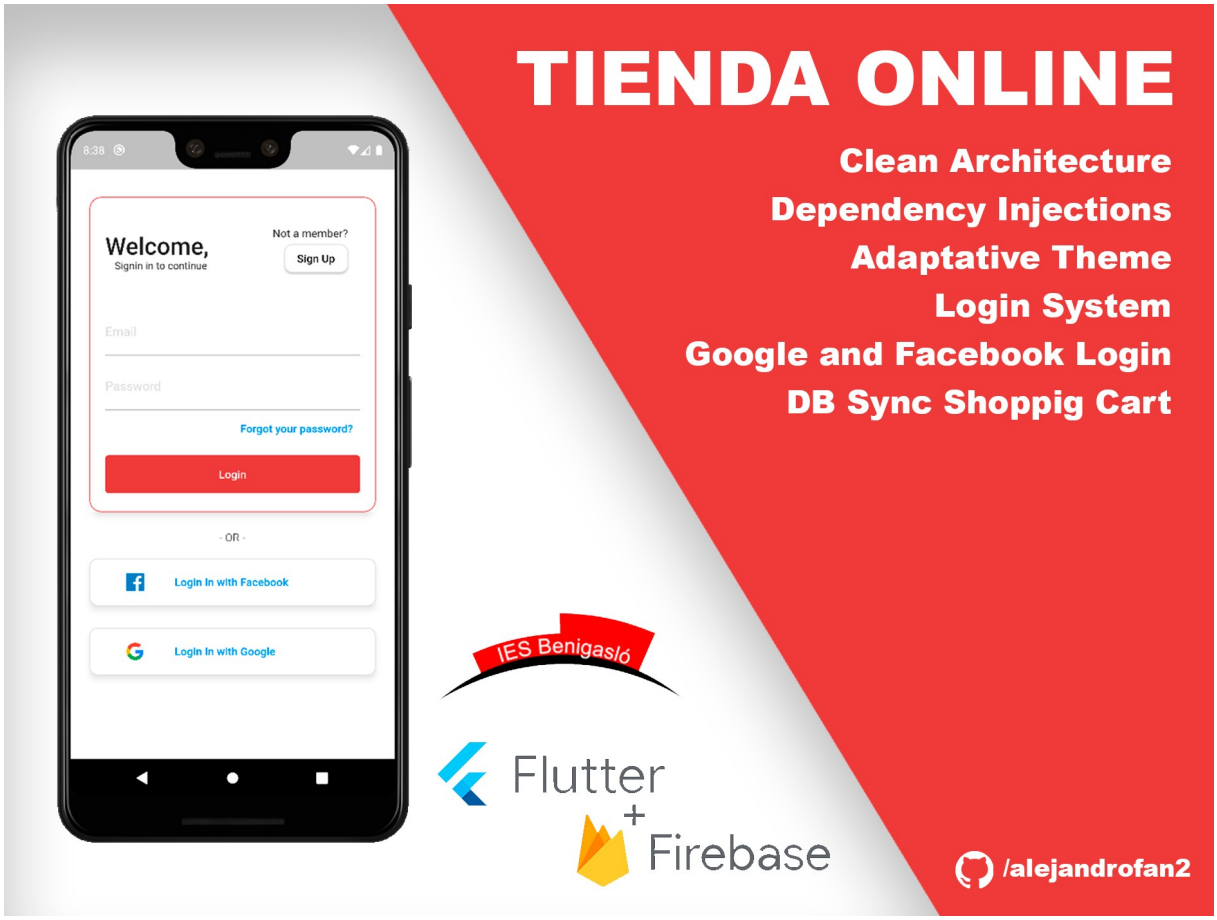

PROYECTO FINAL DE CICLO
DESARROLLO DE APLICACIONES MULTIPLATAFORMA

CURSO 2021-22

TITULO: TIENDA ONLINE

ALUMNO: ALEJANDRO FANDOS FERRADA

TUTOR: NEREIDA SEVILLA TRUJILLO

Índice de Contenidos

Prólogo

Este proyecto surgió cuando conocí a un grupo de aficionados al motor que me propusieron la creación de una tienda ‘Online’ para la venta de todo tipo de productos del mundo del motor y similar.

Un día como cualquier otro, estábamos debatiendo sobre cuál era la mejor página de venta de piezas de coches, aquel día me di cuenta de que las webs en su mayoría estaban desfasadas y basadas en tecnologías anticuadas.

Ademas de ello se ha pretendido dar a conocer las posibilidades que Brinda Flutter como ‘ToolKit’ para la creación de IU, y Dart como lenguaje base siendo estructurado, orientado a objetos, con herencia simple, interfaces…

En un futuro cercano se liberara una versión base del código para el uso y aprendizaje de toda la comunidad educativa.

Información sobre el autor

Alejandro Fandos Ferrada

- E-mail: alejandrofan2@gmail.com
- Estudiante del IES Benigasló en el 2º curso de Desarrollo de aplicaciones multiplataforma.
- Telegram → @alejandrofan2
- Instagram → @alejandrofan2
- Mas información: alejandrofandos.es

Resumen

Este trabajo explica como crear una aplicación multiplataforma (IOS, Android) en el Lenguaje de programación Dart usando Flutter como 'ToolKit' y FireBase como Base de Datos.

Durante la demo solo estará disponible la versión de Android por cuestiones de compatibilidad con el sistema operativo de desarrollo. Sin embargo gracias a Flutter el proceso de exportación a IOS será muy sencillo.

El presente Trabajo Final de Grado recoge el proceso de gestión de un proyecto de desarrollo, desde el momento que surge la idea, pasando por la definición de los requerimientos y completando con un seguimiento sobre el proceso. Todo ello se ha estructurado en 4 capítulos.

En el primero, de introducción, se explica la elección de tecnologías. Se dan las Claves de 'Clean Architecture for Flutter' y se especifican los requisitos previos.

En el segundo capitulo estudiaremos la fase de diseño del proyecto explicando a su vez los objetivos del mismo. Para el tercero haremos un seguimiento completo del proyecto teniendo en cuento los riesgos, recursos, temporalización y seguimiento del mismo.

Para acabar tendremos un capitulo de conclusiones en el que se explica el grado de satisfacción y asimilación. Y posibles propuestas de mejora.

Por ultimo se adjuntara la bibliografía y los anexos al final del documento.

Abstract

This work explains how to create a multiplatform application (IOS, Android) in the Dart Programming Language using Flutter as a 'ToolKit' and FireBase as a Database.

During the demo, only the Android version will be available for compatibility issues with the development operating system. However, thanks to Flutter, the export process to IOS will be very simple.

This Final Degree Project includes the management process of a development project, from the moment the idea arises, passing through the definition of the requirements and completing with a follow-up on the process. All this has been structured in 4 chapters.

In the first, introduction, the choice of technologies is explained. The Keys of ‘Clean Architecture for Flutter’ are given and the prerequisites are specified.

In the second chapter we will study the design phase of the project explaining in turn its objectives. For the third, we will do a complete follow-up of the project taking into account the risks, resources, timing and follow-up of the same.

To finish we will have a chapter of conclusions in which the degree of satisfaction and assimilation is explained. And possible suggestions for improvement.

Finally, the bibliography and annexes will be attached at the end of the document.

Índice de imágenes

1. Introducción

Este trabajo explica como crear una aplicación multiplataforma (IOS, Android) en el Lenguaje de programación Dart usando Flutter como 'ToolKit' y FireBase como Base de Datos.

Ilustración 1: Logotipos de Flutter SDK y Firebase software de Google

Como desarrollador me surgió la necesidad de hacer un proyecto para el grado que estoy estudiando y pensé "Las Tiendas Online están funcionando muy bien y creo que no hay muchos desarrolladores haciendo tiendas de coches". Ese fue el momento en el que decidí hacer este proyecto.

El motivo de Flutter y FireBase no es otro que la gran capacidad que tienen de forma conjunta que otras tecnologías no te brindan. Gracias a ellas puedes crear aplicaciones completamente nativas en Android, IOS, Web, Windows y todo ello sin depender de un backend. Además de eso FireBase también tiene características que me gustan: contenido offline, sincronización de datos en tiempo real, carga de velocidad súper rápida y todo esto de forma gratuita (al menos para proyectos pequeños) sin mantenimiento del servidor.

Por ultimo Flutter tiene funciones muy útiles como:

1. Hot reload

   El fabuloso sistema de Hot reload permite enviar cambios a la app mientras está en ejecución, salta un disparador y se actualiza la interfaz. Cabe destacar que cambios en constructores o objetos que se forman en pre-ejecución no se mostraran y se deberá recompilar la app.

2. Widgets

   “Otro grandísimo aliado que tenemos en Flutter són esto, los Widget. Aunque parezca mentira, han conseguido que programar una interfaz visual fuera del estándar HTML sea una verdadera gozada.” - Aitor, como-progrmar.net

3. Soporte Oficial

   Aparte de la gran comunidad que tiene detrás, hay un equipo de más de 1000 desarrolladores de Google dando soporte.

Me va a permitir experimentar de forma profesional el desarrollo en Flutter teniendo en cuenta muchos aspectos que cuando estas estudiando no indagas, por ejemplo desplegar nuevas versiones a un repositorio para que cualquiera pueda acceder., crear ‘run tests’ y otros aspectos que creo que son casi tan importantes como programar.

2. Diseño

2.1. Descripción del proyecto

Una Tienda Online / e-Commerce es un software completo que permite que un proveedor de productos pueda venderlos de forma sencilla ante los clientes finales. Todo ello implica el uso de distintos sistemas en la aplicación.

Dart es un lenguaje de programación de código abierto basado en C# y JavaScript desarrollado por Google. Que surge como alternativa a algunos de los problemas que tiene JavaScript. Funciona con una Maquina Virtual (MV) que corre de forma nativa en todos los navegadores.

Flutter es un Software Development Kit (SDK) de código abierto desarrollado por Google para la creación de IU para móviles. Aunque también permite el desarrollo web.

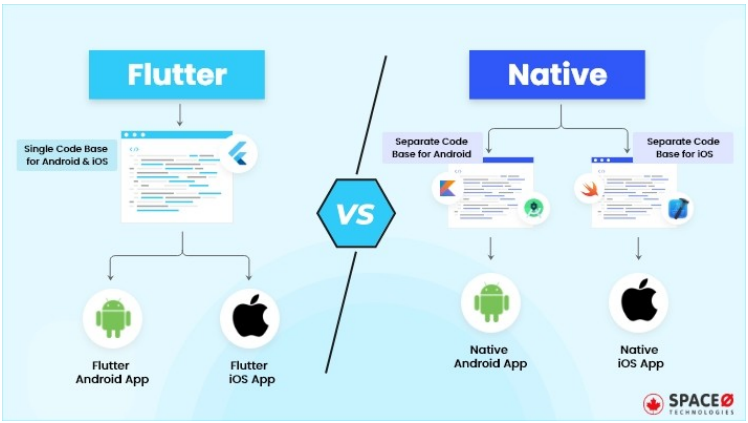

Ilustración 2: Comparativa de como se transpila el código en Flutter/React

FireBase es una plataforma para el desarrollo de aplicaciones tanto web como móvil adquirida por Google en 2011. Es interesante porque te permite crear y sincronizar aplicaciones con una base de datos sin necesidad de un 'backend server'.

La elección de estas herramientas está basada en el buen funcionamiento y la sinergia y facilidad de enlace que tienen debido a ser las tres de Google. Las tres disponen de una amplia comunidad y son de las más eficientes del mercado.

2.1.1. Objetivos

- ✗ Crear una aplicación multiplataforma
- ✗ Que la aplicación tenga sistema de autenticación y de permisos (AdminPanel)
- ✗ Que muestre productos y categorías
- ✗ Que permita editar los productos por los administradores, así como añadir productos, editar productos, consultar pedidos, ver datos y estadísticas de uso.
- ✗ Tener un sistema avanzado de carrito de compra con sincronización en la nube.
- ✗ Crear una IU adaptable siguiendo la filosofía de 'Material Design'.
- ✗ Crear un sistema de Tema adaptativo (Dark/Light)

- ✗ Crear un sistema de selección de coche para poder filtrar las búsquedas unicamente con piezas que sean válidas.
- ✗ Crear un sistema de búsqueda por tags.

2.2. Definición de tareas

(0) Prerrequisitos previos e instalación de dependencias.

- (1) Instalación de Java, Flutter, Dart.
- (2) Configuraciones varias de VS Code y GitHub repo.
- (3) Instalación y configuración de dependencias.

(1) Re-factorización de directorios e implantación de ‘Clean Architecture’.

- (1) Re-factorización directorios.
- (2) Eliminación de código residual de ‘flutter create’.

(2) Creación de la IU.

- (1) Creación sistema de páginas y navegación.
- (2) Diseño de las páginas del sistema de autenticación.
- (3) Diseño del resto de paginas (‘Home’, ‘Cart’, ‘Profile’).

(3) Enlace con FireBase.

- (1) Creación del proyecto en FireBase y configuración básica.
- (2) Creación del Login Service para la autenticación.

(4) Productos y Categorías

- (1) Creación de entidades para la asimilación de los datos en Flutter.
- (2) Creación de JSON para Productos / Categorías.
- (3) Creación de ListViews para mostrar el contenido.

(5) Perfil personal

(1) Creación de IU de las páginas de cambio de nombre, correo, teléfono, contraseña y avatar.

(2) Creación de sistema de cambio de nombre, correo, teléfono, contraseña y avatar.

(6) Carrito Avanzado.

(1) Creación de un sistema de carrito local funcional.

(2) Implementación de ‘FireBaseStorage’ en el carrito.

(7) Sistema de búsqueda por tags.

(1) Creación de un sistema de tags para los productos y categorías.

(2) Implementación del sistema de productos y tags en ‘FireBaseStorage’

(3) Creación de un sistema de búsqueda por tags.

(4) Creación de IU para la búsqueda.

(8) Sistema de elección de coche.

(1) Creación de un sistema de selección de coche.

(2) Implementación del sistema en ‘FireBaseStorage’.

(3) Creación de IU para la selección de coche.

(4) Implementar la selección de coche en el sistema de Búsqueda.

3. Seguimiento

3.1. Riesgos

La abstracticidad del proyecto en si és un riesgo, la gran mayoría de mis compañeros se han limitado a realizar una documentación de las tareas realizadas en la empresa. Mi iniciativa personal de querer desarrollar una Tienda Online causa que en lo práctico tengo que dedicar las 8h de trabajo diarias en otra tarea y realizar esta solo en espacios en los que no tengo asignado nada o días no laborables.

Como tal no existe ningún cliente yo mismo pongo mis expectativas y eso puede causar que dedique mucho tiempo a cosas que simplemente se pueden aplazar o no son tan importantes.

Por ultimo, en mi familia se vive una situación peculiar lo cual ha derivado ya en varias mudanzas durante el tiempo que he cursado el grado, seria una posibilidad que hubiera otra de estas situaciones que derivara en otra mudanza y toda la perdida de tiempo que eso conlleva.

3.2. Recursos

En este proyecto solo me encuentro yo (Alejandro Fandos Ferrada) trabajando y el material que utilizo es mi portátil Dell y los servicios de Google tanto 'Google Cloud platform' (GCP) para FireBase como para Flutter/Dart.

El mayor proveedor de conocimientos de nuevo vuelve a ser Google pero esta vez a través de su plataforma YouTube, me siento muy familiarizado con los videos, ademas soy capaz de absorber mucha información a través de ellos. He consumido también blogs o paginas como StackOverflow o Reddit

El presupuesto previsto para este proyecto es de 0€

3.3. Temporalización

3.4. Seguimiento

3.4.1. Fase 0 – Prerrequisitos previos e instalación de dependencias.

1. Instalación de Java, Flutter, Dart.

Para empezar adjunto los componentes del ordenador usado para 'code & debug':

- Modelo → Dell Inspiron 5415
- OS → Windows 11 Insider
- CPU → AMD Ryzen 5 5500U
- RAM → 24GB DDR5

Como editor uso Visual Studio Code debido a su ligereza y la versatilidad que da en función de las extensiones. Cabe destacar que tiene integración completa con los emuladores de Android Studio, Android ADB y Google Chrome.

Procedemos a la instalación de Java versión 12 a través de el instalador de Windows. Seguimos instalando el Software Development Kit (SDK) de Flutter, que también instala como dependencia Dart.

- Cabe recalcar que se instala Java ya que a la hora de hacer debugging Flutter transpila el código a Java para ejecutarlo en Android.

Por último desde una terminal situada en el espacio de trabajo, se ejecutan los siguientes comandos:

- flutter create .
- code .

2. Configuraciones varias de VS Code y GitHub repo.

(Se está teniendo en cuenta que tanto VS Code como Git ya están instalados en el sistema)

En este momento ya tenemos casi todo listo para empezar. Vamos a requerir de una serie de extensiones, unas más importantes y otras menos, que nos van a ayudar a que todo funcione correctamente.

Extensiones requeridas:

- Flutter
- Awesome Flutter Snippets
- Dart
- Error Lens
- YAML

Una vez está ya todo funcionando correctamente iniciamos el repositorio con el comando ‘git init’.

3. Instalación y configuración de dependencias.

Entramos en el archivo ‘pubspec.yaml’ y anticipándonos a las necesidades del proyecto procedemos a instalar las siguientes dependencias:

- firebase_core: ^1.12.0
- firebase_auth: ^3.3.7

- cloud_firestore: ^3.1.8
- provider: ^6.0.2
- google_sign_in: ^5.3.1
- flutter_facebook_auth: ^4.3.4
- get_it: ^7.2.0
- shared_preferences: ^2.0.15
- shared_preferences_android: ^2.0.12
- firebase_storage: ^10.2.16

Por último creamos en el directorio /lib un archivo llamado di.dart. En este archivo crearemos un método estático asíncrono que utilizará el principio de Inyección de dependencias el cual usaremos de forma muy simple gracias a la herramienta GetIt. Gracias a esto tendremos una serie de Singletons registrados dentro de GetIt los cuales serán accesibles de forma global en todo el código.

Tiempos: Esta es una de las etapas mas abstractas, hay que pensar, verse mucha información, crear diagramas y plantearse como hacerlo. También hay que tener en cuenta las dependencias. Aproximadamente 40h.

3.4.2.Fase 1 – Re-factorización de directorios e implantación de ‘Clean Architecture’.

4. Re-factorización directorios.

Clean Architecture es un tipo de arquitectura de software que se basa en la separación por capas que se comunican entre si, cada una tiene un nivel de responsabilidad y se encargan de una cosa. Yo he implementado en cierta parte esta arquitectura por que me parece muy interesante y siempre he creído en la estandarización ya que permite que si otra persona consulta el código pueda entenderlo solo con saber que se basa en ‘CleanArch’

Durante este proceso se han creado diversas carpetas que se encargan de distintas cosas: (Todas ellas se encuentran dentro de '/lib')

- /device → En este directorio se implementarán los métodos y clases necesarias para conectar nuestra app con diversos sistemas del dispositivo en el que está corriendo.
- /domain → En este directorio se encuentran todas las implementaciones, servicios y entidades que necesitamos usar en nuestra app. Cabe destacar que en esta carpeta solo puede contener elementos de la lógica del negocio, nunca elementos de la interfaz. Dentro de la misma existirán al menos /entities y /services.
- /ui → Por último en este directorio almacenaremos todas las paginas y widgets que necesitaremos en nuestra app.

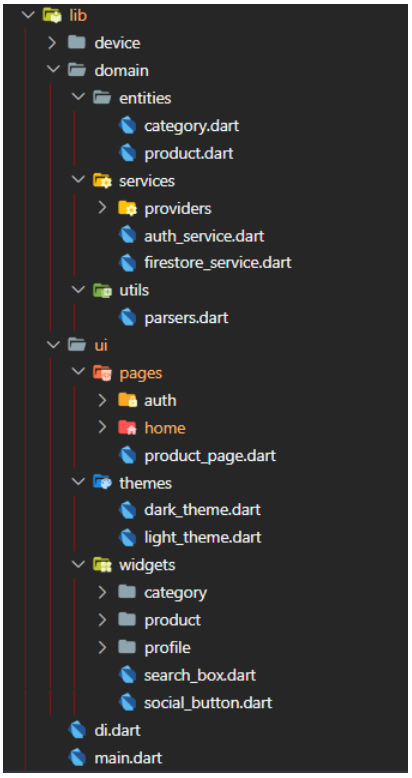

Ilustración 3: Contenido del directorio /ib (Final Proyecto)

* Además en este directorio se encuentran los siguientes ficheros:

- main.dart → Contiene el método main() de nuestra app y se encarga de cargar las dependencias y la configuración antes de iniciar la app.

```dart
// The Main method is the entry point for all our Flutter apps.
// Its job is to load al the prerequisites for our app and then run the app.
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  await MyApp.loadJson();
  await di.init();
  GetIt.I<FirebaseAuth>().authStateChanges().listen((user) {
    if (user != null && !(GetIt.I.isRegistered<CartProvider>())) {
      di.registerCartProvider();
    } else {
      if (GetIt.I.isRegistered<CartProvider>()) di.unRegisterCartProvider();
    }
  });
  runApp(MyApp());
}
```

Ilustración 4: Método main() del Fichero main.dart

- di.dart → Como ya explicamos antes contiene un método ejecutado en el 'main.dart'.

```dart
// di.dart file, this is the file that will be imported in the
// main.dart file and will be used to inject the dependencies.
GetIt _l = GetIt.instance;

Future<void> init() async {
  _l.registerFactory(() => FirebaseAuth.instance);
  _l.registerFactory(() => FirebaseFirestore.instance);
  _l.registerFactory(() => FirebaseStorage.instance);

  SharedPreferences sharedPreferences = await SharedPreferences.
getInstance();
  _l.registerSingleton<SharedPreferences>(sharedPreferences);

  bool isDarkMode = sharedPreferences.getBool('isDarkMode') ?? false;
  _l.registerSingleton(ThemeProvider(isDarkMode));
}

Future<void> registerCartProvider() async {
  List<Product> cart = await getCartFromFB().then((value) {
    print('Cart loaded from firebase $value');
    return value;
  }).catchError((error) => List<Product>.empty(growable: true));
  _l.registerSingleton<CartProvider>(CartProvider(cart));
}

Future<void> unRegisterCartProvider() async {
  _l.unregister<CartProvider>();
}
```

Ilustración 5: Contenido del fichero di.dart (Final Proyecto)

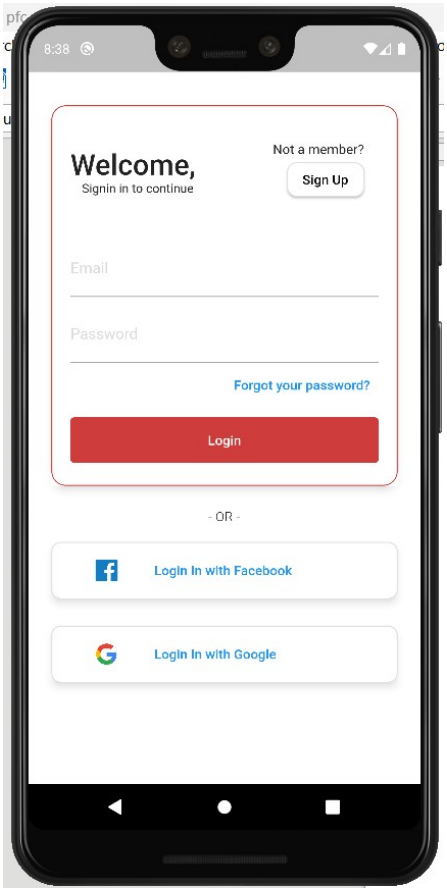

Ilustración 7: Página de inicio de sesión (Tema Claro)

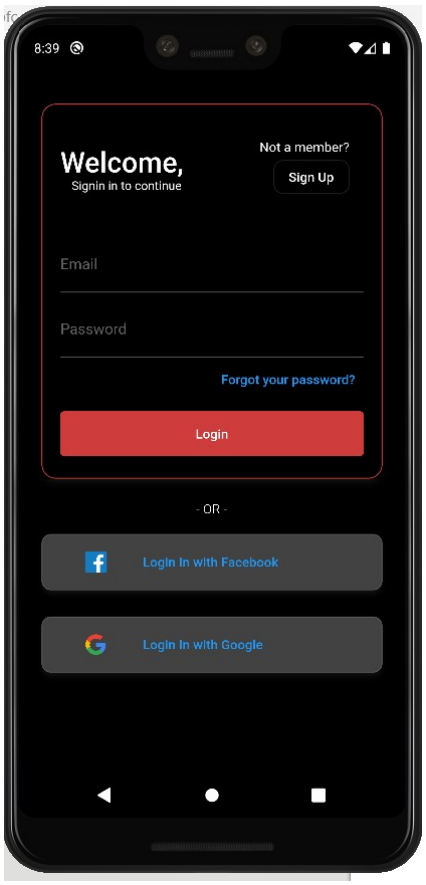

Ilustración 6: Página de inicio de Sesión (Tema Oscuro)

5. Eliminación de código residual de ‘flutter create’.

   Este apartado parece algo simple pero es necesario ya que después del comando ‘flutter create’ se generan bastantes archivos con una gran cantidad de comentarios, es importante eliminarlos para ganar un poco más de espacio en el tamaño del paquete final y para que sea mas cómodo leer el código.

   Tiempos: Esta etapa es más sencilla pero necesaria, aproximadamente 10h.

3.4.3.Fase 2 – Creación de IU.

6. Creación sistema de páginas y navegación.

Una vez ya está todo listo podemos proceder a realizar un conjunto de páginas sin lógica para poder ubicarnos. Dentro de la carpeta /ui crearemos la carpeta /ui/pages y dado que necesitaremos estilizar la app también crearemos /ui/themes.

- Durante este proceso paramos un momento de crear la IU para crear un Provider que gestione el estado del tema de forma que de manera global se actualice el tema. Esto lo haremos extendiendo la clase con ChangeNotifier que te permite tener 'listeners' cada vez que se modifica el objeto. Esta clase se registrará como 'Singleton' en GetIt para tener siempre la misma instancia de la misma.

Dentro de /ui/themes crearemos dos archivos dark_theme.dart y light_theme.dart. Dentro de ellos crearemos un método estático que devolverá un objeto ThemeData que contiene las preferencias de estilo. Seguidamente crearemos un conjunto de constantes con los colores principales. Durante el proceso de diseño se seguirán añadiendo datos de estilo en función de necesidad.

```dart
// This is the theme for the light theme. Defines the colors and the styles.
// You can change the predefined theme of a Widget here.
class LightTheme {
  static const brightness = Brightness.light;
  static const primaryColor = const Color.fromARGB(255, 207, 60, 60);
  static const lightColor = const Color.fromARGB(255, 220, 220, 220);
  static const backgroundColor = const Color.fromARGB(255, 255, 255, 255);
  static const reverseColor = const Color.fromARGB(255, 0, 0, 0);

  static ThemeData getLightTheme() {
    return ThemeData(
        brightness: brightness,
        cardTheme: CardTheme().copyWith(
          elevation: 6,
          shadowColor: Color(Colors.black45.value),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
            side: BorderSide(
              color: Colors.black12,
            ),
          ),
        ),
        listTileTheme: ListTileThemeData().copyWith(
          dense: true,
        ),
        elevatedButtonTheme: ElevatedButtonThemeData(
            style: ButtonStyle(
                backgroundColor: MaterialStateProperty.all(primaryColor),
                shape: MaterialStateProperty.all<RoundedRectangleBorder>(
                    RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(10.0),
                        side: BorderSide(color: Colors.black12))))));
  }
}
```

Ilustración 8: Contenido del fichero light_theme.dart (Final Proyecto)

En /ui/pages crearemos dos carpetas /ui/pages/home y /ui/pages/auth y en la primera crearemos una página vacía con un texto ‘home’.

7. Diseño de las páginas del sistema de autenticación.

Ahora procedemos a crear y estilizar las páginas del sistema de autenticación necesitaremos dos páginas login_page.dart y register_page.dart.

Para el Inicio de sesión tendremos que crear un formulario de usuario y contraseña y además unos botones para iniciar sesión con Facebook y Google. En todos ellos dejaremos los ‘callbacks’ vacíos para trabajar con ellos en la Fase 3.

Para la página de registro contaremos con nombre, email y contraseña como campos del formulario. Cabe destacar que el registro a través de Google y Facebook se realizarán desde el mismo botón que hay en la página de inicio de sesión.

8. Diseño del resto de páginas (‘Home’, ‘Cart’, ‘Profile’).

Ahora vamos a realizar el mismo proceso con /ui/pages/home, en ella encontramos diversos archivos y el directorio profile. En el directorio profile guardaremos las distintas páginas de configuración que necesitamos en la app. Fuera encontramos tabs_page.dart en la que tenemos un TabBarView que nos permite crear la barra inferior de navegación. Y realiza la redirección a cart_page.dart, home_page.dart y profile_page.dart

En cart_page.dart tenemos un Scaffold con separado en dos secciones, la parte superior donde aparecen los ítems en el carrito y la barra inferior donde aparece el precio total y el botón de ‘checkout’.

Siguiendo con la home_page.dart lo primero que construimos es un widget que hemos guardado én la carpeta /ui/widgets llamado search_box.dart, a eso le sigue un listado de categorías con un scroll lateral. En el inferior contendrá unas ListViews que se crearán en la Fase 4.

Por ultimo en el Perfil Personal Crearemos una tarjeta de presentación editable con los datos del usuario. Seguido de ello una serie de botones con el acceso a los submenus de configuración que por ahora se quedarán pendientes al igual que el avatar editable.

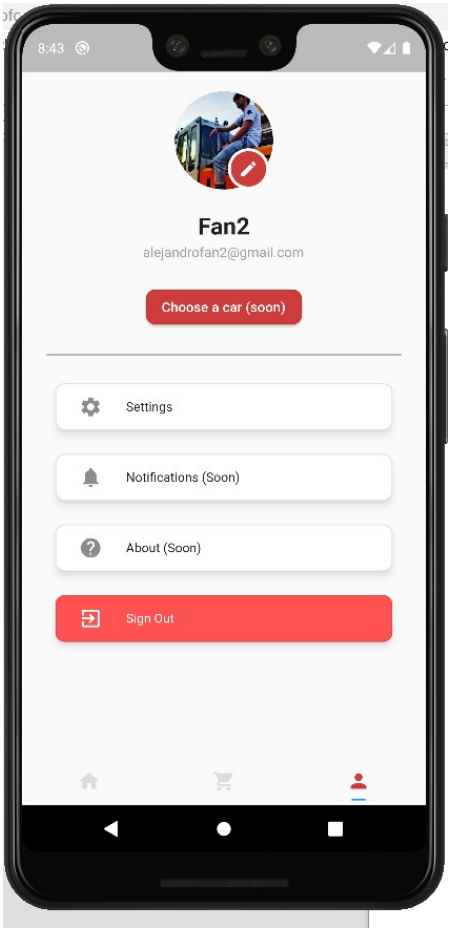

Ilustración 10: Perfil personal del usuario (Tema Claro)

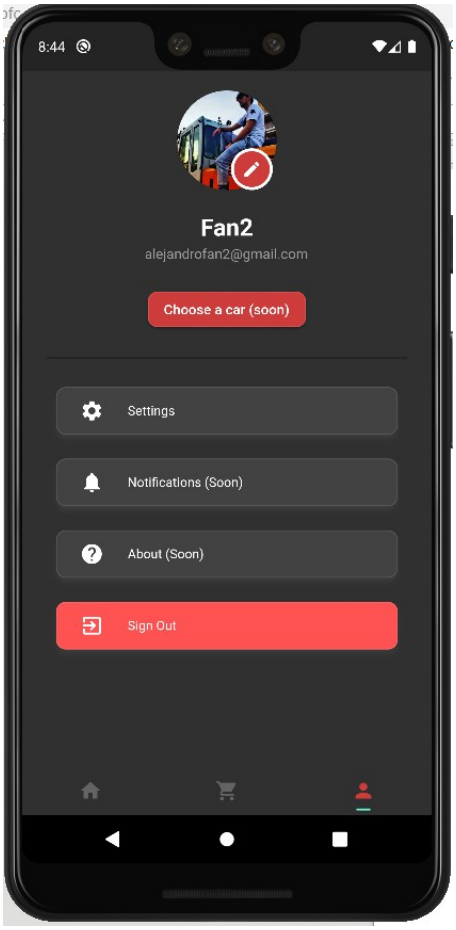

Ilustración 9: Perfil personal del usuario (Tema Oscuro)

Tiempos: Esta parte ha llevado bastante tiempo hasta llegar a ser una interfaz cómoda y vistosa. He tenido ciertos problemas a la hora de hacer que funcionara correctamente el Tema ya que no se actualizaba cuando le dabas al botón. Aproximadamente 30h.

3.4.4.Fase 3 – Enlace con FireBase.

9. Creación del proyecto en FireBase y configuración básica.

Para empezar debes de entrar en la consola de FireBase y crear un proyecto, después de ello debemos seguir los pasos que nos indica FireBase para completar su configuración. Por ahora solo configuraremos la versión de Android y dejaremos la de IOS para la Fase de Transpiración a IOS.

Una vez hecho esto deberemos ir al apartado de autenticación en la consola y activarla, seguido de realizar la configuración de Google y Facebook para FireBase Auth.

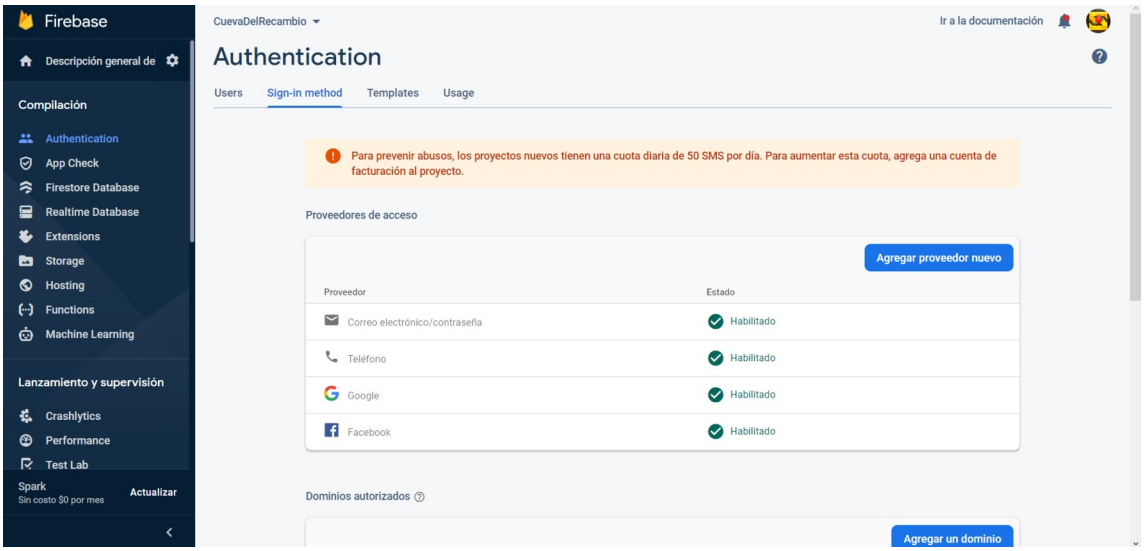

Ilustración 11: Consola de Firebase, sección Autenticación

10. Creación del Login Service para la autenticación.

Una vez hecho esto ya podemos volver al código, como ya tenemos todas las dependencias instaladas y registradas en GetIt simplemente

tendremos que crear dentro de /domain/services/ un archivo auth_service.dart en el que dispondremos de un método register, login, loginWithGoogle y loginWithFacebook. Y dentro de ellos llamaremos a los métodos respectivos de FireBase con sus correspondientes checks.

El código usado en para el Login Service se encuentra en el Anexo 2

Tiempo: Esta ha sido una de las partes más tediosas ya que era la primera vez que añadía FireBase a una aplicación en Flutter, pero gracias a la fácil sincronización entre ambas todo ha ido como la seda. Aproximadamente 30h.

3.4.5.Fase 4 – Productos y Categorías.

11. Creación de entidades para la asimilación de los datos en Flutter.

    Dentro del directorio /domain/entities debemos de crear dos clases una llamada category.dart y otra llamada product.dart.

    Dentro de ellos deberemos de tener la declaración con todas las variables necesarias para cada una y sus métodos para codificar y decodificar en JSON. Por último en el product.dart necesitaremos un método para que nos devuelva un producto por id.

```dart
// Is the most important entity of the application,
// it will be used to store the products of the application.
class Product {
  final String id;
  String name;
  String image;
  String brand;
  double price;
  String description;
  List<String> tags;
  List<dynamic> variantes;
  int quantity = 1;

  Product({
    this.id,
    this.name,
    this.image,
    this.variantes,
    this.brand,
    this.price,
    this.description,
    this.tags,
  });

  Product.fromJson(Map<String, dynamic> json)
      : id = json['id'],
        name = json['nombre'],
        brand = json['marca'],
        price = json['precio'],
        description = json['descripcion'],
        tags = List<String>.from(json['tags']),
        image = json['imagen'],
        variantes = json['variantes'];

  Map<String, dynamic> toJson() => {
        'id': id,
        'nombre': name,
        'marca': brand,
        'precio': price,
        'descripcion': description,
        'tags': tags,
        'imagen': image,
        'variantes': variantes,
      };


// This method will be used to search products by id into the json file.
  static Product productFromId(String id, {int quantity = 1}) {
    Product product;
    String jsonString = MyApp.productsJson;
    List<dynamic> jsonResponse = json.decode(jsonString);
    jsonResponse.forEach((element) {
      if (element['id'] == id) {
        product = Product.fromJson(element);
        product.quantity = quantity;
      }
    });
    return product;
  }
}
```

12. Creación de JSON para Productos / Categorías.

    En este momento los productos no están disponibles ya que dependerán del distribuidor pero por ahora basta con editar el JSON para actualizarlos.

    En el caso de las categorías he tenido que buscar uno por uno los logos de las marcas sin fondo y con una calidad decente a la vez de ligera. Para ya estar prevenidos este JSON ya viene con todos los modelos por marca.

13. Creación de ListViews para mostrar el contenido.

    Dentro del directorio /ui/widgets crearemos dos mas llamados ‘category’ y ‘product’. En el interior crearemos un archivo ‘list’ y otro ‘ítem’. Los archivos ‘list’ devolverán una ListView que contendrá una serie de iteraciones del archivo ‘ítem’ al cual se le pasan por parámetro los datos de cada producto/categoría.

    Tiempo: Otras 30h de trabajo he dedicado a esta parte en cierto modo también influye en la parte de IU del Home ya que las ListViews se cargan ahí.

3.4.6. Fase 5 – Perfil personal.

14. Creación de IU de las páginas de cambio de nombre, correo, teléfono, contraseña y avatar.

    Nos situamos en /ui/pages/home/profile y dentro de ella crearemos paginas acorde con los menús que necesitamos.

    - about_page.dart → Esta pagina contiene información sobre la empresa, términos y contacto. Se definirá dependiendo de las circunstancias a la hora del despliegue.
    - edit_profile_page.dart → Esta es una de las páginas con más widgets ya que permite el cambio de nombre, contraseña, email, teléfono y avatar. Y en cada uno de ellos hay que actualizar, parsear y testear con FireBase.

```dart
// Parsers file for the domain package,
// this file will be used to parse the json files
// and convert them to the domain entities.

List<int> parseIntList(List<dynamic> data) {
  List<int> parsed = List<int>.empty(growable: true);
  data.forEach((element) {
    parsed.add(int.parse('$element'));
  });
  return parsed;
}

int parseInt(String id) {
  return int.parse(id);
}

String parseEmail(String data) {
  if (data.contains('@') &&
      data.contains('.') &&
      data.length > 5 &&
      data.length < 100 &&
      data.contains(' ') == false) {
    return data;
  } else {
    return null;
  }
}
```

Ilustración 12: Contenido del fichero parsers.dart

- notifications_page.dart → Este es uno de los sistemas que se implementarán en futuras versiones de la app, consistirá en recibir notificaciones por ofertas o productos que han añadido al carrito y que están disponibles.
- settings_page,dart → En esta página daremos al usuario acceso a la configuración lógica de la app por el momento aparece la opción de cambio de tema entre claro y oscuro. En el futuro contendrá también la opción de idioma entre otras.

15. Creación de sistema de cambio de nombre, correo, teléfono, contraseña y avatar.

    Dentro de las ‘Cards’ que hemos definido en edit_profile_page.dart vamos a realizar la correspondiente llamada al método para actualizar el dato en la base de datos. Con excepción del avatar para el que necesitaremos subir la foto ya que solo almacena la URL de la misma.

Para ello tendremos que ir a la Consola de FireBase y entrar en FireBase Storage, después de realizar la configuración de FireBase podemos proceder al código. Como tenemos la dependencia registrada en GetIt e instalada simplemente tendremos que subir un archivo del almacenamiento de FireBase con la UID del user como nombre. Después de esto almacenamos la URL en FireBase Auth como normalmente haríamos.

Tiempo: Esta es otra de las partes de la app que no es complicada pero sí tediosa, al tener que crear 4 páginas nuevas de IU y editarlas para que sigan el estilo preestablecido. Aproximadamente 20h.

3.4.7.Fase 6 – Carrito Avanzado.

16. Creación de un sistema de carrito local funcional.

Vamos a crear un sistema muy similar al de theme_provider.dart, esta vez se llamara cart_provider.dart y también extenderá de ChangeNotifier. Esta clase tendrá un objeto List<Product> que almacenará todos los productos del carrito. Cada vez que se modifique la lista deberemos de realizar un notifyListeners() para avisar a la interfaz de que se tiene que actualizar. Por lo tanto crearemos un setter del carrito en el que recorreremos el array y calcularemos el precio total. Además para no tener que estar sobrescribiendo la lista crearemos los métodos addProduct() y removeProduct() los cuales se encargan de añadir un producto o eliminarlo de la lista. Cabe decir que estos métodos tienen una complejidad extra ya que cuando añades un producto que ya esta añadido no debe de duplicarlo sino añadir 1 a la variable cantidad. La misma situación pasa al revés cuando eliminas un elemento solo debe desaparecer si la cantidad es menor a 1.

17. Implementación de ‘FireBaseStorage’ en el carrito.

Crearemos un archivo llamado firestore_service.dart dentro de /domain/services/ en el que definiremos los métodos getCartFromFB() y updateCartToFB() estos métodos se encargan de actualizar los

```dart
// This class manages the state of auth and the current user.
// It is a singleton class. It is used to get the current user,
// change the user and know if user is logged in or not.
// The ChangeNotifier is used to notify the widgets.
// The widgets will be repainted.
class CartProvider with ChangeNotifier {
  CartProvider(this._cart) {
    _cart.forEach((element) {
      if (element.quantity > 0) {
        totalPrice += element.quantity * element.price;
      }
    });
  }

  List<Product> _cart;
  double totalPrice = 0;
  bool loading = false;

  List<Product> get cart => _cart;

  set cart(List<Product> cart) {
    cart.forEach((element) {
      if (element.quantity > 0) {
        totalPrice += element.quantity * element.price;
      }
    });
    _cart = cart;
    notifyListeners();
  }

  void addProduct(Product product) {
    int index = _cart.indexWhere((element) => element.id == product.id
);

    if (index == -1) {
      totalPrice += product.price;
      _cart.add(product);
    } else {
      _cart[index].quantity++;
      totalPrice += product.price;
    }

    updateCartToFB(_cart);
    notifyListeners();
  }

  void removeProduct(Product product, bool removeAll) {
    if (_cart
        .any((element) => element.id == product.id && element.quantity
> 1)) {
      if (removeAll) {
        int quantity;
        _cart.forEach((element) {
          if (element.id == product.id) {
            quantity = element.quantity;
          }
        });
        _cart.removeWhere((element) => element.id == product.id);
        totalPrice -= quantity * product.price;
      } else {
        _cart.forEach((element) {
          if (element.id == product.id) {
            element.quantity--;
          }
        });
        totalPrice -= product.price;
      }
    } else {
      _cart.removeWhere((element) {
        return element.id == product.id;
      });
      totalPrice -= product.price;
    }
    updateCartToFB(_cart);
    notifyListeners();
  }
}
```

Ilustración 13: Contenido del fichero cart_provider.dart

carritos en la base de datos, lo cual permite que al iniciar sesión desde otro lugar tu cuanta mantiene el carrito. Para lograrlo realiza una serie de checks en ambos casos y luego procede a guardar/obtener la información. Se guarda como arrays ordenados en FireBase Storage.

Por último tendremos que añadir los métodos recientemente creados a addProduct() y removeProduct() del cart_service.dart.

Tiempo: Tuve unos cuantos fallos durante esta etapa, al carrito no le gustaba eso de guardarse en la base de datos, pero al final todo funciona correctamente. Aproximadamente 40h.

3.5. Control de seguimiento

Hasta donde está especificado en el seguimiento es donde he llegado, además de eso he creado un workflow de Github Actions que permite que una vez hagas un commit en la rama main automáticamente cree una build de la app y la publique en Github.

```yaml
# That code is a GitHub Actions workflow that builds and deploy a
# new build every time we make a commit into main branch.

on:
  push:
    branches:
      - main

name: "Build and release"
jobs:
  build:
    name: "Build and release"
    runs-on: ubuntu-20.04
    steps:
      - uses: actions/checkout@v1
        with:
          fetch-depth: 0
      - uses: actions/setup-java@v1
        with:
          java-version: '12.x'
      - uses: subosito/flutter-action@v1
        with:
          flutter-version: '3.0.0'
      - run: flutter pub get
      - run: flutter build apk --debug --split-per-abi
      - name: "Push to releases"
        uses: ncipollo/release-action@v1
        with:
          artifacts: "build/app/outputs/apk/debug/*"
          token: ${{ secrets.token }}
```

Ilustración 14: Workflow de Github Actions

Al principio del diseño de la interfaz casi todos mis widgets eran muy cuadrados y toscos. En este proyecto también he aprendido que un buen uso de los bordes redondeados, la elevación y las sombras es la clave que nos proporciona Material Design la cual es una normativa/estandarización de diseño para Android o cualquier otra plataforma.

Por ultimo GetIt me ha solucionado mucho la vida ya que es una forma muy sencilla y eficaz de tener objetos Globales que no cambian y que podemos acceder a ellos cuando queramos.

4. Conclusiones

He conseguido llegar aproximadamente al 70% de finalización de la aplicación, utilizando software gratuito. En principio el objetivo era acabarlo al 100%, pero no esperaba estar tan centrado en la formación en el centro de trabajo como lo he estado. La temporalización de los objetivos ha sido bastante cercana a lo esperado, sin embargo los tiempos transcurridos entre fase y fase han limitado el tiempo global disponible.

En este momento esta disponible una versión 'APK' en versión beta a la espera de la configuración de la pasarela de pago para que se pueda utilizar en producción como Producto Mínimo Viable(PMV).

Debido a que la aplicación no esta acabada al 100% no se ha realizado aun la exportación a IOS por lo que no estará disponible por el momento.

Gracias a este trabajo he podido aprender muchísimos paradigmas y flujos de funcionamiento para Flutter como el fabuloso sistema de inyección de dependencias 'GetIt'. Considero que el grado de asimilación de objetivos ha sido correcto, incluso superior a lo esperado teniendo en cuenta que al estar trabajando, al fin y al cabo solo podía dedicar ciertos días libres a este proyecto. Se han completado 7 de los 10 capítulos planteados en el 'roadmap', como se ha comentado antes, realizar un proyecto como este solo, en horas libres durante 3 meses, conlleva riesgos. Pero por mucho que no este completado estoy deseando acabarlo y seguir aprendiendo.

El nivel de satisfacción es muy bueno considero que la app tiene una IU muy agradable y que cualquiera puede entender. Pese a que le faltan unos meses para estar del todo lista creo que puede funcionar muy bien cuando todos los sistemas estén acabados y funcionando.

Justo antes del paso a producción hay una serie de tareas a realizar, creación de un panel de administración, la optimización de la app, el cambio de textos a variables almacenadas en un CSV para el selector de idioma y la transpilación a IOS.

Mas adelante me gustaría incluir mas sistemas que le den al usuario mucha más sensación de estar en la mejor aplicación del motor aparte de una tienda. Cosas

como indicadores de cambio de aceite, neumáticos, calculadoras de precios, y demás…

Carta de agradecimiento

Quiero agradecer a todos los profesores por todo lo que me han enseñado durante todos los años que he estudiado informática, empece a cursarlo en 4º de la ESO en el IES Honori García (Vall d' Uxó) y ahí ya me di cuenta de que me gustaba mucho. Por eso decidí entrar en el IES Benigasló (Vall d' Uxó) a cursar el Ciclo Formativo de Grado Medio Técnico Sistemas Microinformáticos y Redes. Por cuestiones familiares el segundo curso lo curse en el IES Jaume I (Burriana). Después de esto ya estaba preparado para lo que yo siempre había querido, programar, y volví al IES Benigasló a cursar el Ciclo Formativo de Grado Superior Desarrollo de Aplicaciones Multiplataforma. En todos ellos he tenido muy buenos profesores, quiero agradecer enormemente a la mayoría de ellos todo lo que me han ayudado.

En especial quiero mencionar a Rosa Albert del IES Jaume I que me ayudó un montón y me enseñó el funcionamiento de un Sistema Operativo de forma inigualable. Fran Meneu del IES Jaume I siempre ayudo a la cooperación y el trabajo en equipo y se lo agradezco también.

También a Verónica Mascarós del IES Benigasló por que ha sido mi tutora 3 años, bueno 2 pero ya me entendéis, y me ha ayudado mucho y he aprendido muchas cosas.

Bibliografía

1. GARV_WADHWA, Re: Comparación Dart y JavaScript.
   https://es.acervolima.com/comparacion-de-dart-y-javascript/
2. Aitor, Re: ¿Por qué elegir Flutter? 9 Puntos para saber porque es la mejor opción.
   https://como-programar.net/blog/por-que-elegir-flutter/
3. Rakesh Patel, Re: Flutter vs Native: Which is the Best Technology to Develop Apps?
   https://www.spaceo.ca/blog/flutter-vs-native/
4. Flutter documentation.
   https://docs.flutter.dev
5. FlutterFire Firebase for Flutter documentation.
   https://firebase.flutter.dev/docs/overview/
6. Best Flutter UI templates. (ideas)
   https://github.com/mitesh77/Best-Flutter-UI-Templates
7. Shared_preferences. (pub.dev package)
   https://pub.dev/packages/shared_preferences

Anexos

Anexo 1 – Diagrama Ciclo de vida

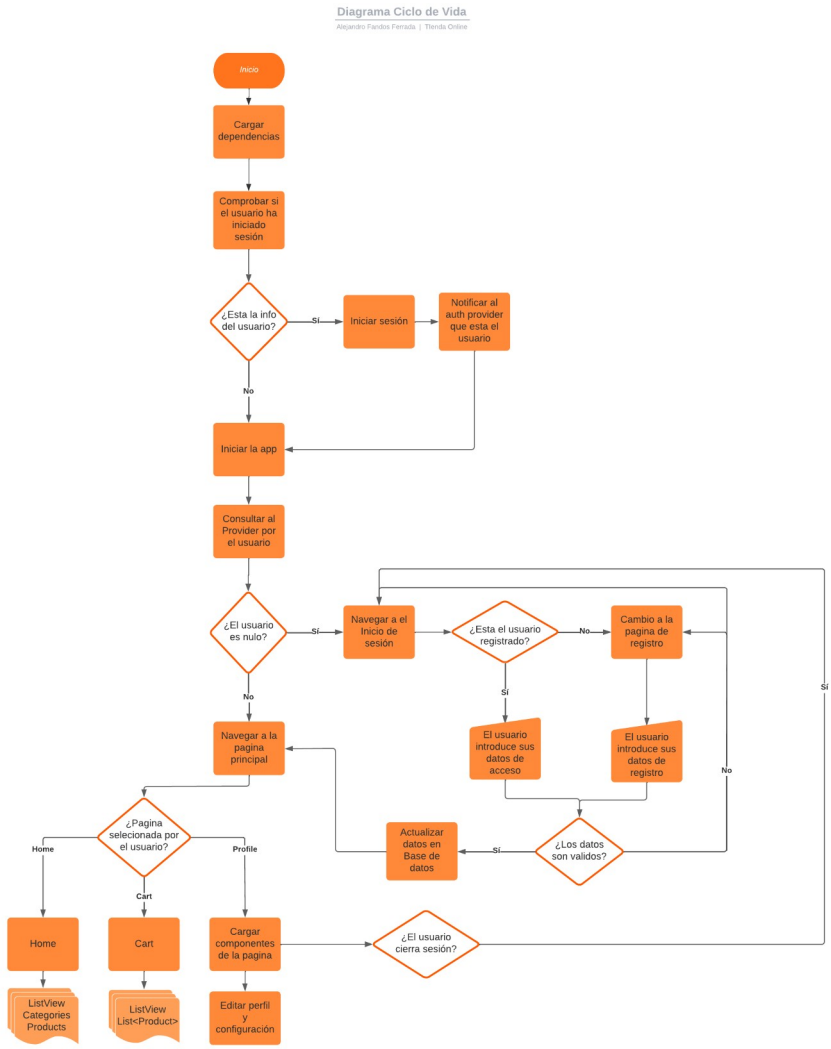

Ilustración 15: Diagrama del Ciclo de vida de la aplicación

Anexo 2 – Contenido del fichero auth_service.dart

```dart
// This static method logs the user with the given email and password.
// It returns a Future that resolves to the FirebaseUser if the login is successful.
// The method notifies the listeners of FirebaseAuth.
// The listeners make the UI update.
Future<bool> login(BuildContext context, String email, String password) async {
  try {
    FirebaseAuth.instance
        .signInWithEmailAndPassword(email: email, password: password);

    return true;
  } on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password')
      ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text("Try using a strong password.")));
    else if (e.code == 'email-already-in-use')
      ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text("That email is in use yet.")));
    return false;
  }
}

// This static method logs the user with Google.
// It returns a Future that resolves to the FirebaseUser if the login is successful.
// The method notifies the listeners of FirebaseAuth.
// The listeners make the UI update.
Future<bool> loginWithGoogle(BuildContext context) async {
  try {
    FirebaseAuth auth = FirebaseAuth.instance;

    final GoogleSignInAccount googleSignInAccount =
        await GoogleSignIn().signIn();

    if (googleSignInAccount != null) {
      final GoogleSignInAuthentication googleSignInAuthentication =
          await googleSignInAccount.authentication;

      final AuthCredential credential = GoogleAuthProvider.credential(
        accessToken: googleSignInAuthentication.accessToken,
        idToken: googleSignInAuthentication.idToken,
      );

      try {
        await auth.signInWithCredential(credential);
      } on FirebaseAuthException catch (e) {
        if (e.code == 'account-exists-with-different-credential') {
          ScaffoldMessenger.of(context).showSnackBar(SnackBar(
              content: Text("You have signed up with a different account.")));
          return false;
        } else if (e.code == 'invalid-credential') {
          ScaffoldMessenger.of(context)
              .showSnackBar(SnackBar(content: Text("Invalid credential.")));
          return false;
        }
      } catch (e) {
        ScaffoldMessenger.of(context)
            .showSnackBar(SnackBar(content: Text("An error occurred.")));
        return false;
      }
      return true;
    }
    return false;
  } on FirebaseAuthException catch (e) {
    if (e.code == 'account-exists-with-different-credential')
      ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text("Try using a different email.")));
    else if (e.code == 'invalid-credential')
      ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text("Invalid email or password.")));
    return false;
  }
}
```

```dart
// This static method logs the user with Facebook.
// It returns a Future that resolves to the FirebaseUser if the login is successful.
// The method notifies the listeners of FirebaseAuth.
// The listeners make the UI update.
Future<bool> loginWithFacebook(BuildContext context) async {
  try {
    FirebaseAuth auth = FirebaseAuth.instance;

    FacebookAuth instance = FacebookAuth.instance;

    final LoginResult result = await instance.login();

    if (result.status == LoginStatus.success) {
      final AuthCredential credential =
          FacebookAuthProvider.credential(result.accessToken.token);
      try {
        await auth.signInWithCredential(credential);
        if (auth.currentUser != null) {
          return true;
        }
      } on FirebaseAuthException catch (e) {
        if (e.code == 'account-exists-with-different-credential') {
          ScaffoldMessenger.of(context).showSnackBar(SnackBar(
              content: Text("You have signed up with a different account.")));
          return false;
        } else if (e.code == 'invalid-credential') {
          ScaffoldMessenger.of(context)
              .showSnackBar(SnackBar(content: Text("Invalid credential.")));
          return false;
        }
      } catch (e) {
        ScaffoldMessenger.of(context)
            .showSnackBar(SnackBar(content: Text("An error occurred.")));
        return false;
      }
    }
  } catch (e) {
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("An error occurred.")));
    return false;
  }
  return false;
}

// This static method registers the user with the given name, email and password.
// It returns a Future that resolves to the FirebaseUser if the registration is successful.
// The method notifies the listeners of FirebaseAuth.
// The listeners make the UI update.
Future<bool> register(
    BuildContext context, String name, String email, String password) async {
  try {
    User user = (await FirebaseAuth.instance
            .createUserWithEmailAndPassword(email: email, password: password))
        .user;
    await user.updateDisplayName(name);

    return true;
  } on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password')
      ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text("Try using a strong password.")));
    else if (e.code == 'email-already-in-use')
      ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text("That email is in use yet.")));
    return false;
  }
}
```

Anexo 4 – Diagrama Kaban