
En la pantalla primaria la idea inicial es mostrar una animación con una imagen o pantalla de bienvenida, también tengo la idea de usar un alert en una tarjeta para hacerlo más elegante.

Esto dependerá del tiempo del que disponga y según se vayan desarrollando las partes funcionales de la aplicación, se irá implementando un mejor diseño o uno más básico para centrarse en la lógica.

Otra vista primaria mostrará una lista de recetas, con una principal que será la mejor calificada.

Las siguientes pantallas mostrarán un selector de ingredientes que se amontonarán en un contenedor y que permitirá al usuario hacer una lista de recetas que los contengan

Al final encontramos también una vista dedicada a añadir recetas favoritas y almacenarlas en un usuario logeado a través de google.



Algunos de los componentes que voy a usar en las vistas en principio:

- SliverAppBar Es un AppBar con animación de fundido con la parte superior.
- NavigationDrawer Despliega un menú lateral.
- SliverListChilds Permite una inserción de widgets con animación predefinida.
- ListView Setea una lista de elementos (Widgets).
- ListTiles Es un elemento de las listas que puede contener otro widget.
- Containers Similares a los divs de html, contienen otros widgets.
- Padding Espacios entre elementos.
- Margin Espacios de contenedor.
- Row Filas.

Esta no es una estructura definitiva, así que irá mutando a medida que vaya implementando las utilidades necesarias.

En flutter es necesaria la notación en el archivo guía que recoge dependencias y utilidades varias como imágenes, fuentes, sonidos, etc.

Hay que registrarlas para que sean visibles por la aplicación; este archivo se llama pubspec.yaml, se trata de un fichero que recoge los datos de forma estructural por lo que hay que ser muy cuidadoso en lo que a espacios y saltos de linea se refiere.

### 3.2. FUNCIONALES

- Desarrollo de un backend que permita obtener recetas e ingredientes de estas por http
- Una lista de recetas clickable que te lleve al detalle de la misma.
- Un detalle de receta que permita valorar y visualizar el nombre, dificultad, preparación por pasos, imagen de receta y lista de ingredientes.
- Una pantalla que te permite seleccionar los ingredientes de una lista para comprobar las recetas que contienen tales ingredientes.
- Disposición de una pantalla que permita logearse con google para acceder y almacenar una lista de recetas favoritas.
- Montar un navigation drawer que permita navegar entre las diferentes pantallas.
- Que la aplicación sea responsive.

### 3.3 . NO FUNCIONALES

- Aprender flutter
- Mejorar mis habilidades en el uso de git y sistemas de control de versiones.
- Aprender a programar siguiendo el método del código limpio, codificando de manera clara y concisa.
- Aprender del uso de packages de la comunidad.
- Aprender a leer documentación.
- Diseñar vistas bonitas y funcionales.
- Aprender a usar bases de datos no relacionales.
- Conocer y estudiar docker y como tratar de montar mi backend en docker para servirlo en una web host.

## 4. HIPÓTESIS DE TRABAJO

A continuación se describen las distintas alternativas para implementar la funcionalidad descrita anteriormente.

### 4.1 PATRÓN DE DISEÑO

No existe en una forma establecida para el desarrollo estructural de una aplicación, puede utilizarse el modelo vista presentador para distribuir el código y las carpetas de utilidades y librerías pero en flutter a veces es inevitable mezclar lógica y vista por la utilidad de algunos métodos, como estados, eventos con triggers en widgets, etc.

Mi estructura va a basarse en algo similar al modelo vista controlador que utilicé durante mis estudios en DAM.

Esta es la estructura de directorios que pretendo integrar:

 controllers	Subiendo nueva estructura del proyecto	1 hour ago
 models	Subiendo nueva estructura del proyecto	1 hour ago
 routes	Subiendo nueva estructura del proyecto	1 hour ago
 styles	Subiendo nueva estructura del proyecto	1 hour ago
 views	Subiendo nueva estructura del proyecto	1 hour ago

En controladores la idea es meter archivos tales como screen controllers para controlar tamaños de pantalla y pedir al dispositivo esta información y, trabajando con porcentajes, hacer un diseño responsivo de la aplicación según los diferentes tamaños de pantallas que hay en el mercado, incluso tablets.

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

void setScreensControls() {
  SystemChrome.setEnabledSystemUIOverlays([]);
  SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp,
    DeviceOrientation.portraitDown,
  ]);
}

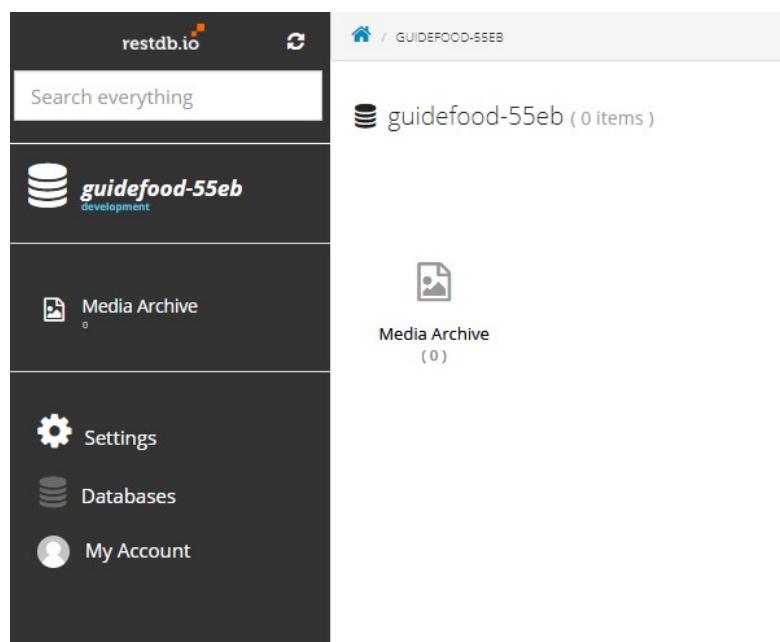
Size getMediaSize(BuildContext context) {
  return MediaQuery.of(context).size;
}

EdgeInsets getMediaPadding(BuildContext context) {
  return MediaQuery.of(context).padding;
}
```

## 4.2. TECNOLOGÍAS POSIBLES

### 4.2.1. RESTDB.IO

He buscado entre varios servidores rest gratuitos que permitan alojar un sistema de servidor básico gratuito y me he decantado por utilizar <https://restdb.io/> .



Finalmente fue descartado por falta de tiempo.

#### 4.2.2. DOCKER

La idea primaria es dockerizar mi backend y montar la api con sus endpoints en el repo anterior de manera que pueda servirme de la misma desde cualquier parte sin necesidad de lanzar el servicio en local con json-server.

También he pensado en dockerizar mongo para tener ambos servicios disponibles sin tener que lanzar el servicio cuando quiera utilizarlo y tenerlo activo.

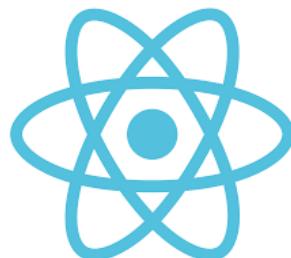
He comenzado un investigación para el uso de docker compose up y conseguir lanzar todos los servicios a la vez.

#### 4.2.3. IDEAS DESECHADAS

**Angular:** Una de las primeras ideas sobre tecnologías a utilizar era Angular, ya que habíamos trabajado con ella recientemente en clase y era muy potente además de flexible, porque permitía empaquetar también para móvil. Finalmente cambié de idea respecto a esto porque en la empresa íbamos a trabajar con flutter y preferí cambiar de idea porque sería más provechoso para la empresa, ya que aprendería más rápido.



**React:** React también se me pasó por la mente cuando estuve barajando las tecnologías que utilizar, ya que al igual que angular permite una portabilidad a otras plataformas bastante satisfactoria además de ser muy potente y permitir depurar instantáneamente. Fue rechazada finalmente por los mismos motivos que la anterior.



**Android:** Era una fuerte candidata a las tecnologías que usaría, ya que hemos trabajado en clase con android y para mí ha sido una experiencia muy agradable, por lo visual, que a priori es lo que queremos nada más empezar en esto de la programación.

Finalmente fue desechada ya que al entrar la alternativa de Flutter que era desarrollo móvil quise probar algo nuevo.



## 5. MATERIAL

### 5.1. HARDWARE

#### -----Al inicio del proyecto-----

General:

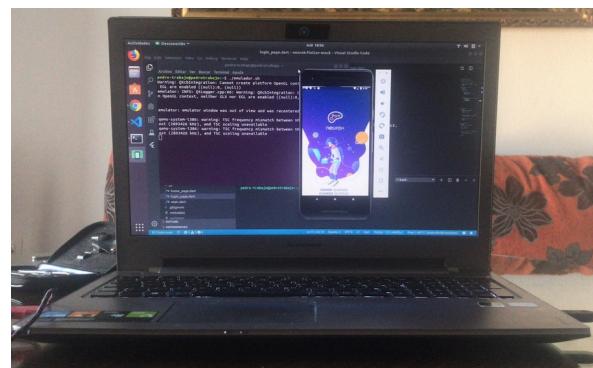
Notebook Lenovo IdeaPad Z500 64 bits i7 quinta generación. Con lubuntu.

Tarjeta de video:

VGA compatible controller  
product: GF108M  
[GeForce GT 635M] NVIDIA

Disco duro:

ATA Disk ST1000LM024 HN-M Seagate  
931GiB (1TB)



(Este es el portátil que estoy usando para desarrollar el proyecto al inicio del mismo aunque estoy esperando recibir otro por parte de la empresa de prácticas con unas mejores características.)

También me serví de mi equipo de sobremesa para el desarrollo:

General:

Sobremesa montado por piezas Ryzen 5 2200  
with vega graphics

Tarjeta de video:

XFX GTX 435 2GB

Disco duro: EVO 500GB

Pantalla: AOC 22b1h

Fuente de alimentación EVGA 600W

Memoria: 8GB



-----Al final del proyecto-----

AspGems compró un equipo portátil que es el que he utilizado para el desarrollo de la última parte del proyecto.

General:

Dell latitude 3400 Intel® Core™ i5-8265U CPU @ 1.60GHz × 8 8<sup>a</sup> generación.  
Ubuntu 18.04.

Tarjeta de video:

intel integrated

Disco duro: 120GB

Memoria: 8GB



## 5.2. SOFTWARE

Voy a enumerar los programas utilizados a efectos de las webs, ya que están listadas en la bibliografía.

- **Visual Studio Code:** Ha sido el IDE que he utilizado durante todo el desarrollo del proyecto. Es un entorno muy liviano comparado con Android Studio que me ha permitido hacer debugging y desarrollar código de una manera muy cómoda.



Imagen de la interfaz de Visual Studio Code mostrando el archivo 'Ingredient.dart' con código Dart. El código define una clase 'Ingrediente' que implementa 'Equatable'. El constructor recibe un Map<String, dynamic> json y asigna sus valores a los campos id, nombre, tipo, imagen, medida y cantidad. Los campos son requeridos y tienen tipos String, String, String, String y int respectivamente. El métodofromJsonMap hace lo mismo para crear un objeto Ingrediente a partir de un mapa JSON.

```

class Ingrediente extends Equatable {
  String nombre;
  String tipo;
  String imagen;
  String medida;
  int cantidad;

  Ingrediente({
    @required this.id,
    @required this.nombre,
    @required this.tipo,
    @required this.imagen,
    @required this.medida,
    this.cantidad
  }) : super();
}

Ingrediente.fromJsonMap(Map<String, dynamic> json) {
  id = json['id'];
  nombre = json['nombre'];
  tipo = json['tipo'];
  imagen = json['imagen'];
  cantidad = json['cantidad'];
  medida = json['medida'];
}

```

- **Android Studio:** Lo he utilizado al principio del desarrollo pero durante un periodo muy corto de tiempo, es bastante cómodo ya que entremezcla muy bien la emulación y creación de dispositivos para depuración.



Imagen de la interfaz de Android Studio mostrando el archivo 'main.dart' con código Dart. El código define una clase 'MyApp' que hereda de 'StatelessWidget'. El método 'build' establece el tema Material Theme y la ruta inicial 'splash'. La función 'main' ejecuta la aplicación.

```

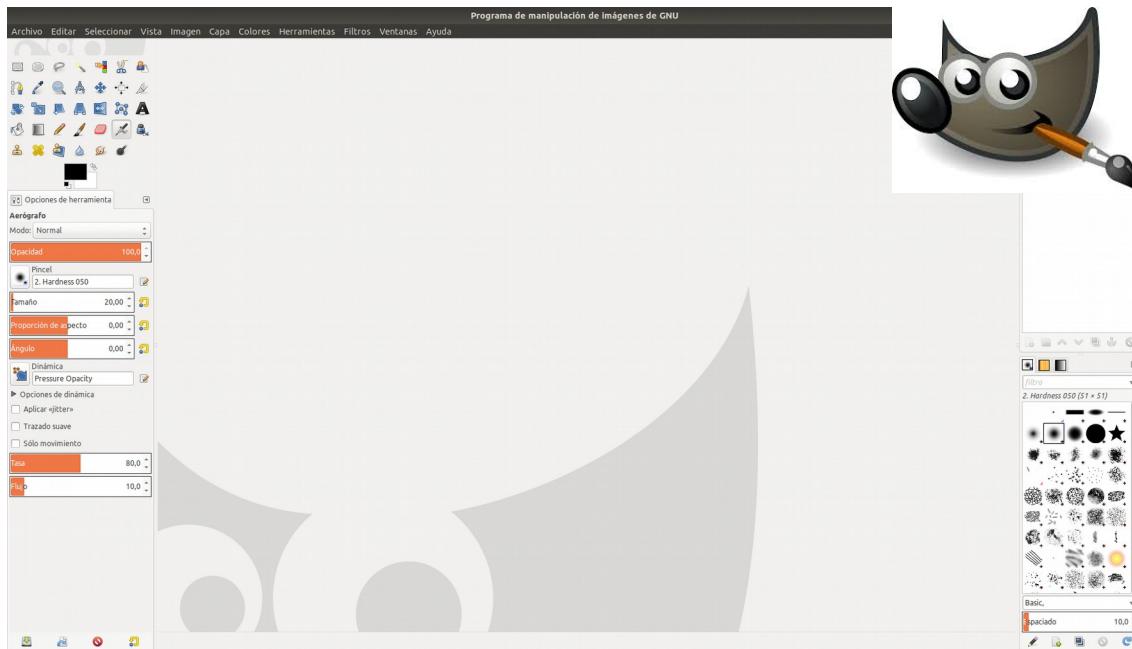
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    setMaterialTheme();
    return MaterialApp(
      theme: GuideFoodTheme,
      debugShowCheckedModeBanner: false,
      title: 'Material App',
      initialRoute: 'splash',
      routes: getApplicationRoutes(),
    );
  }
}

```

- **Gimp2:** Se trata de una herramienta de diseño de imágenes que me ha permitido realizar retoques a imágenes como la que encabeza el appBar en la vista del listado de recetas y otras muchas.



- **AVD de Android Studio:** Ha sido la herramienta que me ha permitido hacer las depuraciones ya que carezco de dispositivo Android. Permite emular dispositivos móviles en pantalla.



Este dispositivo es el que he utilizado para realizar las pruebas. Es el Pixel 3a XL de google.

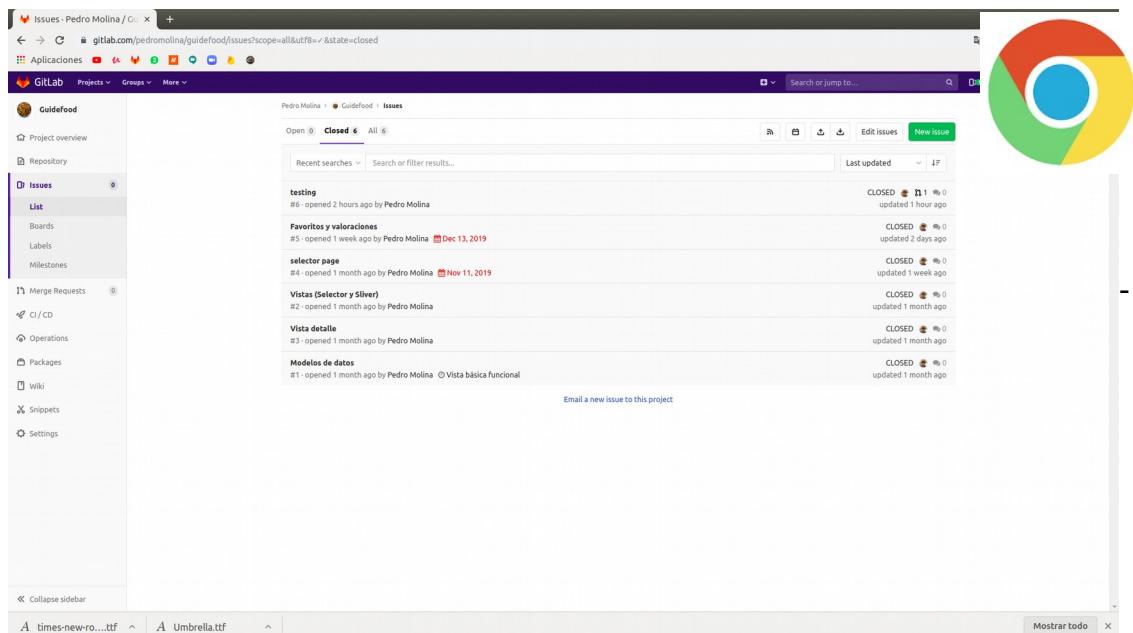
- **Slack:** Herramienta de comunicación que he utilizado para comunicarme con mis compañeros de trabajo y que me ha servido a la hora de hacer el setup de mis equipos.

The screenshot shows the Slack web interface. On the left, there's a sidebar with navigation links like Archivo, Editar, Ver, Historial, Ventana, Ayuda, and a dropdown for 'ASPgems'. Below that are sections for Hilos de conversaciones, Canales (#desarrollo, #english, #general, #mobile\_dev, #oficial, #oficina-jen, #random, #tmp-pizzas), and Mensajes directos. A 'Slackbot' channel is selected. The main area shows a timeline of messages. On July 26, Slackbot says: 'If you're not sure how to do something in Slack, Just type your question below.' On September 27, it says: 'To get the most out of Slack, it's good to be where the conversation's happening: in channels! You can always browse a full list by choosing Channels on the left, or take a look at a few we recommend: #oficial - 53 members Popular channel in ASPgems, #desarrollo - 49 members Popular channel in ASPgems, #english - 46 members Popular channel in ASPgems.' On October 1, it says: 'Si quieres disponer de un modo sencillo de acceder a Slack siempre que lo necesites, descarga la aplicación para ordenador. (Además, tendrás más control sobre tus notificaciones). Descargar la app para ordenador'.

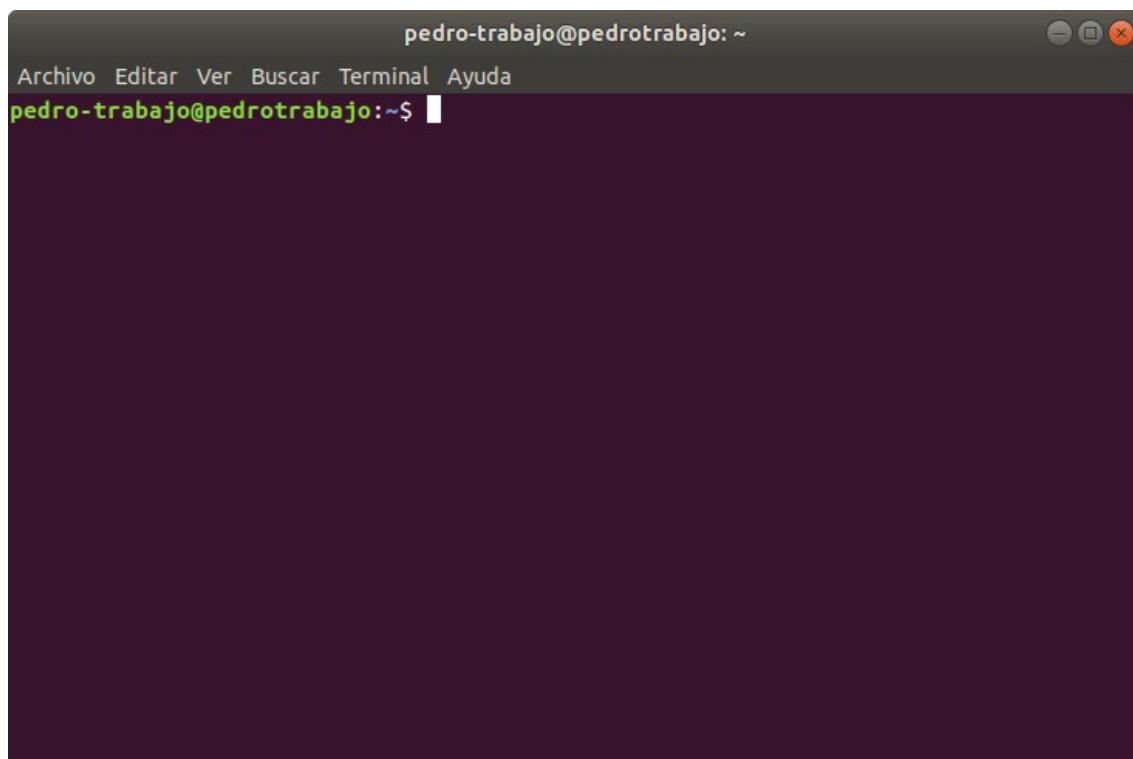
- **Postman:** Postman permite realizar pruebas de depuración con servidores http y ese ha sido el motivo principal por el cual lo he utilizado, para hacer peticiones http y testear mis servidores.

The screenshot shows the Postman application interface. At the top, there are tabs for File, Edit, View, Help, and a workspace dropdown. The main area shows a 'History' tab with a list of API requests. One request is highlighted: a POST to 'http://localhost:3006/favoritos/5de32e2d2cd8d359156a394'. The 'Params' tab shows parameters: email (pedromolines@gmail.com), receta (1), comentario (Pero que buena pinta), valoracion (9), and id (5de32e2d2cd8d359156a394). To the right, there's a large orange circular icon with a white pen writing on a document. The bottom of the screen has a toolbar with icons for Bootcamp, Build, Browse, and others.

- **Chrome:** Lo he utilizado para la descarga de assets, depuración con el backend para hacer consultas http y también para la consulta de información y obtención de recursos. También para utilizar gitlab y tener mi repositorio en la nube.



**Terminal:** La terminal de ubuntu me ha permitido hacer importaciones y exportaciones de datos con mongo, comprobar ips, comprobar tráfico de puertos y si estaban en uso o no. Además la he utilizado para instalación múltiple de recursos como programas además de con git.



## 6. MÉTODO

### 6.1 BACKEND INICIAL (JSON SERVER)

Los objetivos lógicos primarios son hacer que la api rest sea instalada en un servidor web para poder utilizar nuestra aplicación desde cualquier parte del mundo con solo tener conexión a internet.

Para el testing al principio estoy utilizando Json Server, que es una utilidad del paquete de utilidades y herramientas de nodeJS que nos permite crear rápidamente con un sencillo paso de instalación previa y con un archivo de nuestra estructura y modelos en un Json, ya lo tendremos disponible para utilizar con sus endpoints, get, put, post y delete.

Voy a desplegar aquí un sencillo tutorial de instalación de Json server:

El primer paso es escribir en un terminal de ubuntu:

```
npm install -g json-server
```



Si node no está previamente instalado deberemos instalarlo con:

```
sudo apt install nodejs
```

Si se trata de windows deberá instalarse desde <https://nodejs.org/en/download/> .

Ya completados estos pasos queda correr nuestro servidor en local:

```
json-server --watch data.json
```

Nuestro archivo json contiene algunos datos para realizar pruebas y las primeras vistas a través de la petición http.

He decidido llenar yo la base de datos porque he intentado servirme de una api externa pública de recetas y pasarlal a través de una api de traducción pero los resultados han sido pésimos. La api utilizada ha sido <http://www.recipepuppy.com/> , que está en inglés.

Este es un ejemplo del json que utilizo con json server:

```
{  
  "ingredientes": [  
    {  
      "id": 1,  
      "nombre": "Tomate",  
      "tipo": "Rojo",  
      "imagen": "https://www.stickpng.com/assets/images/580b57fc9996e24bc43c238.png",  
      "medida": "g"  
    },  
    {  
      "id": 2,  
      "nombre": "Pimiento",  
      "tipo": "Rojo",  
      "imagen": "https://www.stickpng.com/assets/images/580b57fc9996e24bc43c229.png",  
      "medida": "g"  
    },  
    ],  
  "recetas": [  
    {  
      "id": 1,  
      "nombre": "Pollo asado",  
      "descripcion": "Lo primero que hacemos es limpiar el pollo de posibles restos de grasa y vísceras que pueda tener. Pondremos especial cuidado en retirar las plumitas que le puedan haber quedado. Le pasamos un agua para limpiarlo bien y lo secamos con papel de cocina. Es importante que quede seco para que en el proceso de horneado el pollo no comience cociéndose, lo que queremos es que se haga bien horneado. En un mortero machacamos la pimienta negra y la sal. Con esta mezcla untamos bien el pollo por todas partes, tanto dentro como fuera. Cortamos el limón por la mitad y exprimimos un poco de su jugo por encima y dentro del pollo. Reservamos el resto del limón. En un vaso mezclamos el aceite con el tomillo. Con la ayuda de un pincel de cocina pintamos el pollo con la mezcla por dentro y por fuera. Reservamos el aceite de oliva virgen extra sobrante. Colocamos el pollo en la bandeja de horneado y metemos las mitades de limones en su interior. Añadimos también los dientes de ajo pelados y aplastados dentro del pollo. Pelamos las patatas, las cortamos en rodajas y las repartimos por la bandeja. Regamos tanto el pollo como las patatas con el aceite con tomillo sobrante.",  
      "ingredientes": [  
        10,  
        9,  
        8,  
        7,  
        11,  
        12,  
        13  
      ],  
      "calificacion": 0.0,  
      "imagen": "https://freepngimg.com/thumb/chicken/1-chicken-head-png-image-thumb.png"  
    }  
  ]  
}
```

## 6.2. SERVIDOR DE IMAGENES

Durante una etapa del proyecto decidí montar un servidor de imágenes que me permitiera acceder a ellas desde una url guardada en cada objeto json que cargaría de mi base de datos mongo.

Realicé un montaje básico de un servidor en nodejs sirviendo una carpeta estática de acceso público desde un puerto de mi url local para comprobar que todo funcionaba y así fue, la estructura lógica de drivers contaba con express, driver que servía mi sistema en la red.

El código empleado para servir una carpeta como pública:

```
const express = require('express');
const App = express();
App.use('/static', express.static('public'))
module.exports = App;
```

En el archivo server.js le estoy pasando el puerto por el cual quiero que me sirva las imágenes. En el archivo config dispongo de esta variable:

```
const CONFIG = require('./app/config/config');
const App = require('./app/app');
module.exports =
{
  PORT: 3007,
  App.listen(CONFIG.PORT, function (error) {
    if (error) return console.log(error);
    console.log(`Servidor corriendo en el puerto: ${CONFIG.PORT}`);
  });
}
```

Esta es la estructura de directorios que contiene las imágenes tanto de recetas como de ingredientes.

```
const CONFIG = require('./app/config/config');
const App = require('./app/app');

App.listen(CONFIG.PORT, function (error) {
  if (error) return console.log(error);
  console.log(`Servidor corriendo en el puerto: ${CONFIG.PORT}`);
});
```

Mi intención inicial con respecto al servidor de imágenes era montarlo en una web pero aquí se me cerraron varias puertas.

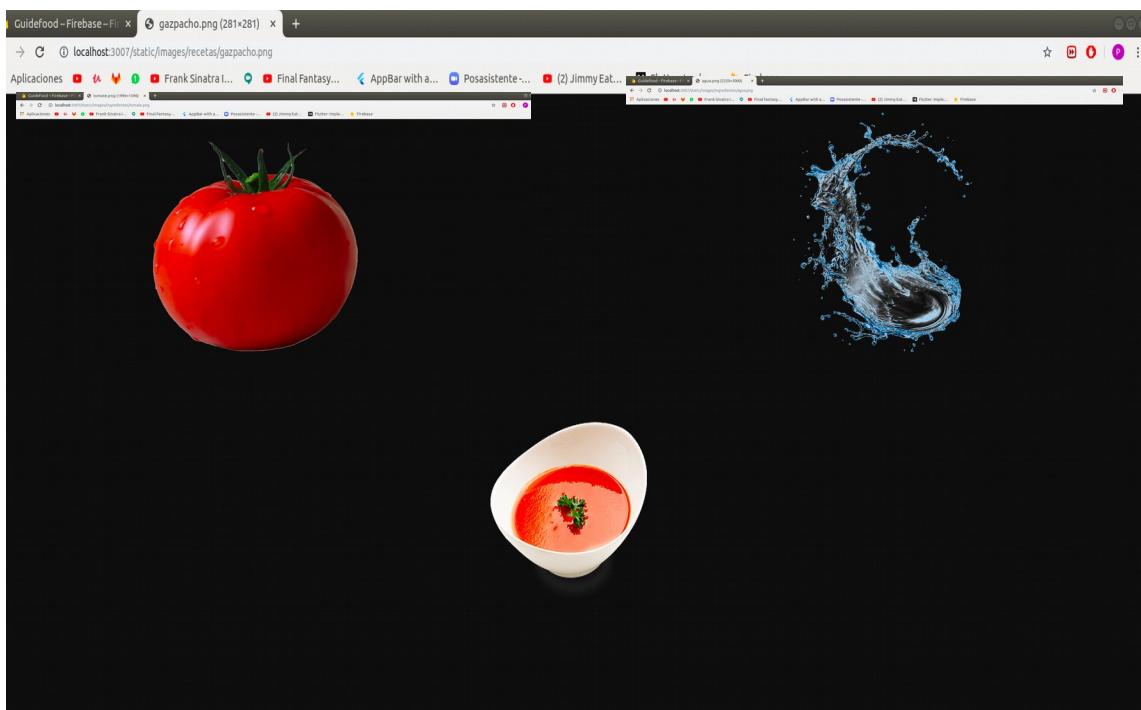
-Derechos de imagen: Si montaba un servidor público con imágenes de recetas e ingredientes obtenidas de sitios con derechos sobre tales imágenes podría meterme en líos con la ley de protección de datos y de este modo estudiar derecho desde adentro.

Este fue un motivo de peso para desechar dicha idea.

-Tiempo: Siguiendo el consejo de mi tutor de prácticas en la empresa, prefiero enfocarme en las funcionalidades de flutter y toda la potencia en vistas y animaciones que proporciona más que en apartado de backend lo cual puede suponer un freno y teniendo en cuenta que la magnitud de este proyecto y los dos meses que me han sido concedidos para realizarlo son temas incompatibles, he decidido poner una cruz a este apartado del backend.

Finalmente por agilizar el desarrollo he utilizado los enlaces a las imágenes directamente de los servidores de origen, introduciéndolos en mi json y así en mongodb.

Estos son algunos ejemplos de peticiones a mi servidor con  
[http://localhost:3007/static/images/recetas/\\*](http://localhost:3007/static/images/recetas/*)



### 6.3. SERVIDOR DE DATOS (BACKEND PRINCIPAL)

La disposición del backend se basa en un sistema de servidor de nodejs.

La estructura de carpetas se basa en modelo controlador separando también la configuración del servidor de node y del puerto desde dónde se podrá acceder a los datos de la api.

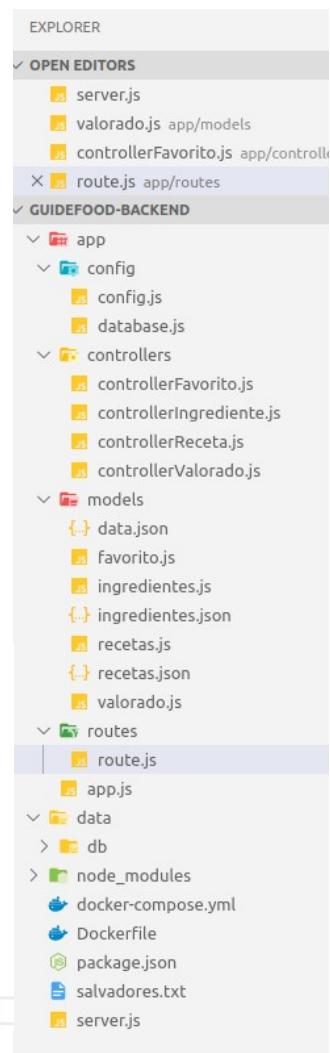
También se pueden visualizar los controladores que disponen de los endpoints de cada modelo, como podemos apreciar en las próximas imágenes.

Este es un ejemplo de modelo de mongoose que he formado.

```
'use strict';
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var IngredientesSchema = new Schema({
  id: Number,
  nombre: String,
  tipo: String,
  imagen: String,
  medida: String
});

module.exports = mongoose.model('Ingrediente', IngredientesSchema);
```



El controlador contiene lo siguiente:

```
function index(req, res) {
  Ingrediente.find({}, function (err, igrediente) {
    if (err)
      res.send(err);
    res.json(igrediente);
  }).then(ingredientes => {
    if (ingredientes.length) return res.status(200);
    return res.status(204).send({ message: 'NO HAY CONTENIDO' });
  }).catch(error => res.status(500).send({ error }));
}
```

Para cada endpoint que presentamos a las rutas:

```
module.exports = {
  index,
  show,
  create,
  update,
  remove,
  find
}
```

En esta imagen podemos ver como se controla la conexión con la base de datos de mongo.

Y en config.js:

```
module.exports = {
  PORT: 3006,
  DB: process.env.DB || 'mongodb://127.0.0.1:27017/guidefood'
}
```

```
const mongoose = require('mongoose');
const CONFIG = require('./config');

module.exports = [
  connection: null,
  connect: function () {
    if (this.connection) return this.connection;
    return mongoose.connect(CONFIG.DB, {
      useNewUrlParser: true,
      useUnifiedTopology: true }).then(connection => {
      this.connection = connection;
      console.log('Conexion a Base de Datos Exitosa');

    }).catch(error => console.log(error));
  }
]
```

Para controlar las rutas que se llaman con cada endpoint y url tenemos el archivo de rutas.js:

```
const express = require('express');
const IngredienteCtrl = require('../controllers/controllerIngrediente');
const RecetaCtrl = require('../controllers/controllerReceta');
const ValoradoCtrl = require('../controllers/controllerValorado');
const FavoritoCtrl = require('../controllers/controllerFavorito');

const Router = express.Router();

Router.get('/ingredientes', IngredienteCtrl.index)
  .post('/ingredientes', IngredienteCtrl.create)
  .get('/:key/:value', IngredienteCtrl.f
  .put('/:key/:value', IngredienteCtrl.f
  .delete('/:key/:value', IngredienteCtrl

Router.get('/recetas', RecetaCtrl.index)
  .post('/recetas', RecetaCtrl.create)
  .get('/:key/:value', RecetaCtrl.find, const RecetaCtrl: {
    index: (req: any, res: any) =
    show: (req: any, res: any) =
    create: (req: any, res: any) =
    update: (req: any, res: any) =
    remove: (req: any, res: any) =
    find: (req: any, res: any, r
  })
  .put('/:key/:value', RecetaCtrl.find, RecetaCtrl.update)
  .delete('/:key/:value', RecetaCtrl.find, RecetaCtrl.remove);
```

En el archivo de App.js tenemos la declaración de express para utilizar el fichero de rutas y el body-parser de json:

```

const express = require('express');
const bodyParser = require('body-parser');

const App = express();
const rutas = require('./routes/route');

App.use(bodyParser.json());
App.use(bodyParser.urlencoded({ extended: false }));

App.use('/', rutas);
module.exports = App;

```

Y en el server.js, que es el archivo que controla enlace de todo lo anterior y su correcto funcionamiento, sirviendo el servicio por el puerto indicado y escuchando las conexiones entrantes:

```

const Database = require('./app/config/database');
const CONFIG = require('./app/config/config');
const App = require('./app/app');

Database.connect();

App.listen(CONFIG.PORT, function (error) {
  if (error) return console.log(error);
  console.log(`Servidor corriendo en el puerto: ${CONFIG.PORT}`);
});

```

En el fichero de docker-compose.yml he utilizado este código, aunque finalmente por problema de puertos y falta de tiempo no he podido configurarlo correctamente:

```

version: '3'
services:
  app:
    container_name: docker-node-mongo
    restart: always
    build: .
    ports:
      - '3006:3006'
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
    ports:
      - '27017:27017'
    volumes:
      - ./data/db:/data/db

```

En el backend estoy utilizando nodemon que me permite realizar cambios en caliente en el servidor con el demonio de nodjs lanzado.

Los comando más utilizados en misistema de backend han sido los siguientes:

netstat -aon | grep 27017, para ver que servicio tenía ocupado el puerto 27017, que usualmente era mongo y tenía que realizar el comando siguiente;

sudo service mongodb restart, para reiniciar el servicio de mongodb y liberar los puertos que estuviera ocupando en el anterior lanzamiento de servicio.

sudo mongod --dbpath ./data/db/ --port 27017, lanzamiento de mongo en un puerto concreto y en una estructura de directorios previammente definida.

ps -aux | grep mongo, utilizado para tratar de encontrar si el servicio estaba lanzado y su puerto.

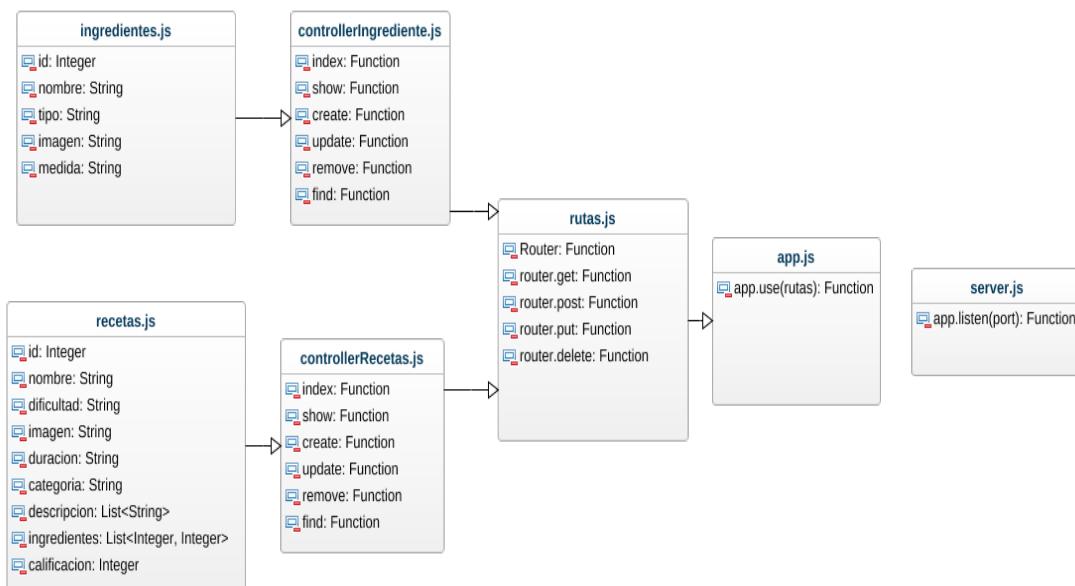
Los próximos comandos han sido utilizados en múltiples ocasiones para importar los archivos json de datos a la base de datos de mongo:

```
mongoimport --db guidefood --collection recetas --file ./app/models/recetas.json --jsonArray
mongoimport --db guidefood --collection ingredientes --file ./app/models/ingredientes.json --jsonArray
```

Este es un comando de mongo que era utilizado para borrar la base de datos cuando quería setear un flujo grande de datos y actualizarlos desde el fichero json:

`db.dropDatabase()`

Diagrama de clases de la estructura del backend:



## 6.4. FRONTEND CON FLUTTER

### 6.4.1. DATOS

Para realizar peticiones al backend he utilizado una clase llamada ApiProvider que provee de datos con una petición a las rutas de ingredientes y recetas.

En flutter como en Javascript se utilizan fundamentalmente los futures que permiten funciones asíncronas. Serían equiparables a las promesas.

Aquí podemos ver el método para la petición de recetas, más abajo está la petición para ingredientes, que es exactamente igual solo que el tipo es Ingrediente y la url es /ingredientes.

```
class ApiProvider {
  bool _cargando = false;
  String _apiKey = 'fbe9ea629dc8abd2036403c3d5a1e0c2';
  String _urlRecetas = 'http://192.168.0.28:3006/recetas';
  String _urlIngredientes = 'http://192.168.0.28:3006/ingredientes';
  String _urlValorados = 'http://192.168.0.28:3006/valorados';
  String _urlFavoritos = 'http://192.168.0.28:3006/favoritos';

  Future<List<Receta>> getRecetas() async {
    var resp;
    if (_cargando)
      return [];
    else
      _cargando = true;

    try {
      resp = await http.get(Uri.encodeFull(_urlRecetas),
        headers: {"Accept": "application/json"});
    } catch (e) {
      print(e);
      return [];
    }
    final decodedData = json.decode(resp.body);

    final recetas = new Recetas.fromJsonList(decodedData);
    _cargando = false;
    return recetas.items;
  }
}
```

Para construir una vista con una petición a la api existe la posibilidad de utilizar un FutureBuilder, que es un componente en el cual, pasándole un future podemos utilizar su constructor para pasarle los datos a un hijo, que puede ser una lista a la cual le seteamos uno por uno con la iteración de este componente, los elementos que serán introducidos en widgets y dibujados.

EL próximo paso a realizar para mejorar la estructura de datos sería convertir el sistema de petición de datos de un patrón provider a un patrón bloc y tal vez añadiendo un repository junto a este para permitirnos realizar el mínimo número posible de peticiones, suponiendo así un menor uso de datos en caso de no estar conectados a una wifi mientras usamos la app.

Otro tema a tener en cuenta es el cargo que supone la descarga de las imágenes, existe un package para flutter llamado cached\_network\_image que permite descargar las dependencias para usar un sistema de cacheado de imágenes que reduce significativamente el coste de descarga en megas.

Para más información, visitad [https://pub.dev/packages/cached\\_network\\_image](https://pub.dev/packages/cached_network_image).

#### 6.4.2. LA VISTA INTRO SPLASH

Entramos a mi parte favorita: las vistas; Hemos de empezar por el principio, la primera vista que visualizamos cuando lanzamos la aplicación de Guidefood es una vista sencilla que cuenta con dos elementos fundamentales en lo que a visualización se refiere: un gradiente, y una imagen de un oso goloso, que es más bien un gif (imagen en movimiento).

En el método Build de esta vista (que es el método que construye la vista principal),

Nos encontramos primero con un contenedor cuyo hijo directo es un stack, que permite apilar una lista de widgets encima de otros.

El contenedor tiene una propiedad llamada decoration en la cual podemos insertarle un objeto tipo BoxDecoration que a su vez nos permite añadir colores de fondo, gradientes, imágenes, bordes, sombras etc.

En este caso yo me he determinado por añadir un gradiente, el cuál he realizado con el siguiente código:

```
gradient: LinearGradient(
  begin: Alignment.topRight,
  end: Alignment.bottomLeft,
  colors: [Color(0xFF94CF48),
            Color(0xFF006AB3)],
),
```

En esos dos atributos, end y begin, podemos decirle donde empieza y donde acaba el gradiente, y debajo tenemos un array de colores al que podemos añadirle cuantos colores queramos.

Dentro del stack tenemos un widget positioned, que nos permite posicionar lo que alberga en su interior, en la posición que deseemos dentro del stack, de hecho es un widget que solo se puede utilizar dentro de Stacks.

De este modo hemos posicionado la imagen del oso dentro de este contenedor porque al principio, estaba usando una columna en vez de un stack, y lo que ocurría es que la imagen tardaba en posicionarse unos cuantos milisegundos debido al componente FadeInImage, que permite setear una imagen con una animación de fadein, entonces el texto de debajo se descuadraba y esto dejaba una vista poco elegante que se descuadraba y ajustaba cuando ya se había cargado la vista.



De este modo todos los elementos aparecen en pantalla a su momento.

Este es el código de la imagen:

```
Positioned(
    top: size.height * 0, //Esta linea le dice a flutter que posicione el elemento hijo a 0 de
    //la parte de arriba de la pantalla
    bottom: size.height * 0.5, //Aquí queremos que llegue como tope a la mitad desde
    //abajo.
    left: size.width * 0.1, //Desde la izquierda, empieza a un 10%
    right: size.width * 0.08, //Desde la derecha, deja un margen de un 8%
    child: Container(
        margin: EdgeInsets.only(top: size.height * 0.1), //margen por arriba de un 10%
        width: size.width * 0.7, //Anchura de un 70%
        child: FadeInImage(
            fadeInDuration: Duration(milliseconds: 300), //Duración de la animación de
            300mls
            image: AssetImage("assets/images/bear.gif"), //Dirección relativa de la imagen
            placeholder: AssetImage("assets/images/transparent.png"),
        )
    ),
),
),
```

La pantalla contiene un temporizador para pasar a la siguiente pantalla. El temporizador ha sido declarado como atributo de la clase e inicializado en el constructor de esta para que se inicie cuando sea construida y en 4000mls o 4s, utilice navigator para ir al listado de recetas.

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:guidefood/src/styles/colores.dart';

class SplashPage extends StatefulWidget {
    @override
    _SplashPageState createState() => _SplashPageState();
}

class _SplashPageState extends State<SplashPage> {
    Timer _timer;
    SplashPageState() {
        timer = new Timer(const Duration(milliseconds: 4000), () {
            setState(() {
                Navigator.pushReplacementNamed(context, "listado");
            });
        });
    } // Timer
    @override
    void dispose() {
        super.dispose();
        _timer.cancel();
    }
}
```

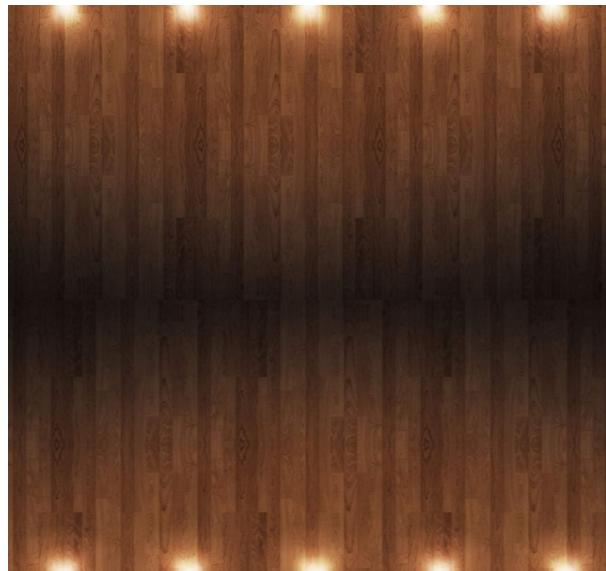
La vista se carga siempre que iniciamos la aplicación, sería más óptimo guardar una instancia de haber iniciado sesión previamente para que no se muestre y vaya directamente a la parte del listado.

#### 6.4.3. LA VISTA DEL LISTADO

En esta vista he utilizado múltiples elementos visuales y de animaciones tales como imágenes, la animación Hero, que es un widget nativo de flutter y otros muchos assets.

Esta es la imagen de fondo que he editado con GIMP 2.0 para poner sobre ella la lista de recetas.

La imagen la he extraído de google.es imagenes.



Esta es la imagen que he utilizado como cabecera del sliverAppBar, que es un widget con animación y que al hacer scroll se comprime hasta que solo queda un appbar más pequeña en la parte de arriba.

Esta vista está contenida en un scaffold que contiene un Drawer customizado llamado DrawerGuidefood que se despliega con un botón del navbar en la parte superior de la derecha de la pantalla.



Este appbar custom nos permite setear un color de fondo para esta y varios botones, en este caso el número de botones es el que tiene por defecto y el color es transparente, aunque cuando se comprime la barra al hacer scroll el color es marrón con opacidad de un 30%.



Permitiendo así visualizar los elementos que hacer scroll por debajo de esta.

Aquí podemos ver la vista principal de la lista de recetas con el appBar desplegado.



En el appBar podemos encontrar que en su atributo title hemos seteado el appBar contraida pequeña que ha sido definida para todas las vistas, que es la que contiene los botones de atrás y el drawer.

Ocupando el atributo de flexibleSpace tenemos un Layout builder con un FlexibleSpaceBar que es la parte que contiene la imagen del sliverappbar y que se contrae al hacer scroll.

La imagen está añadida sobre el widget FadeInImage que la anima para entrar lentamente que ha sido añadida dentro del atributo de FlexibleSpaceBar background.

La lista está dentro de un future builder que es una clase que nos permite utilizar un constructor, pasándole un future, que en nuestro caso es una llamada a la api del provider para obtener datos de la lista de recetas que hay en nuestro repositorio.

La parte de la lista dispone de un SliverList, todo esto añadido junto al SliverAppBar dentro de un CustomScrollView que es un listview con algunos atributos más avanzados tales como animaciones y el uso de sliver...

Son contenedores de un tipo diferente a los contenedores, por lo que son muy excrupulosos a la hora de insertarles otros widget como hijos o como padres y psuelen provocar que en desarrollo crashee la aplicación si no son introducidas correctamente.

En la imagen de debajo podemos ver como se construye el FutureBuilder:

```
FutureBuilder(
  future: apiProvider.getRecetas(),
  builder: (BuildContext context, AsyncSnapshot<List> projectSnap) {
```

Tras comprobar dentro del FutureBuilder si la conexión ha sido establecida y si la lista pedida contiene datos, entonces se salta a un return de la lista de recetas:

```
return SliverList(  
    delegate: SliverChildBuilderDelegate( context, index ) {  
        Receta receta = projectSnap.data[index];  
        return SliverItem(  
            receta: receta,  
        );  
    },  
    childCount: childCount,  
),  
);
```

El SliverList delega la vista sobre el constructor de child que itera la lista según el número de veces dado por childCount, que es un atributo de esta y que le he dado el valor del array de recetas que ha sido obtenido a través del future y con el atributo de los List de .length que nos da el número de elementos de una lista.

Este constructor devuelve un objeto widget llamado SliverItem que es un objeto custom que he creado para cada vista de cada receta al que le paso con su constructor la receta obtenida con el index de la lista.

### El objeto SliverItem

El objeto sliver item es un widget definido en una clase aparte cuyo constructor:

```
final Receta receta;
```

```
SliverItem({this.receta});
```

Al que se llama al construirlo:

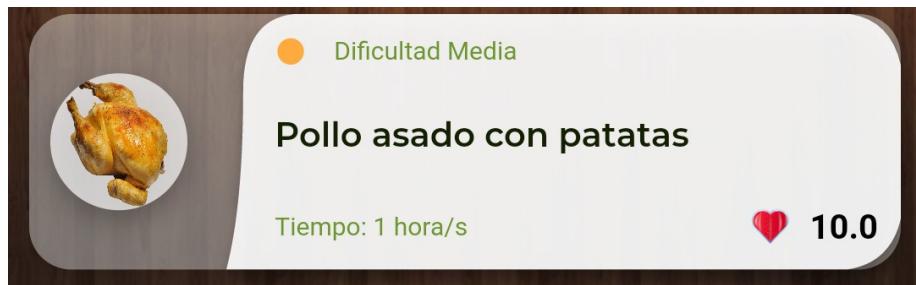
```
new SliverItem(  
    receta: Receta(),  
);
```

El widget principal en el que está metido el resto de la vista es un GestureDetector, que permite añadir funciones de estado y escuchadores de tipo onTap, onLongPressed, behavior y otra larga serie de funcionalidades de control de acción sobre el elemento hijo:

```
GestureDetector(  
    onTap: () {  
        Navigator.pushNamed(context, "detalle", arguments: receta);  
    },
```

En la función onTap he añadido un Navigator que es una clase que nos permite navegar entre vistas, en este caso navegamos a la vista del detalle pasándole un argumento receta a la vista del detalle que más tarde será recogido y sobre este se pintará la vista.

Así es como se visualiza un SliverItem:



- El elemento sliver está contenido en un Container() con una altura fija de 14% del alto.
- También tiene un margin simétrico que añade un margen que he determinado de 10 px en horizontal y en vertical (lo más óptimo para evitar descuadres indeseados por temas de responsividad es utilizar las medidas con porcentajes).
- La decoración del contenedor responde al uso de un BoxDecoration:

```
decoration: BoxDecoration(
  borderRadius: BorderRadius.all(Radius.circular(20)),
  color: Colors.white38,
  boxShadow: <BoxShadow>[
    BoxShadow(
      color: Colors.black26,
      blurRadius: 10,
      spreadRadius: 0.5,
      offset: Offset(0.0, 5.0)),
  ],
),
```



Este decoration dispone de un border radius que redondea el borde con un radio de 20, tiene también un color de fondo blanco con opacidad del 38% y además dispone de una sombra de color negro con 26% de opacidad y un radio de 10 y una extensión gradual cada 0.5 y por último las coordenadas de expansión de la sombra son de 0 horizontalmente y de 5.0 verticalmente en sentido descendente.

Dentro de este contenedor disponemos de un Row (fila) que es un widget que permite añadir elementos que se dispondrán horizontalmente en el interior del padre.

Dentro de este Row hay un Container() con margenes horizontales de 10u y este contenedor tiene ancho y alto el 15% de la anchura, nos interesa que el alto también lo tenga para que ambos lados tengan la misma medida.

A su vez tiene como hijo a otro contenedor que dispone de un BoxDecoration con un shadow igual al que vimos más arriba con un color blanco con opacidad del 70% y un shape (forma) circular.

El hijo de este contenedor es un hero. La animación Hero o widget Hero, es un widget con una animación que permite animar un widget al ser pulsado y cambiar de vista, arrastrando ese widget dentro de la vista hacia la que hemos navegado.

Se trata de un widget nativo que con el solo requisito de pasarle un tag, que debe ser añadido también en la vista hacia la que queremos navegar, como padre del elemento móvil que queremos animar, y este tag debe ser único para ese elemento, por eso lo más óptimo es pasarle un id, así no obtendremos incongruencias entre tags.

```
Hero(
  tag: receta.id,
  child: ClipOval(
    child: FadeInImage(
      placeholder: AssetImage("assets/images/loading.gif"),
      image: NetworkImage(receta.imagen),
      fit: BoxFit.fill,
    ),
  ),
),
```

El ClipOval redondea la imagen que contiene como hija. En el placeholder del FadeInImage he añadido un gif que aparece mientras se carga la imagen que le pasamos por el NetworkImage, que es un proveedor que nos permite obtener una imagen de la red con solo pasar una url, en este caso que almacenamos en la receta.



- El otro elemento del SliverItem dentro del Row es un clipperCard, que contiene un Container con un ClipShadowPath que es un Clipper custom (es un elemento que permite recortar un child según sea definido con un clipper que le pasemos), gracias al cual, además de aplicar un clipper podemos añadir una sombra que le pasamos también al constructor.
- Como hijo del Clipper custom tenemos un contenedor que a su vez contiene una columna para disponer elementos en vertical.
- Dentro del Row tiene un row donde está un Container con un método que según la dificultad pinta un ícono con un color determinado:

```
Color getIconColorDificultad(String dificultad) {
  switch (dificultad) {
    case "Baja":
      return primaryColorLight;
      break;
    case "Media":
      return Colors.orangeAccent;
      break;
    case "Alta":
      return Colors.red;
      break;
  }
}
```



El otro elemento del Row es un texto que imprime la dificultad de la receta:

```
Container(
  margin: EdgeInsets.only(left: size.width * 0.03),
  child: Text(
    'Dificultad ${receta.dificultad}',
    style: textTitle,
  ),
),
```



Dificultad Media

- El texto del nombre este dentro de un widget Flexible que permite que lo que contiene en su interior se alargue todo lo posible y se ajuste al espacio del contenido.

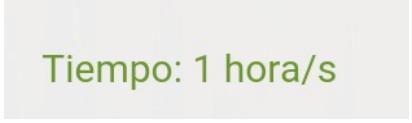
```
Flexible(
  child: Container(
    margin: EdgeInsets.only(left: size.width * 0.03),
    child: Text(
      receta.nombre,
      style: titleTile,
      overflow: TextOverflow.ellipsis,
      maxLines: 2,
    ),
  ),
),
```

TextOverflow permite elegir el comportamiento en caso de que el texto se desborde.

- Debajo hay un Row que contiene el tiempo de preparación y la valoración con un icono:

La duración:

```
Container(
  alignment: Alignment.bottomLeft,
  margin: EdgeInsets.only(left: size.width * 0.03),
  child: Text(
    'Tiempo: ${receta.duracion}',
    style: textTitle,
  ),
),
```



Tiempo: 1 hora/s

Obtiene el tiempo de preparación de la receta.

La vista no dispone de paginación de las recetas, por lo que sería interesante implementarla. La dificultad de implementar una paginación es mínima, ya la he realizado anteriormente en un proyecto que recopilaba una lista de películas de TheMovieDB.

- Por último hablando del SliverItem queda la parte de la calificación que permite visualizar otro pequeño row con un icono obtenido por un metodo getIconCalificacion() y el texto del número double en un widget text, todo dentro de un flexible para ajustarse a la anchura de la row padre:

```
Flexible(
  child: Container(
    alignment: Alignment.bottomRight,
    width: 100,
    height: 20,
    margin: EdgeInsets.only(left: size.width * 0.03),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.end,
      children: <Widget>[
        getIconCalificacion(receta.calificacion),
        Container(
          margin: EdgeInsets.only(left: size.width * 0.02),
          child: Text(
            '${receta.calificacion}',
            style: calificationTile,
          ),
        ),
      ],
    ),
  ),
),
```



Y para colocar los iconos según el puntaje de la valoración uso el método:

```
Image getIconCalificacion(double calificacion) {
  if (calificacion < 2.0)
    return Image(
      image: AssetImage("assets/iconos/heart0.png"),
    );
  if (calificacion >= 2.0 && calificacion < 4.0)
    return Image(
      image: AssetImage("assets/iconos/heart25.png"),
    );
  if (calificacion >= 4.0 && calificacion < 6.0)
    return Image(
      image: AssetImage("assets/iconos/heart50.png"),
    );
  if (calificacion >= 6.0 && calificacion < 9.0)
    return Image(
      image: AssetImage("assets/iconos/heart75.png"),
    );
  if (calificacion >= 9.0)
    return Image(
      image: AssetImage("assets/iconos/heart100.png"),
    );
}
```



#### 6.4.4. LA VISTA DEL SELECTOR

En la vista del selector he empleado una imagen clara de background de madera, que va muy a tono con la temática de la comida.

El selector es una página que hace una petición de datos al provider y gracias a la cual podemos desplazar ingredientes de la parte de la izquierda de la pantalla hacia la derecha, quedándose así en forma de lista en la parte de la derecha y al ser soltados actualizar una vista inferior que contiene una lista de recetas.

La lista de recetas contiene items pulsables que te llevan al detalle de cada receta, además les ha sido implementada la animación hero para que al pulsar la imagen se mueva hasta la otra vista.

Para deslizar los elementos he utilizado la clase `draggable` que es nativa de flutter.  
El item desplazable que he definido ha sido la imagen de los ingredientes .

En la parte de la derecha cada elemento contiene un ícono de eliminar para quitarlos de la lista.

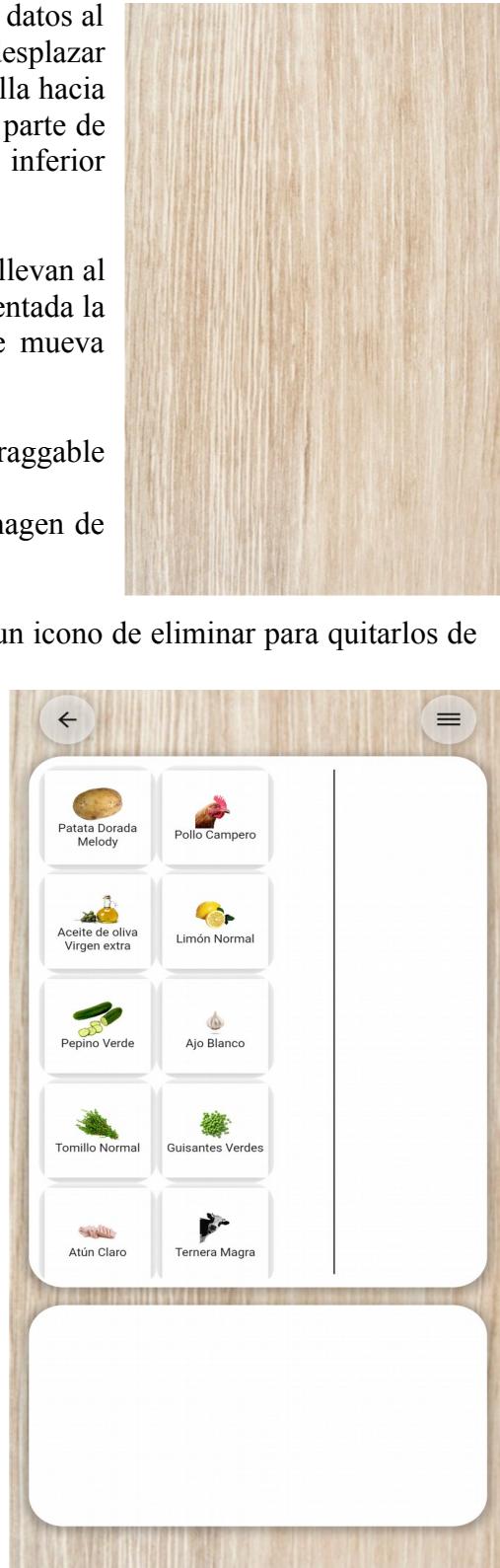
Esta es la vista principal del selector que es la parte más dinámica de la aplicación a pesar de su diseño simplista.

El hueco que hay entre la lista y la barra de separación nos permite hacer scroll por la lista de ingredientes.

La zona de la derecha acogerá los elementos que arrastremos desde la izquierda que actualizará la vista entera realizando una nueva petición.

En la parte inferior encontramos un lienzo sobre el que dibujaremos una lista de recetas compatibles con la lista de ingredientes pasada.

Sería interesante implementar la funcionalidad de poder escribir también una cantidad de ingredientes al añadirlos a la parte de la derecha para que al añadirlo haga un match de ingrediente con cantidad.



En la imagen de la derecha podemos ver la funcionalidad de arrastrar ingredientes gracias a una sencilla integración de código:

El GridSelección, que es la lista de ingredientes de la izquierda, se compone de un contenedor que tiene como hijo un FutureBuilder que hace la petición de datos a la api y rellena la vista a dos columnas con widgets personalizados que he definido en una clase aparte llamada DragBox (buen spanglish), que en su método builder contiene un Draggable, debajo de un Container, que nos permite arrastrar y soltar elementos y producir un efecto al arrastrar de movimiento.

```
Container(
  child: Draggable(
    data: widget.ingrediente,
    child: ContenedorIngredienteWidget(
      seleccionOElegido: true,
      size: widget.size,
      ingrediente: widget.ingrediente),
    onDraggableCanceled: (velocity, offset) {
      setState(() {
        position = offset;
      });
    },
    feedback: Opacity(
      opacity: 0.7,
      child: Container(
        child:
        Image.network(widget.ingrediente.imagen),
        width: widget.size.width * 0.2,
        height: widget.size.width * 0.2,
      ),
    ),
  ),
);
```



En el código podemos ver que como hijo de draggable empleo una clase llamada ContenedorIngredienteWidget que no es más que un widget que contiene el diseño de cada elemento de la lista: la imagen y el nombre son los dos atributos de la vista.

El elemento ContenedorIngrediente contiene un contenedor en el cual hay una columna que contiene la imagen y el nombre más el tipo de ingrediente dentro del widget Text(). Lleva consigo un textStyle y en caso de la imagen contiene un FadeInImage.

En este elemento he definido shadows en el contenedor principal a través de su atributo decoration y dentro del BoxDecoration que le dibuja esas sombras en todos los sentidos dándole un aspecto de cromo flotante.

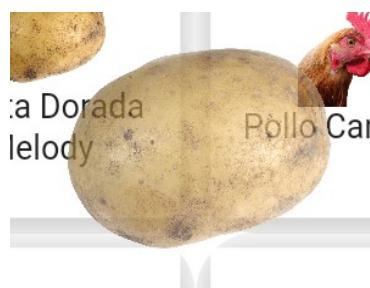


Esta clase es reutilizable pues uno de los atributos principales es un bool o booleano, que nos permite decirle a la clase si queremos un elemento tipo de la columna de selección o con la vista dedicada para el panel de los ingredientes elegidos.

Así que en el build hago una comprobación lógica con un if() y de este modo se discrimina entre una u otra vista.

En el código de más arriba vemos un atributo interesante de draggable que es feedback, feedback es un atributo que permite insertar un widget que será pintado a la hora de arrastrar el elemento.

El widget que hemos pintado dentro de feedback es un Opacity() que permite poner un atributo opacity para cambiar la opacidad del elemento hijo, que será la imagen del ingrediente.

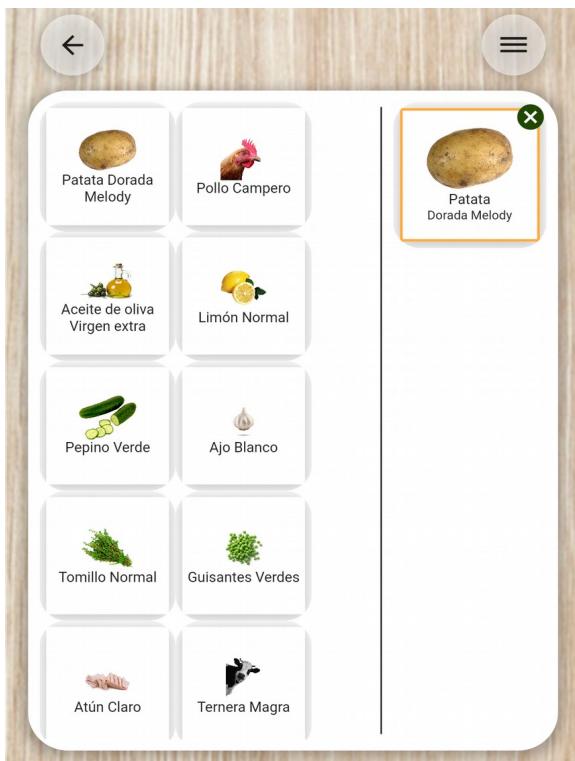


```
feedback: Opacity(  
    opacity: 0.7,  
    child: Container(  
        child: Image.network(widget.ingrediente.imagen),  
        width: widget.size.width * 0.2,  
        height: widget.size.width * 0.2,  
    ),  
,
```

Esta es la vista en la que ya podemos ver la funcionalidad del elemento arrastrado.

El grid elegidos es un método tipo widget que contiene un flexible con el que le decimos que si tamaño ha de ser 1/total de flexes que hayan sido definidos.

Dentro de él hayamos un contenedor que contiene un DragTarget() que es un widget que nos permite detectar si sobre él se ha soltado algún elemento, en este caso utilizamos su atributo onAccept al cual le pasamos el ingrediente que hemos soltado y tras una comprobación para saber si el ingrediente se halla ya dentro de la lista de elegidos con un método dentro de un if()



```
bool existeIngrediente(List<Ingrediente> listaPasada, Ingrediente ingrediente) {
    for (var i = 0; i < listaPasada.length; i++) {
        if (listaPasada[i].id == ingrediente.id)
            return true;
    }
    return false;
}
```

En este método le paso una lista y compruebo uno por uno iterándolo con un bucle for si alguno de los ingredientes de la lista tiene el mismo id, de este modo si los ids son iguales devuelve true, sino, al salir del bucle devuelve un false y así:

```
listaPasada.add(ingrediente);
setState(() {});
```

Añado el ingrediente a la lista de los elegidos y actualizo la vista con setState, que es una función que se puede llamar dentro de los widget con estado (Stateful) y se utiliza para redibujar la vista cuando hay cambios.

Y aquí tenemos el ContenedorIngredienteWidget con valor a false que significa que dibujará la vista de los widgets elegidos.

En este caso el widget de esta vista también está compuesto por un container con sombras, con un border, todo esto dentro de un stack para poder poner el icono de eliminar en la esquina superior derecha. Y en el centro del contenedor una columna con la imagen del ingrediente y el nombre junto al tipo en un container y un widget Text como hijo.



El elemento es pulsable y nos permite visualizar un CustomDialogIngredient que he definido como detalle a cada ingrediente en la lista de elegidos.



Esta vista se compone de una imagen contenida en un widget ClipOval que recorta en forma circular al hijo, contenida a su vez en un Container() con sombras y con el atributo shape del BoxDecoration en BoxShape.circular, que permite que el contenedor sea un círculo.

El contenido esta metido dentro de una columna en la cual tenemos un row que contiene el icono de cerrar el diálogo, el cual también puede ser cerrando clickando fuera del diálogo.

Además del row del icono, dentro de la columna hay otro row que es el que contiene la imagen descrita anteriormente y la descripción del ingrediente, que sería una columna con dos text, uno con el nombre y otro con el tipo.

Por último pasamos a la parte de abajo en la que empleamos un listview builder que permite seleccionar el modo de desplazamiento horizontal de la lista.

El contenedor que contiene la lista es un Container con sombra dentro de un ClipRRect que con su atributo borderRaius permite redondear las esquinas del hijo.



La vista inferior se encuentra en un método llamado areaRecetasMatches, con una anchura del total y una altura del 25% de la altura, ambos atributos definidos dentro del contenedor padre.

El hijo directo es un FutureBuilder que pide al api provider el método de getRecetas de la api, que devuelve un future.

```
FutureBuilder<List<Receta>>(  
    future: provider.getRecetas(),
```

Tras comprobar que la conexión ha sido exitosa y que contiene datos, creo una lista de Receta:

```
List<Receta> recetasMatches = new List<Receta>();
```

La cual es rellenada con un método que comprobará si las recetas hacen match con los ingredientes introducidos:

```
recetasMatches = _recetasMatches(recetas, listaPasada);
```

Este método contiene tres bucles for anidados que buscarán el id dentro de cada receta, el cual está contenido en el atributo ingredientes de cada una que contiene una lista de objetos compuestos por el id en primer lugar, y la cantidad necesaria para realizarla en segundo lugar:

```
List<Receta> _recetasMatches(List<Receta> recetas, List<Ingrediente> listaPasada) {  
    List<Receta> recetasMatches = new List<Receta>();  
    int matches = 0;  
  
    for (var j = 0; j < recetas.length; j++) {  
        for (var x = 0; x < recetas[j].ingredientes.length; x++) {  
            for (var i = 0; i < listaPasada.length; i++) {  
                if (listaPasada[i].id == recetas[j].ingredientes[x][0]) {  
                    matches++;  
                }  
            }  
        }  
        if (matches == listaPasada.length) {  
            recetasMatches.add(recetas[j]);  
        }  
        matches = 0;  
    }  
  
    return recetasMatches;  
}
```

Con el atributo matches compruebo si todos los ingredientes pasados en la lista de elegidos se encuentran también entre los ingredientes de la receta que hará o no, match.

De este modo, si el número de ingredientes que han hecho match comparando los ids es igual al número de ingredientes introducidos desde la zona de elegidos, entonces significa que todos los ingredientes han hecho match con una receta determinada.

Realmente para facilitar un poco el código a nivel de rendimiento, sería más oportuno utilizar otro tipo de atributo en el backend diferente de la lista de listas de id de ingrediente y cantidad, que he creado, porque recorrer un triple bucle con una gran cantidad de elementos puede ralentizar el hilo principal, por eso sería oportuno, o cambiar el modelo, o crear un hilo a parte que devuelva los datos cuando estén listos.

El contenedor de las recetas en el área de coincidencias está contenido por un GestureDetector que permite que al ser pulsado, en su atributo onTap naveguemos a la vista del detalle seteandole como argumento la receta:

```
onTap: () {
  Navigator.pushNamed(context, "detalle", arguments: receta);
},
```

Como hijo tiene un Contenedor() con shadow, con una altura de 25% de la altura del total, y una anchura del 20% del ancho.

En el BoxDecoration tiene un borderRadius:

```
borderRadius: BorderRadius.circular(20),
```



Como hijo contiene una columna que a simple vista diríamos que tiene una imagen con un ClipOval en un contenedor con shadow y circula shape, pero además de esto este elemento estaría contenido por un Hero de modo que animemos la vista al clickar sobre el contenedor.

Debajo también dentro de la columna contiene un Container() con texto del nombre de la receta y el tipo.

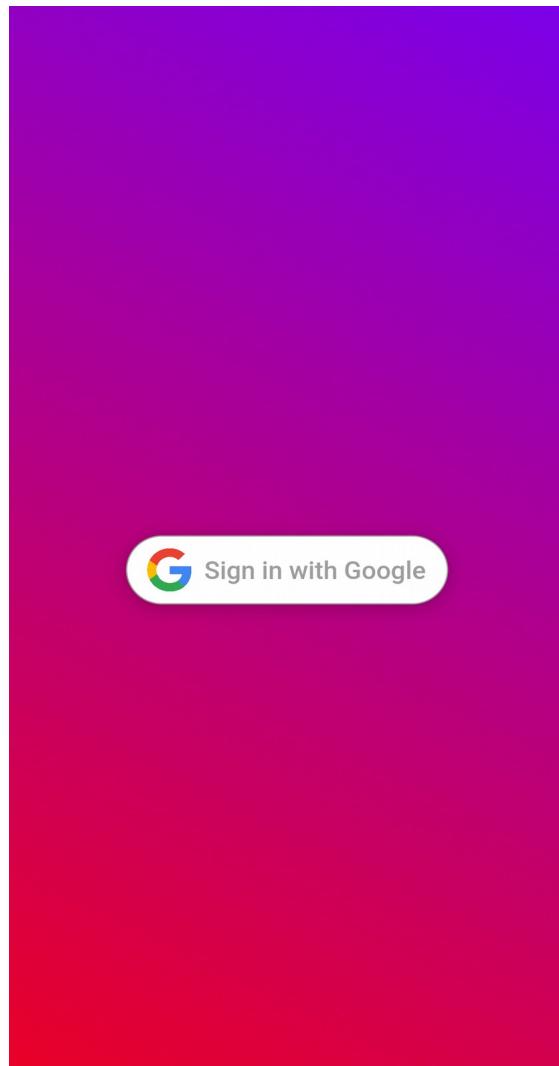
Tanto el icono de la dificultad y el icono de la calificación utilizan el mismo método que en las otras vistas que en código vienen a ser una serie de if() y switch.



#### 6.4.5. LA VISTA DEL LOGIN

La pantalla de login se basa en un gradient como el que vimos anteriormente en la pantalla de inicio o splash, con un color violeta que empieza por la parte superior derecha y un rosa que acaba en la parte inferior derecha:

```
gradient: LinearGradient(  
  begin: Alignment.topRight,  
  end: Alignment.bottomLeft,  
  colors: [  
    Color(0xFF7C00EA),  
    Color(0xFFEA0027),  
  ],  
)
```



El botón del centro es un contenedor con un color blanco de fondo y una sombra y un border radius de 40 que lo hace circular en sus lados.

Dentro del onPressed tenemos el método de login con google:

```
onPressed: () async {  
  await signInWithGoogle()  
  .then((FirebaseUser user) =>  
  Navigator.pushReplacementNamed(  
  context, "favoritos",  
  arguments: user))  
  .catchError((e) => print(e));
```

Se trata de un método asíncrono ya que espera una llamada con Future para la comprobación del usuario.

En caso de resultar una autenticación exitosa se recibe el usuario y navegamos a la vista de los favoritos.

Posteriormente tengo intención de poner un appbar que permita retroceder porque si el usuario cambia de idea no tiene la opción de volver a no ser que sea Android.

#### 6.4.6. LA VISTA DE LOS FAVORITOS Y VALORADOS

La vista de favoritos es una vista muy condensada de información en la que podemos acceder a la lista de recetas que hemos valorado y también a las recetas que hemos marcado como favoritas.

El diseño principal se basa en un stack dentro de un contenedor con el tamaño total de la pantalla, dentro del cual apilamos varios widget, entre otros, dos positioned que posicionamos uno en el top, con lo que podría ser la parte superior de un corazón, y el otro positioned, en la parte inferior con la forma de la parte inferior de un corazón, así de este modo todo el contenido se encuentra englobado en un corazón a pantalla completa.

En la parte superior tenemos el AppBar que he utilizado en el resto de vistas, que permite abrir la vista del Drawer y volver hacia atrás.

Deabajo del clipper con forma ondulada de la parte superior, podemos ver que hay dos iconos cuya funcionalidad es alternar dos vistas, la de la izquierda que es la de valoraciones, en cambio en la derecha tenemos los favoritos.

Estos dos iconos se corresponden con un TabBar que permite cambiarse de vista pulsando uno de los dos iconos o también arrastrando la vista.



```
DefaultTabController(
  length: 2,
  child: Stack(
    children: <Widget>[
      Container(
        margin: EdgeInsets.only(top: size.height * 0.15),
        width: size.width,
        child: Column(
          children: <Widget>[
            _panelUsuario(size, usuario),
            Container(
              height: size.height * 0.71,
              child: TabBarView(
                children: [
                  _vistaValorados(context, size),
                  _vistaFavoritos(context, size),
                ],
              ),
            ),
          ],
        ),
      ),
    ],
  ),
);
```

En la página anterior está escrito el código que corresponde a la parte del TabBar. Se puede ver TabBar debe ser descendiente de DefaultTabController, que es el que controla la paginación, el cual junto con su atributo length le dice a la vista que va a tener dos páginas en nuestro caso. Luego dentro del TabBarView tenemos los tabs que son widgets que componen cada una de las vistas.



Dentro del stack, tenemos también el panel de usuario que lo conforma un Column con un avatar con un ClipOval y con un contenedor que le da una sombra a la imagen, que está compuesta por un FadeInImage con una duración de animación de 300 milisegundos al que le pasamos la imagen del usuario con un NetworkImage.



Deabajo, dentro del mismo column tenemos también el correo del usuario y debajo el nombre y un saludo. Esta información se ha recopilado gracias al usuario que pasamos al hacer el login anteriormente, de modo que obtenemos los datos del usuario.

El texto tiene una fuente de tipo montserrat y para que no haga overflow y se desborde tiene un TextOverflow.ellipsis.

La parte de la lista es un widget llamado vistaValorados que aglutina dos FuturesBuilders anidados para hacer la petición de las recetas y también la petición de los valorados, que es un tipo definido tanto en backend como en la aplicación con un modelo en Dart:

```
class Valorado {
  String email;
  int receta;
  double valoracion;
  String comentario;
  String fecha;

  Valorado(
    {@required this.email,
     @required this.receta,
     @required this.valoracion,
     @required this.comentario,
     this.fecha});

  Valorado.fromJsonMap(Map<String, dynamic> json) {
    email = json['email'];
    receta = json['receta'];
    comentario = json['comentario'];
    valoracion = json['valoracion'];
    fecha = json['fecha'];
  }
}
```

En caso de que ambos futures realicen la petición y sea satisfactoria dentro de un CustomScrollView le pasamos primero un SliverToBoxAdapter, que no es más que un elemento que traduce un contenedor tipo renderBox a renderSliver, osea, que al fin y al cabo, nos permite meter dentro un container con un texto que dice “Recetas valoradas”.

## Recetas valoradas

Esta etiqueta de texto insertada dentro de la vista móvil permite diferenciar en qué vista nos encontramos, y justo debajo tenemos la lista de recetas que se llenan comparando el email del usuario logueado y el email de la lista de favoritos que descargamos del backend, de este modo, si un favorito tiene el mismo email del usuario logueado, comparamos ese favorito con el listado de recetas y de ese modo se añade una RecetaValorada, que es un objeto que contiene la receta y el objeto valorado, así a la hora de pintar la vista resulta más sencillo de distribuir.

```
class RecetaValorada {
    Receta receta;
    Valorado valorado;
    RecetaValorada({this.receta, this.valorado});
}
```

La vista de las recetas valoradas se compone de una columna en un método llamado \_recetaValoración, y con las esquinas recortadas por un ClipRRect y un borderRadius, en primer lugar, el padre de la columna es un contenedor de este modo tiene un fondo grisáceo para unificar la vista de cada ítem que toma el ancho total del width y la altura fija del SliverItem y el total del despliegue del texto.

Como acabo de mencionar, disponemos de nuevo del SliverItem del que disponíamos en la vista del listado de recetas, recicrándolo así en su plenitud y colocándolo en la parte superior del ítem.

Justo debajo tenemos un Row en el que podemos observar un texto que dice “Valoración” y a su derecha tenemos otro

Row con dos componentes, uno un Text y otro con una imagen del corazón según la valoración que hemos mandado.



Por último debajo de esta fila tenemos el texto del comentario de la valoración que hemos hecho:

```
List<RecetaValorada> getRecetasValoradas()
List<Valorado> valoradas, List<Receta> recetas) {
List<RecetaValorada> recetasValoradas = new List<RecetaValorada>(),
for (var i = 0; i < valoradas.length; i++) {
for (var j = 0; j < recetas.length; j++) {
if (valoradas[i].email == usuario.email &&
valoradas[i].receta == recetas[j].id) {
RecetaValorada recetaValorada =
new RecetaValorada(receta: recetas[j], valorado: valoradas[i]);
recetasValoradas.add(recetaValorada);
}
}
return recetasValoradas;
}
```

**Valoracion**  1.0  
Es que me parece increíble que pongan los animales así vivos y luego la receta, osea, esto va en contra de los derechos de los animales, voy a denunciarlos por opresores.

Este es el método empleado para recoger las valoraciones.

La anterior era la vista de los valorados; la vista de los favoritos es más sencilla de explicar después de haber visto ya esta vista anterior.

No cambia nada a excepción del listado, que en este caso volvemos a ver los SliverItems como los veíamos en la pantalla del listado.

```
class RecetaFavorita {
Receta receta;
Favorito favorito;
RecetaFavorita( {this.receta, this.favorito} );
}
```

Al igual que antes empleo una clase dedicada al aglutinamiento de la receta y el tipo favorito que es también un tipo que contiene el email y el id de la receta.

```
List<RecetaFavorita> getRecetasFavoritos(
List<Favorito> favoritos, List<Receta> recetas) {
List<RecetaFavorita> recetasFavoritas = new
List<RecetaFavorita>();

for (var i = 0; i < favoritos.length; i++) {
for (var j = 0; j < recetas.length; j++) {
if (favoritos[i].email == usuario.email &&
favoritos[i].receta == recetas[j].id) {
RecetaFavorita recetaFavorita =
new RecetaFavorita(receta: recetas[j], favorito:
favoritos[i]);
recetasFavoritas.add(recetaFavorita);
}
}
}
```



Este es el método empleado para encontrar las recetas favoritas.

#### 6.4.7. LA VISTA DEL DETALLE

Esta es la vista detalle, la más rica en funcionalidades (por terminar de implementar, al menos en el momento en el que redacto la documentación).

Disponemos del appbar que hemos estado usando en las demás vistas.

En esta vista tenemos un clipper en la parte superior, dentro de un stack y además un método que construye el avatar u consiste en ClipOval y un Container() con BoxShape.circular y un FadeInImage en su interior.

Debajo un Row que contiene el icono de dificultad dentro de una columna en la cual tenemos el icono coloreado con la dificultad y el texto del tipo de la dificultad justo debajo.

A la derecha tenemos la calificación con un column definiendo el icono y debajo el texto de la puntuación.

Debajo tenemos la categoría de la receta en un Text(). A otro lado tenemos el tiempo de realización con un icono de un reloj.

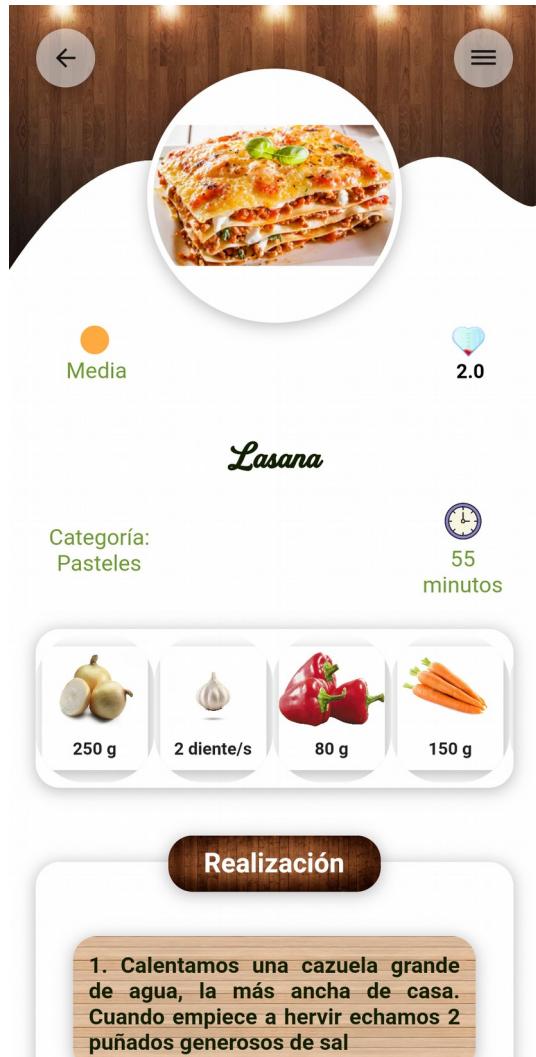
Debajo una fila de ingredientes que es deslizable y scrollea lateralmente, cada ingrediente al pulsarse salta un dialog con un detalle.

Debajo tenemos una zona de realización que explica detalladamente el proceso de realización de las recetas.

Justo debajo tenemos la sección de comentarios.

La valoración, como he comentado anteriormente se hace desde el icono del corazoncito que hay en la parte superior derecha de la vista.

La idea del dialog es una fila con imágenes que permitan seleccionarse a modo de RadioButton.



El contenido está dentro de un CustomScrollView lo que implica slivers.

La parte superior del diseño es un método aparte que se dibuja en un Stack que contiene en primer lugar el clipper que adorna la parte superior, que es un contenedor con una imagen de fondo.

También adentro del Stack tenemos el appBar y también el método buildAvatar:

```
Hero(
  tag: receta.id,
  child: ClipOval(
    child: FadeInImage(
      placeholder: AssetImage("assets/images/loading.gif"),
      image: _avatar),
  )),
```

La imagen del avatar está contenida por un Hero animation.

Justo debajo, en el interior de una columna tenemos la descripción de la receta que incluye :

- La dificultad: ----->  Media
  
- La valoración o calificación: ----->  2.0
  
- El nombre: -----> ***Lasana***
  
- La categoría: -----> Categoría:  
Pasteles
  
- La duración: ----->  55 minutos

```
Container(
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: <Widget>[
      _iconoDificultad(size, receta),
      _iconoCalificacion(size, receta),
    ],
  ),
  width: size.width,
  height: size.height * 0.07,
),
_nombreReceta(size, receta),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    _apartadoCategoria(size, receta),
    _iconoDuracion(size, receta),
  ],
),
```

Después debajo tenemos la fila de ingredientes que está compuesto por un método donde encotnramos un Row dentro de un contenedor que setea los margenes y llama a un método que contiene el constructor de los ingredientes items.

Se compone de un FutureBuilder que devuelve una lista de ingredientes y construye con un ListView.Builder.

Para llenar la lista de ingredientes utilizo un método que itera la lista de ingredientes que tiene la receta y compara los ids. En caso afirmativo lo guarda en una lista, la cual devuelve cuando ha acabado la iteración:

```
List<Ingrediente> getListaIngredientesReceta(
    List<Ingrediente> lista, Receta receta) {
    List<Ingrediente> listaReceta = new List<Ingrediente>();

    for (var x = 0; x < receta.ingredientes.length; x++) {
        for (var i = 0; i < lista.length; i++) {
            if (receta.ingredientes[x][0] == lista[i].id) {
                Ingrediente ingrediente = lista[i];
                ingrediente.cantidad = receta.ingredientes[x][1];
                listaReceta.add(ingrediente);
            }
        }
    }
    return listaReceta;
}
```

Esta lista se la pasamos al builder que construirá cada elemento con un Widget customizado que es la vista de cada uno de los elementos.



La clase se llama IngredientesHorizontal y se compone de un GestureDetector que hará saltar un alert al ser pulsado.

Se compone de una columna que contiene una imagen con un FadeInImage y un gif de placeholder, además debajo se puede encontrar la cantidad del ingrediente que se necesita para la receta.

El diálogo contiene una columna que tiene en primer lugar un row donde ponemos la equis en icono para salir de la vista (aunque también se puede salir pulsando fuera) y debajo en una fila tenemos la imagen metida dentro de un ClipOval que la recorta para hacerla circular y luego tenemos otra columna con Dos text metidos dentro los cuales son, uno el nombre y tipo del ingrediente y otro la cantidad.



Así quedaría el dialog de la lista de ingredientes.

Debajo de la parte de ingredientes encontramos el apartado de preparación que es una lista de párrafos listados con un fondo de imagen seteados con ayuda de un bucle:

```
List<Widget> _generarPasos(BuildContext context, Receta receta, Size size) {  
    List<Widget> listaInstrucciones = new List<Widget>();  
    listaInstrucciones.add(  
        SizedBox(height: size.height * 0.07),  
    );  
    for (var i = 0; i < receta.descripcion.length; i++) {  
        listaInstrucciones.add(_pasoPreparacion(size, i));  
    }  
    listaInstrucciones.add(  
        SizedBox(height: 20),  
    );  
    return listaInstrucciones;  
}
```

Vemos que a cada iteración se dibuja un widget llamado pasoPreparacion al que le paso un index para pintar el paso de la lista.

Esta es la imagen que usado de fondo en cada paso de la realización.



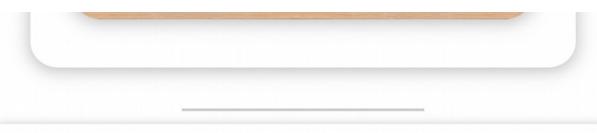
En la zona de la valoración, que se realiza pulsando sobre el ícono de valoración, tenemos un alertDialog que contiene una fila de 5 imágenes y debajo un TextFormField que recogerá el texto introducido y tratará de hacer el envío.



La función llamaría al create de valorados en el backend seteando el modelo y añadiéndolo a la base de datos.

Por último la zona de valoraciones se compone de una lista SliverList a la cual le pasamos con un FutureBuilder la lista de fvalorados y comparamos el id de receta que hay en el modelo de valorado con el id de la receta, de este modo añadimos la valoración.

La vista básica del comentario se compone de un avatar construido como los anteriores mencionados y en otra columna el correo del usuario que ha valorado y el comentario, además de la valoración con su ícono.



pedromolesp@gmail.com

Es que me parece increíble que pongan los animales así vivos y luego la receta, osea, esto va en contra de los derechos de los animales, voy a denunciarlos por opresores.



pedromolesp@gmail.com

Menuda receta, me encanta, es genial



pedromolesp@gmail.com

Esto no es comestible, llevo un mes en urgencias



#### 6.4.8. EL DRAWER

El drawer se invoca desde los scaffold, llamando a su atributo drawer o endDrawer (que sale de derecha a izquierda) y seteando un Widget tipo Drawer cuyo child en este caso es un ListView que permite poner varios elementos entre los cuales encontramos un drawer header, que será la parte superior del drawer:



```
ListView(
  children: <Widget>>[
    DrawerHeader(
      padding: EdgeInsets.all(0),
      child: Stack(
        children: <Widget>[
          Container(
            width: size.width,
            child: Image.asset(
              "assets/images/drawerheader.gif",
              fit: BoxFit.fill,
            ),
        ),
      ),
    ),
  ],
)
```



El elemento hijo es un stack porque necesitamos poner un icono para cerrarlo con una imagen sobre el fondo, un icono envuelto en un GestureDetector que hace un pop() con Navigator.

También he puesto una columna en la cual he insertado cada uno de los contenedores clickables (que disponen de un GestureDetector) y que nos permiten navegar a vistas.

Disponen de una imagen de fondo y un text con el texto deseado.

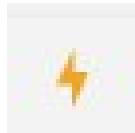


## 6.5. DEBUGGEANDO CON FLUTTER

El AVD de android studio nos permite crear dispositivos virtualizados y lanzarlos en nuestro ordenador para realizar pruebas de debugging con ellos. Son una opción notable, pero también existen los dispositivos que crea flutter.

Fundamentalmente son mucho más ligeros y más rápidos aunque ciertas funciones, como escribir con el teclado, están desactivadas, así que si la potencia del equipo del usuario que va a hacer debugging lo permite, sería aconsejable usar un dispositivo del avd de android studio.

Uno de los atractivos más notables de flutter que están comenzando a integrar otros entornos, es el just in time de su hot reload que permite visualizar los cambios en cuestión de milisegundos. Con solo guardar, mientras la app está lanzada en un dispositivo debuggeando se pueden visualizar los cambios, o también pulsando sobre el icono del hot reload. Puede tardar de 50 milisegundos, a 400 normalmente.



El hot reload es estupendo en lo que a velocidad respecta, pero también tiene algunos peros, como que a veces no lanza los últimos cambios si estos tienen que ver con assets o peticiones de datos externas. Por ejemplo: si yo cambiara la dirección de la api en mi api provider, tendría que utilizar hot restart.

Para reparar las faltas que tiene el hot reload, el hot restart compila la aplicación y la lanza haciendo un build del apk e instalándola si no lo está. Suele tardar en torno a 3s aunque depende el equipo desde el que se ejecuta y la magnitud de la aplicación. A veces puede llegar a tardar 15s.



A veces cuando importamos nuevos assets tenemos que hacer un stop and play y volver a lanzar la aplicación por eso la paramos con el botón de stop.

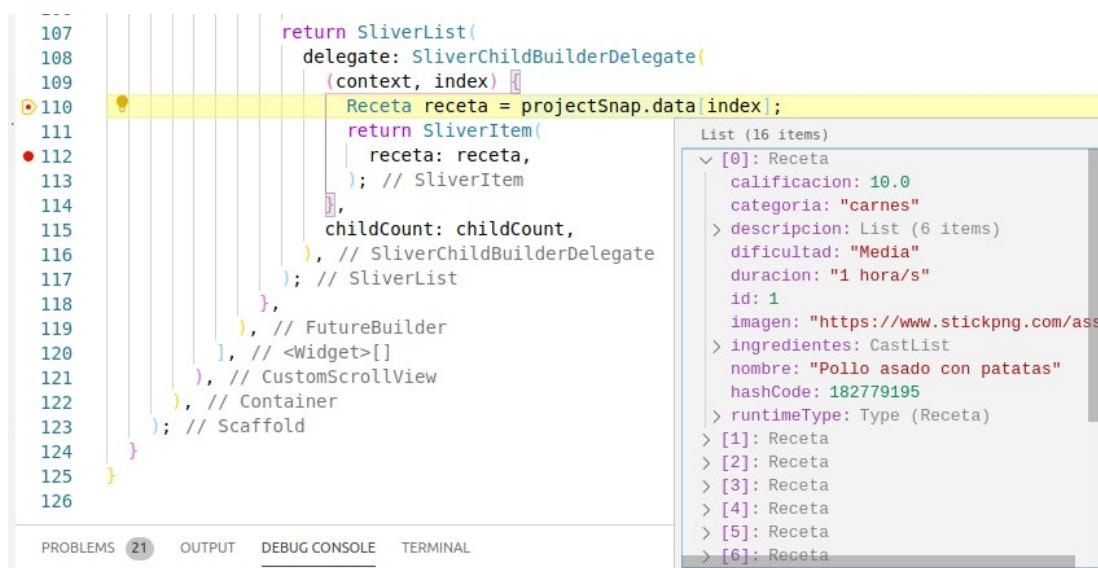


Otra forma de hacer debugging es con la función de dart: `print("valor");` Esta se podría decir que es un clásico pero también un quebradero de cabeza porque no es muy ágil, además entornos como visual studio code nos brindan funcionalidades geniales para depurar código:

- Los breakpoints, o puntos de interrupción, que paran el código en el punto indicado cuando se está ejecutando justo en esa linea. Nos permiten ver los valores de las variables en la linea y ver si los valores están siendo erróneos o diferentes de los que queremos conseguir:

```
247
248 List<Widget> _generarPasos(BuildContext context, Receta receta, Size size) {
249   List<Widget> listaIntrucciones = new List<Widget>();
250   listaIntrucciones.add(
251     | SizedBox(height: size.height * 0.07),
252   );
253   for (var i = 0; i < receta.descripcion.length; i++) {
254     listaIntrucciones.add(_pasoPreparacion(size, i));
255   }
256   listaIntrucciones.add(
257     | SizedBox(height: 20),
258   );
259   return listaIntrucciones;
260 }
```

En la imagen superior podemos ver como el programa ha sido parado en ese punto y nos permite visualizar el valor de una variable, ya sea de un solo campo como esta, u objetos completos:



También disponemos, por supuesto, de la parte de debug en los menús inferiores, que es dónde dispara los errores y los stack cuando algo falla.

```
PROBLEMS (21) OUTPUT DEBUG CONSOLE TERMINAL
Restarted application in 10.658ms.
index
Error: <timed out>
childCount
': error: org-dartlang-debug:synthetic_debug_expression:1:1: Error: The getter 'childCount' isn't defined for the class'.
- 'ListadoRecetasPage' is from 'package:guidefood/src/vista/pages/listado_recetas.dart' ('lib/src/vista/pages/listado_recetas.dart').
Try correcting the name to the name of an existing getter, or defining a getter or field named 'childCount'.
childCount
^^^^^^^^^
>
```

## 6.6. TESTING BÁSICO DE MODELOS

Para realizar un test básico de modelo en flutter nos vamos a la carpeta de test, en la ruta raíz del proyecto. Voy a realizar un test del modelo de ingredientes, para ello voy a crear un método que compare un objeto tipo Ingrediente, con un json que le vamos a pasar a través del método definido en el modelo para formatear json a Ingrediente, que es fromJsonMap:

```
class Ingrediente extends Equatable {
    int id;
    String nombre;
    String tipo;
    String imagen;
    String medida;
    int cantidad;

    Ingrediente(
        {@required this.id,
        @required this.nombre,
        @required this.tipo,
        @required this.imagen,
        @required this.medida,
        this.cantidad})
        : super();

    Ingrediente.fromJsonMap(Map<String, dynamic> json) {
        id = json['id'];
        nombre = json['nombre'];
        tipo = json['tipo'];
        imagen = json['imagen'];
        cantidad = json['cantidad'];
        medida = json['medida'];
    }
}
```

Después es necesario definir un main dónde se va a correr. Cada test que pasemos debe estar en un main, a poder ser independiente para poder localizar el test en caso de que falle.

```
import 'dart:convert';

import 'package:flutter_test/flutter_test.dart';
import 'package:guidefood/src/models/ingredient.dart';

import '../fake/ingrediente_fake.dart';
import '../fixtures/fixture_reader.dart';

void main() {
    Run | Debug
    test('Ingrediente fromJson should return a valid model', () async {
        final Map<String, dynamic> jsonMap =
            json.decode(fixture('ingrediente.json'));

        final result = Ingrediente.fromJsonMap(jsonMap);

        expect(result, ingredient1);
    });
}
```

Se puede apreciar que realmente se está haciendo una petición al json a través de un método fixture que lo que hace es localizar un archivo pasándole el nombre del archivo en su carpeta:

```
import 'dart:io';

String fixture(String name) =>
    File('../test/fixtures/$name').readAsStringSync();
```

Esta petición devuelve el contenido del json a un objeto result que es comparado con un objeto que hemos creado en ingredientes\_fake.dart, que nos permitirá comparar el contenido de ambas instancias:

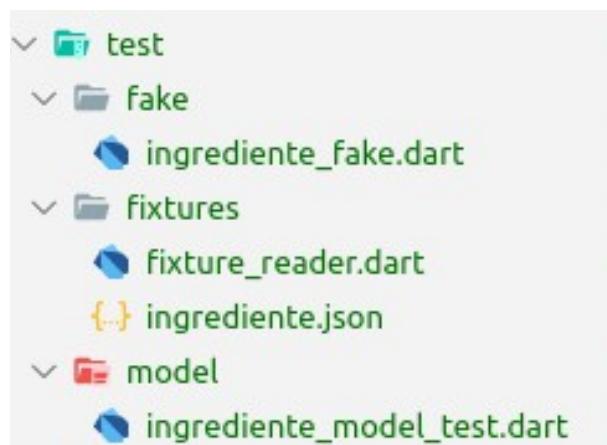
```
import 'package:guidefood/src/models/ingrediente.dart';

var ingrediente1 = Ingrediente(
    id: 66,
    nombre: "Mermelada",
    tipo: "De fresa",
    imagen:
        "https://static.wixstatic.com/media/c92466_410438a94cf0492dbacae96ec9e7a98a~mv2.png/v1/fit/w_500,h_500,q_90/file.png",
    medida: "g");
```

que se compara en el método expect(); del main con el objeto del ingrediente.json:

```
    "id": 66,
    "nombre": "Mermelada",
    "tipo": "De fresa",
    "imagen": "https://www.thermorecetas.com/wp-content/uploads/2016/07/Empanadillas-de-jam%C3%B3n-y-queso.jpg",
    "medida": "g"
```

La estructura de directorios que nos ha quedado es:



Ahora solo tendríamos que lanzar flutter test y (si la clase modelo no extiende de equatable)...

BOOOOOOM, error.

```
pedro-trabajo@pedrotrabajo:~/projects-flutter/guidefood$ flutter test
00:02 +0 -1: Ingrediente fromJson should return a valid model [E]
  Expected: <Instance of 'Ingrediente'>
  Actual: <Instance of 'Ingrediente'>

    package:test_api                                expect
    package:flutter_test/src/widget_tester.dart 229:3  expect
    model/ingrediente_model_test.dart 16:5           main.<fn>

  00:02 +0 -1: Some tests failed.
pedro-trabajo@pedrotrabajo:~/projects-flutter/guidefood$ 
```

¿Por qué?, ¿qué ocurre? Si hemos comparado dos instancias con el mismo contenido... Sí, pero cada instancia es de su padre y de su madre, con el mismo contenido, eso sí. Esto ocurre porque “==” devuelve una comparativa de instancias, entonces ¿Cómo hacemos para sobreescribir el == y hashCode para que devuelva el contenido y no las instancias?, la respuesta es Equatable, el cual está implementado en la primera imagen del modelo de ingrediente.

Para obtener equatable solo tenemos que ir a <https://pub.dev/packages/equatable> para ver la lista de versiones y desde ahí integrar en nuestro pubspec.yaml una linea:

equatable: ^0.5.0

```
pedro-trabajo@pedrotrabajo:~/projects-flutter/guidefood$ flutter test
00:02 +1: All tests passed!
pedro-trabajo@pedrotrabajo:~/projects-flutter/guidefood$ 
```

Ahora sí.

## 6.7. LIBRERÍAS

Dentro del pubspec tenemos la zona de dependencias donde podemos escribir la importación de packages oportuna, eso sí, correctamente alineados.

En mi caso he necesitado los packages de firebase para la autenticación y google para logear con las cuentas de google.

También está la importación del equatable, explicada en el apartado anterior.

Http se encuentra más arriba, gracias a la cual realizamos peticiones al backend de datos.

```
version: 1.0.0+1

environment:
  sdk: ">=2.2.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2

  # The following adds the Cu
  # Use with the CupertinoIcon
  cupertino_icons: ^0.1.2
  firebase_auth: ^0.15.1
  firebase_analytics: ^5.0.6
  firebase_core: ^0.4.2+1
  google_sign_in: ^4.0.14
  equatable: ^0.5.0
```

## 6.8. UTILIZACIÓN DE RECURSOS

Recursos, si queremos un proyecto con vistas completas, imágenes, fuentes específicas o iconos, lo fundamental es saber como añadirlos a nuestro proyecto. En flutter la complejidad para realizar tales cosas no es muy alta.

### - Imagenes:

Debemos buscar la linea de los assets, que normalmente al crear un proyecto nuevo, estará comentada, y deberemos añadir con la identación adecuada dentro las rutas de las carpetas que contengan nuestros assets:

```
flutter:

# The following line ensures that the M
# included with your application, so th
# the material Icons class.
uses-material-design: true

# To add assets to your application, ad
assets:
- assets/images/
- assets/iconos/
```

### - Fuentes:

Al igual que con las imágenes, existe ya una linea comentada para las fuentes que puede usarse descomentándola o servirnos de guía. Deberemos tener las fuentes en sus archivos .ttf en una carpeta del proyecto a la que hacer referencia, en mi caso es la carpeta fuentes dentro de assets.

```
fonts:
- family: Golden-Hills
  fonts:
    - asset: assets/fuentes/GoldenHillsDEMO.ttf
- family: Simpsom
  fonts:
    - asset: assets/fuentes/PWYummyDonuts.ttf
```

Para utilizar una fuente dentro de un widget tipo Text():

fontFamily: "Montserrat-Medium"

```
child: Text(
  receta.nombre,
  style: nombreDetalle,
  textAlign: TextAlign.center,
  overflow: TextOverflow.ellipsis,
  maxLines: 2,
), // Text
```

## 7. MANUAL DE USO

### PANTALLA SPLASH

Splash, pantalla inicial con temporizador de 3 segundos que pasa automáticamente al listado. No hay interacción posible. La imagen es un gif y el fondo tiene un gradiente sobre el cual hay un texto.



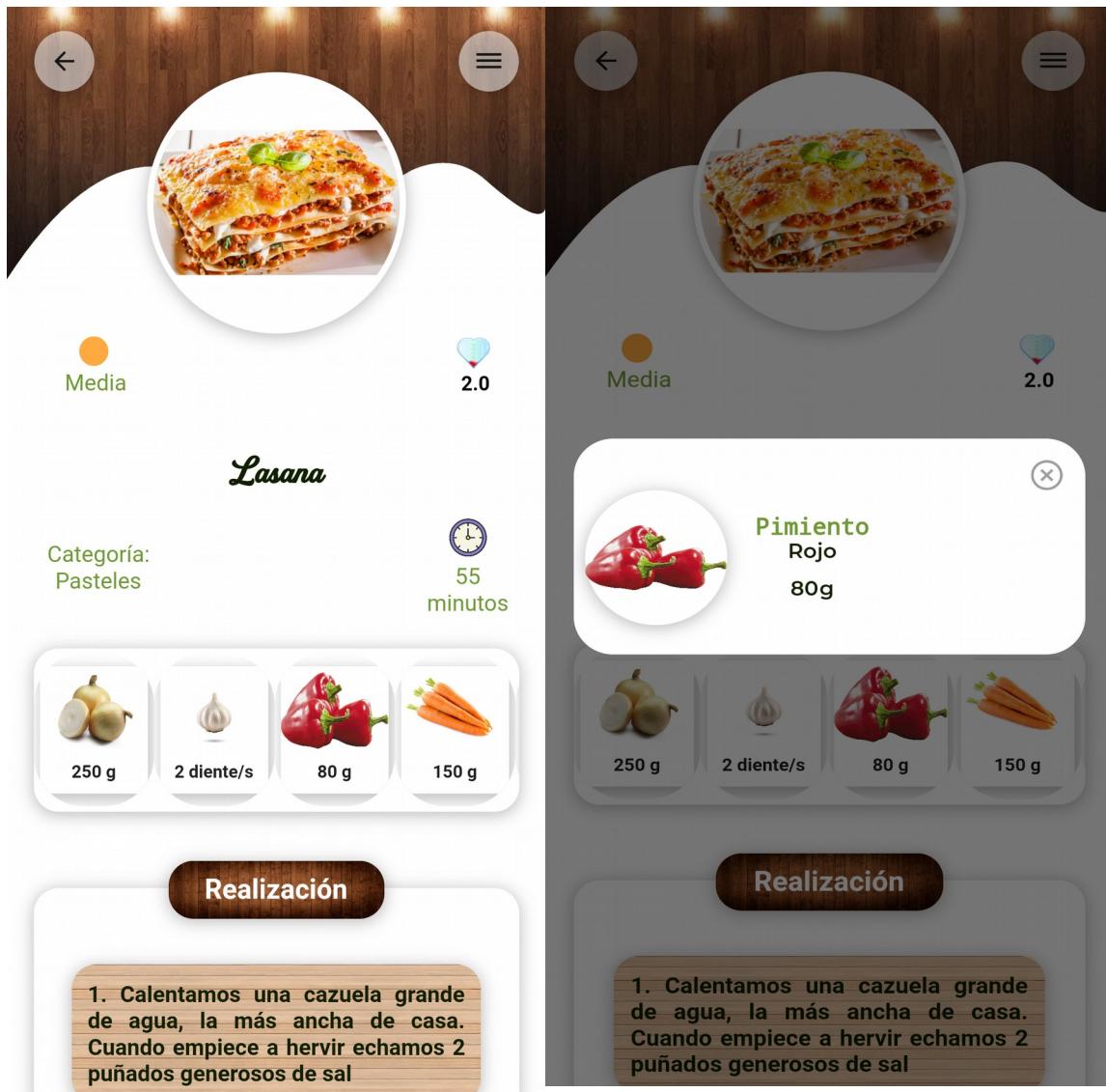
**PANTALLA DEL LISTADO DE RECETAS**

Permite hacer scroll para ver una lista de recetas que pueden ser pulsadas y te conducen a un detalle de las mismas.



PANTALLA DEL DETALLE DE LAS RECETAS

La pantalla de detalle ofrece una lista de ingredientes de los cuales se compone la receta y sus cantidades específicas, además ofrece un detalle también al clickar en alguno de ellos en la lista móvil horizontal.



A parte del detalle de los ingredientes, podemos acceder al menú de valoración de la receta pulsando sobre el ícono del corazoncito con la puntuación.

Más abajo hay una tabla con los pasos para la realización de la receta.

**Realización**

1. Lavamos las patatas y las ponemos a cocer junto con las zanahorias peladas, durante 25 minutos.
2. Introducimos los huevos y un poco de sal, dejandolo cocerse durante 10 minutos más.
3. Dejamos que se temple y pelamos y picamos las patatas y los huevos en daditos.
4. Cortamos las aceitunas por la mitad.
5. Ponemos las patatas, las zanahorias, el huevo y las aceitunas en un bol grande, agregamos los guisantes y el atún.
6. Elaboramos la mayonesa casera y la mezclamos con los demás alimentos. Podemos utilizar mayonesa comprada.
7. Añadimos sal al gusto.

La pantalla de comentarios está justo al acabar la paleta de paso para la realización, está solo orientada a visualización, aunque má adelante añadiré la funcionalidad de poder borrar los comentarios propios.

---

pedromolesp@gmail.com



Es que me parece increíble que pongan los animales así vivos y luego la receta, osea, esto va en contra de los derechos de los animales, voy a denunciarlos por opresores.



---

pedromolesp@gmail.com



Menuda receta, me encanta, es genial



---

pedromolesp@gmail.com



Esto no es comestible, llevo un mes en urgencias

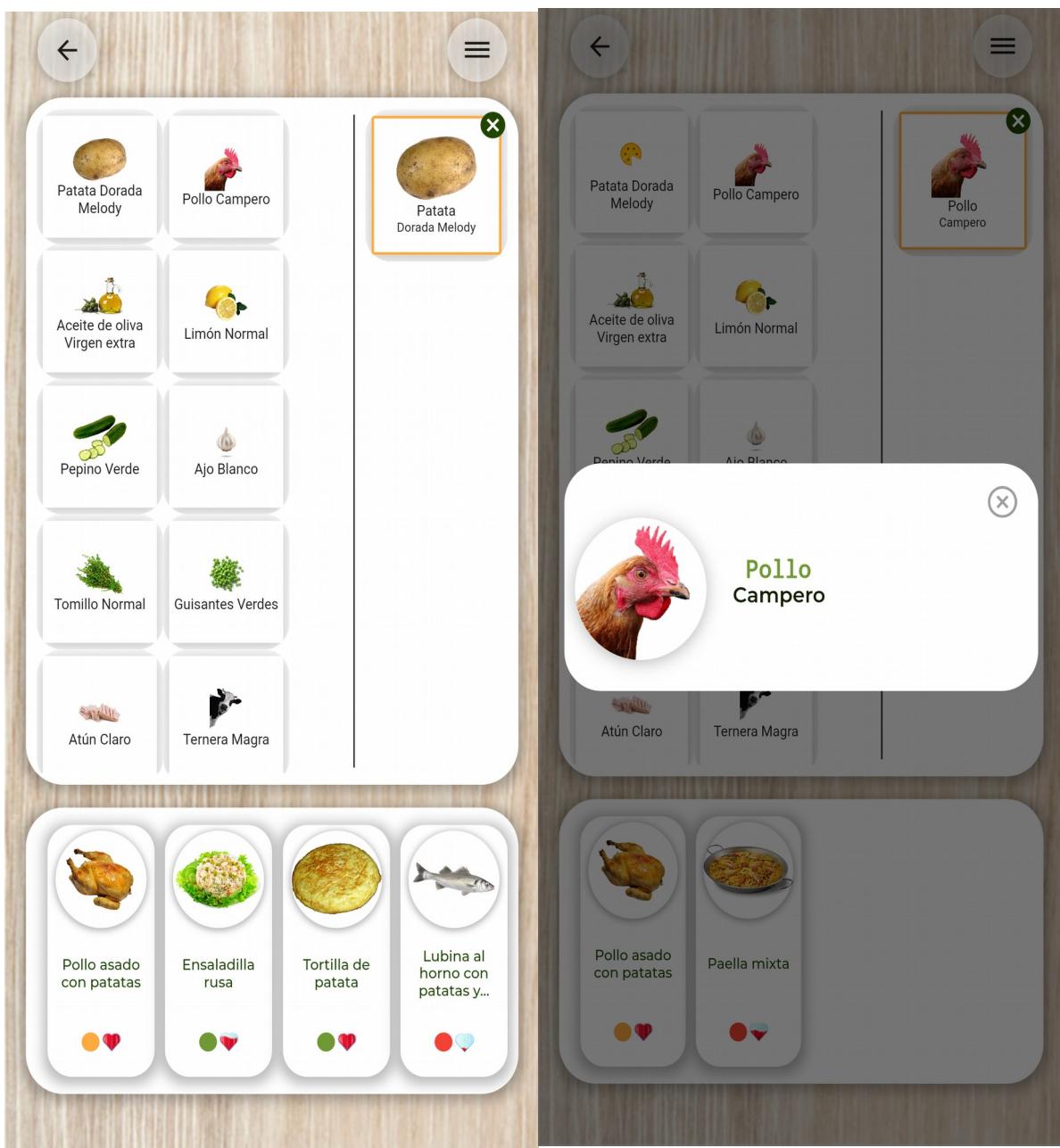


**PANTALLA DEL SELECTOR DE RECETAS**

En esta pantalla podemos arrastrar elementos de la lista de la izquierda a la de la derecha para conseguir debajo una fila de recetas posibles recetas.

Las recetas son pulsables y te llevan al detalle, y los ingredientes de la file de la derecha también lo son, y ofrecen al usuario una vista del detalle del ingrtediente.

Si se pulsa sobre el icono de la equis de la lista de la derecha, el ingrediente se elimina.



**DRAWER**

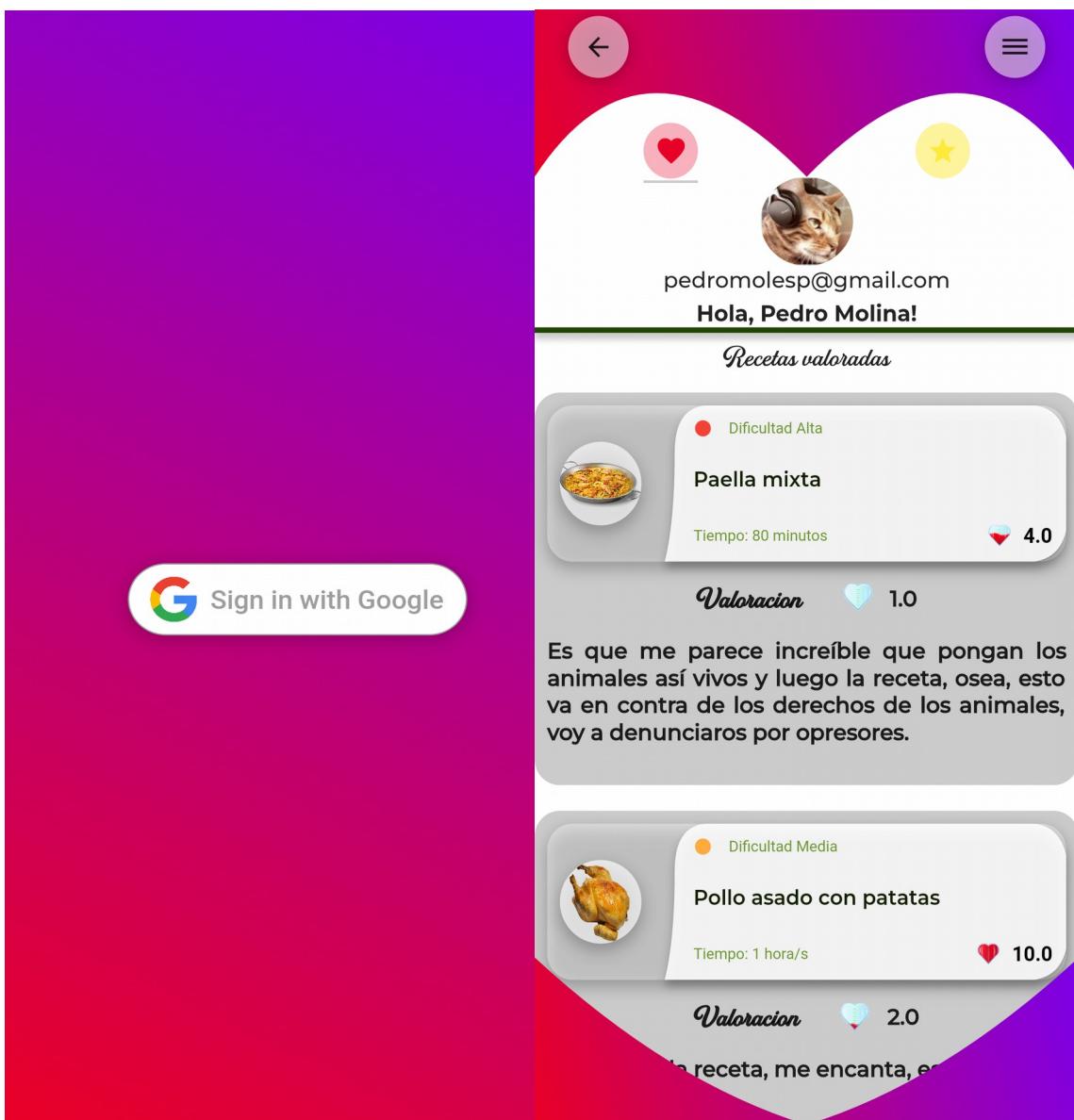
El drawer permite navegar por las vistas del selector, la lista de recetas y la zona de favoritos y valorados.



### PANTALLA DE FAVORITOS

En la pantalla de login accedemos con la cuenta de google y entramos en una vista de favoritos y y valorados.

En esta pantalla no hay interacciones de funcionalidad más allá que consulta de información, aunque más adelante sería interesante implementar el desmarcar favoritos o borrar valoraciones.



Y por último la pantalla de favoritos:



## 8. BIBLIOGRAFÍA Y WEBS

<https://app.genmymodel.com/> es una página muy útil para realizar diagramas de flujo y de clases.

<https://stackoverflow.com/users/11197232/pedro-molina> la biblia de todo programador, es el sitio donde más respuestas he encontrado a los errores, fallos y excepciones durante el desarrollo.

Código limpio de Uncle Bob, es un libro que habla sobre los principios del cuidado, desarrollo y correcto mantenimiento del código, que me ha servido al inicio sobretodo para poner nombres más afines a las acciones que realizan métodos y variables.

<https://firebase.google.com/> firebase, es un repositorio multifunción que he utilizado para guardar los datos de autenticación con la función de login con google sign in.

<https://flutter.dev/> aquí he obtenido información de flutter, así como ejemplos (esta es la documentación oficial).

<https://medium.com/> Medium cuenta con numerosos post y guías sobre desarrollo con flutter y me ha sacado de más de un apuro, por ejemplo de la implementación de la autenticación.

<https://pub.dev/packages/> esta es la página oficial de dependencias para flutter dónde podemos encontrar librerías de la comunidad y librerías oficiales.

<https://www.udemy.com/course/flutter-ios-android-fernando-herrera/> cursos de udemy sobre flutter como este me han ayudado a hacerme rápido con los conocimientos necesarios para llevar a cabo el desarrollo del proyecto.

<https://gitlab.com/pedromolina/guidefood.git> En git lab he subido el despliegue del código y las versiones del mismo.

## 9. CONCLUSIONES

- **Rápido aprendizaje:** Durante estos dos meses he aprendido a generar diseños en flutter de manera bastante rápida. He aprendido sobretodo a tener paciencia cuando algo no funciona y a buscar información, a leer foros y apis y código de terceros, lo cual me ha servido para replicar algunas estructuras que he reciclado para mi proyecto.
- **Superación:** Ha sido muy interesante conseguir desarrollar un proyecto completo a excepción de ciertas funcionalidades que por falta de tiempo me han refrenado.
- **Los idiomas:** Me he dado cuenta de la importancia del inglés para entender foros, apis y documentación, por suerte me manejo bastante bien a pesar de tener medio b2 de inglés.
- **Veloz pero áspero:** Flutter es una herramienta a la que le falta pulirse y mejorar su gama de funcionalidades, pero creo que en cuestión de unos meses, con el hype que tiene y las comunidad tan enorme que lo respalda, pronto veremos una tecnología pulida, mucho más funcional en cuanto a posibilidades e integración con otras funcionalidades, que otras de las tecnologías predominantes.
- **Todo es mejorable:** Este proyecto tiene muchas mejoras posibles para hacerlo más profesional, aunque mi intención primaria era definir diferentes tipos de vistas e implementar en ellas utilidades diferentes para ver el abanico de posibilidades que brinda flutter, por ello alguna parte de mi aplicación puede chirriar con respecto a otra en lo que a diseño respecta. De este modo creo que me vendría bien aplicar lo último que he aprendido en cuanto a refactorización y código limpio para dejar una estructura de código más limpia, estructurada y concisa.
- **Anarquía:** En flutter no hay reglas escritas. En lo que respecta a la separación de widgets en métodos hay múltiples discusiones acerca de lo mejor a nivel de rendimiento y a nivel de diseño de código, ya que excesivas llamadas a métodos acaban ralentizando y siendo flojo en lo que respecta a la velocidad. Yo personalmente, tras estos tres meses trabajando con flutter, creo que lo más óptimo y limpio es separar los componentes que más resaltan en la vista en métodos aparte, pero tampoco hay que llevarlo tan al límite de crear tres métodos para dibujar un solo contenedor, a no ser que se traten de vistas que dependan de otro factores, como de variables.
- **El siguiente paso:** Creo que necesito aprender sobre hilos para explotar aún más las cualidades de flutter aunque el siguiente paso que quiero seguir, porque es realmente lo que siempre me ha gustado, es aprender sobre animaciones.