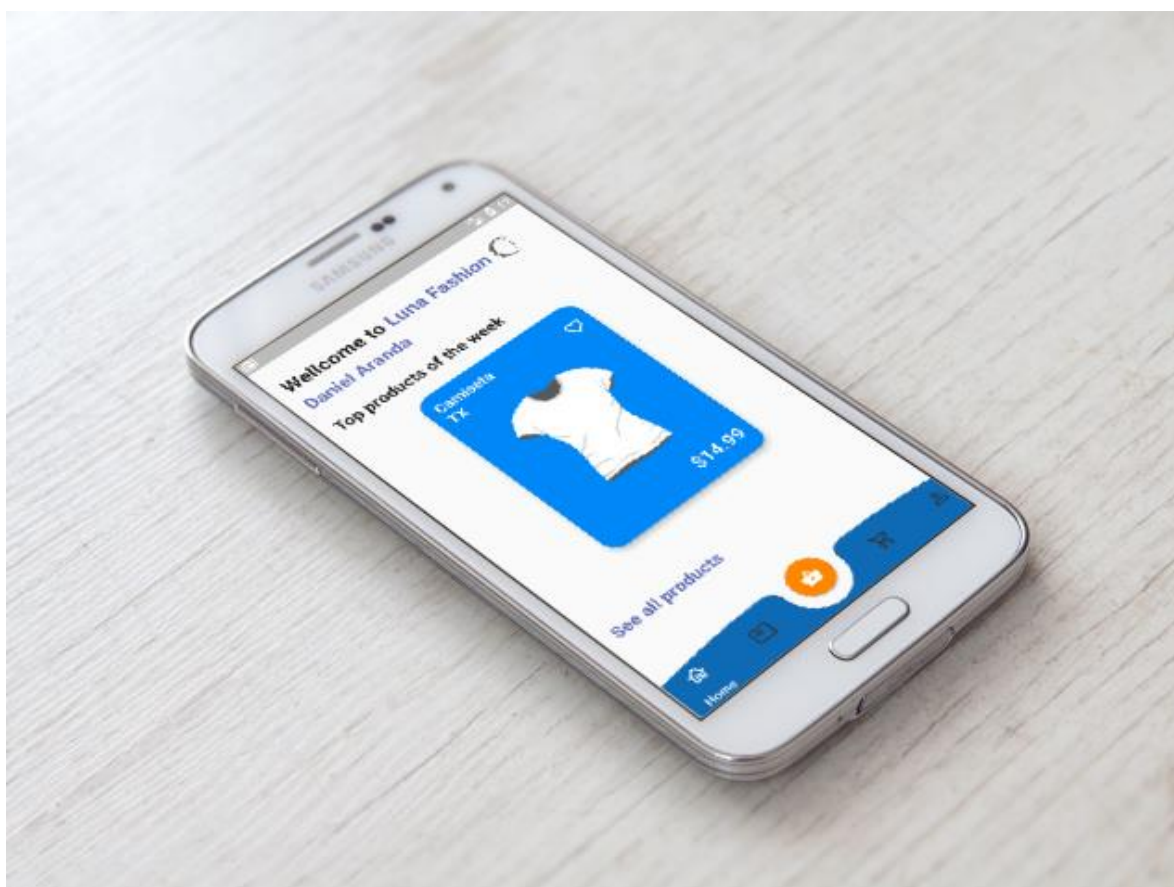


MEMORIA DEL PROYECTO INTEGRADO

“LUNA FASHION”



Realizado por

DANIEL ARANDA MAESTRO

Tutor

JOSÉ ANTONIO LAR SÁNCHEZ

Grado Superior “Desarrollo de aplicaciones multiplataforma”

Curso 2021/2022 – 2º DAM

Cesur (Málaga Este)

Índice

Resumen del proyecto	5
Introducción.....	7
Objetivos	9
Características del proyecto.....	9
1. Framework y lenguaje de programación	9
2. Funcionalidades principales.....	10
3. Infraestructura y servicios.....	11
4. Tipo de arquitectura	12
5. Packages externos.....	13
Finalidad.....	15
Medios materiales usados	17
Humanos	17
Hardware	18
Software.....	19
1. Flutter.....	19
2. Visual Studio Code	20
3. Android Studio / Xcode	21
4. Firebase.....	22
5. GitHub	23
Planificación del proyecto.....	25
Análisis de requerimientos	25
1. Concepto	25
2. Necesidades y posibles inconvenientes.....	26
3. Selección de la solución idónea	26
Cronograma del proyecto	27
Ejecución de las tareas asignadas	28
1. Semana 1: Diseño básico y recopilación de los recursos del proyecto.....	28
2. Semana 2: Proceso de autenticación y onboarding.....	32
3. Semana 3: Funcionalidad rol de Administrador.....	35
4. Semana 4: Funcionalidad rol de Usuario	36
5. Semana 5: Funcionalidad de notificaciones push.....	37
6. Semana 6: Realización de tests y correcciones.....	39
7. Semana 7: Realización de memoria técnica	40

Puntos a destacar del proyecto	40
Fase de pruebas	47
Conclusiones	49
Posibles mejoras	49
Apéndices.....	51
Referencias bibliográficas	53

Resumen del proyecto

Tras la pandemia, una tienda de barrio ha sufrido mucho y sus ventas han caído considerablemente, necesitan un cambio urgente para poder aumentar sus ventas y continuar con el negocio. Se pretende realizar una aplicación en el que los clientes puedan realizar sus pedidos a través de la aplicación y poder gestionar dichos pedidos por los usuarios administradores. De esta forma, los usuarios sólo tendrán que ir a la tienda cuando sus pedidos estén listos para recoger.

Como la aplicación se necesita que esté disponible para todo el mundo, es necesario realizar una aplicación en Android y otra en iOS para poder subirlo a las plataformas de Google Play y App Store. Para ello vamos a realizar la aplicación en Flutter, que nos permitirá exportar la aplicación en Android e iOS sin ningún tipo de problemas desde un mismo código para ambos.

Además, la aplicación necesitará de un backend donde se pueda gestionar las autenticaciones de los usuarios, una base de datos para almacenar la información de los productos, pedidos y usuarios, y un sistema de mensajería para las notificaciones push. Por ello, como solución, hemos optado por la plataforma Firebase que nos provee de todos estos y otros servicios de forma sencilla y rápida.

Introducción

Este proyecto consiste en una aplicación para una tienda de barrio, enfocado para el comercio menor.

Las funcionalidades principales de la aplicación serán las de seleccionar productos escogidos por el usuarios, agregarlos al carrito y realizar posteriormente la compra de dichos productos. Además, la aplicación distinguirá de uno o varios usuarios con el rol de Administrador para poder visualizar y gestionar los pedidos realizados a través de la aplicación.

La intención es que los usuarios compren los productos a través de la aplicación, estos pedidos pasan a un estado pendiente, donde el administrador preparará los pedidos y cuando estén listos cambia el estado del pedido a completado para que el usuario final pueda ir a recoger su compra en la tienda física.

Como funcionalidad adicional, para que la aplicación cobre más sentido, se agregará las notificaciones push para avisar tanto a los usuarios como a los administradores de los estados de los pedidos.

Objetivos

El objetivo principal de la aplicación es poder aumentar las ventas a través de la aplicación y facilitar los estados de los pedidos entre el administrador y el usuario final. Utilizando para ello una de las tecnologías más novedosas que hay actualmente y emplear una arquitectura de forma correcta para llegar a tener una escalabilidad y un mantenimiento fácil y rápido.

Por eso, es necesario conocer bien las características del proyecto.

Características del proyecto

Las características del proyecto podemos dividirlos en Framework a desarrollar, funcionalidades principales de la aplicación, infraestructura y servicios, el tipo de arquitectura y packages externos.

Cada uno de estos puntos es muy importante para conocer bien el proyecto y así entender todas las partes que lo componen.

1. Framework y lenguaje de programación

- Flutter: Es el kit de herramientas de *UI de Google* para realizar hermosas aplicaciones, compiladas nativamente, para móvil, web y escritorio desde una única base de código.

- Dart: Es un lenguaje de programación que usa *Flutter* y es *open source*, creado por Google en 2011. Se puede utilizar para aplicaciones web, servidores, aplicaciones de escritorio y aplicaciones móviles.

2. Funcionalidades principales

- Autenticación: El proceso de autenticación consiste en un ingreso sencillo con *email* y *contraseña*. Este ingreso se auto conectará con la plataforma *Firebase* donde nos proporcionará las credenciales y el token del usuario para determinar si está o no autenticado. Además cuenta con una página de registro con un formulario básico.
- Onboarding: Es un pequeño proceso por el que se ayudará al usuario a entender cómo funciona y para qué sirve la aplicación.
- Productos: Una de las funcionalidades principales de la aplicación en el que podremos obtener un listado de los productos y sus detalles.
- Carrito: Una de las funcionalidades principales de la aplicación en el que gestionará los productos que tiene almacenado el usuario en su carrito.
- Pedidos: Una de las funcionalidades principales de la aplicación en el que se obtendrá un listado de los pedidos tanto para el administrador que lo pueda gestionar como para el usuario para que sepa el estado actual de su pedido.
- Notificaciones Push: Funcionalidad extra para el administrador, con las notificaciones push podrá ser avisado de cuando se ha realizado un pedido, de quien lo ha hecho y de cuantos productos constará dicho pedido.
- Internalización (i18n): Funcionalidad extra que permite manejar varios idiomas dentro de la aplicación.

3. Infraestructura y servicios

- **Firebase:** Es una plataforma diseñada y creada por Google para desarrollar y facilitar la creación de aplicaciones, obteniendo así aplicaciones de manera rápida y de gran calidad. Además cuenta con muchísimos servicios. Esta plataforma tiene una pequeña capa gratuita y en cuanto se sobrepasa comienza a ser de pago.
- **Firebase core:** Permite la conexión y el manejo directo con la plataforma Firebase.
- **Firebase auth:** Servicio de autenticación, permite poder autenticarse con numerosos métodos como: por Email, por Google, por Facebook, por Teléfono, por Apple, por Anónimo, por Twitter, etc...
- **Firebase storage:** Servicio de almacenamiento, permite almacenar cualquier tipo de archivo, obtener una url y la posibilidad de poder descargar dichos archivos.
- **Cloud firestore:** Servicio de base de datos, esta base de datos es una base de datos documental muy similar a las No-SQL, permite almacenar la información a través de colecciones y documentos. También tiene la característica de agregar reglas de seguridad para que sólo pueda tener acceso tanto de escritura y/o de lectura de los usuarios que se deseen. Además cuenta con la posibilidad de crear índices de búsquedas para mayor rapidez.
- **Firebase crashlytics:** Servicio de fallos, es un servicio muy útil para recopilar y detectar errores que se producen en la aplicación, de esta manera podremos corregir estos fallos sin necesidad de que sea el usuario que nos reporte dichos errores.
- **Firebase analytics:** Servicio de métricas, nos permite a través de eventos conocer más al usuario final para poder estudiar así mejor los comportamientos que ocurren dentro de la aplicación.
- **Firebase messaging:** Servicio para poder utilizar las notificaciones push de los dispositivos móviles, se puede usar para enviar mensajes masivos a grupos específicos

o mensajes específicos a un usuario en particular. Se maneja a través de *topics* o *token del usuario*.

- Firebase remote config: Es un poderoso servicio que permite almacenar variables de entornos y poder obtenerlo en la aplicación para no tener que almacenar datos sensibles dentro de la aplicación.

4. Tipo de arquitectura

- Clean Architecture: Es un tipo de arquitectura en el que consiste separar los elementos principales de un proyecto en diferentes capas en forma de anillos. De fuera a dentro podemos encontrar la capa de Infraestructura, donde encontramos las conexiones con todo lo exterior como bases de datos, servicios, interfaces externas, webs, etc. Una capa más a dentro encontramos la de Aplicación donde se definen todas las reglas de negocio y más a dentro la capa de Dominio donde se definen los modelos, las entidades y otras reglas de negocio entre otro. Al separar por capas conseguimos que nada esté completamente acoplado en especial con la capa de Presentación, si hubiera que cambiar algo relativo a la infraestructura como la implementación de una nueva base de datos no se verían afectadas todas las demás capas.
- Principios Solid: Son unas pequeñas reglas que ayudarán a desarrollar software de calidad. Cada letra tiene una característica especial, la primera S de single responsibility principle, trata de convertir algo complejo en muchas cosas simples y definir así métodos que sólo hagan lo que menciona el nombre del método. El segundo principio es la O de open/closed, abierto para escalar el proyecto y cerrado para modificaciones de cambios. De esta forma hacemos más robusto y estable la aplicación. El tercer principio la L de Liskov substitution, consiste en que las clases derivadas deben poder sustituirse por sus clases base. El cuarto principio la I de

Interface segregation, consiste en definir cada interfaz por caso de uso de esta forma queda mucho más independiente cada uso que se le vaya a dar a la aplicación. Por último queda la D de Dependency inversión, recomienda que los módulos de alto nivel no deberían depender de los módulos de bajo nivel, se refiere a que hay que crear clases abstractas para que sus implementaciones sean totalmente independientes.

5. Packages externos

- `Flutter_navite_splash`: Permite crear un splash personalizado en nativo de forma sencilla y rápida. Gracias a esto se elimina el splash blanco que viene por defecto en Flutter.
- `Flutter_launcher_icons`: Permite crear el icono de la aplicación de forma sencilla y rápida para Android e iOS.
- `Flutter_riverpod`: Es uno de los manejadores de estados más potentes que hay junto con Flutter Bloc y Provider.
- `Dio`: Es un poderoso cliente Http con el que podemos crear decoradores y configuraciones de forma sencilla.
- `Logger`: Permite configurar los logs de tu proyecto.
- `Another_flushbar`: Instanciar toasts personalizados ya creados.
- `Lottie`: Renderiza efectos de animaciones con archivos específicos de Lottie.
- `Flutter_svg`: Permite visualizar archivos en svg.
- `Image_picker`: Permite capturar y seleccionar imágenes de tu dispositivo.
- `Url_launcher`: Permite abrir un navegador.
- `Flutter_slidable`: Widget personalizado listo para usarse, permite deslizar ítems para agregar funciones extras.

Finalidad

La finalidad de esta aplicación es la de poder ayudar a pequeños comercios con dificultades para que puedan de una manera fácil y económica aumentar sus ventas. Con esto se consigue más participación en el uso de las aplicaciones móviles. Además que facilitamos a los usuarios finales el realizar pedidos sin la necesidad de ir a la tienda física.

Como se puede observar, todo es una cadena, que va desde el propio comercio hasta los usuarios finales. Con esto, indirectamente ganan todos.

Medios materiales usados

El equipamiento técnico es uno de los puntos más importantes para un desarrollador, ya que es con lo que trabaja en el día a día. Tener un buen espacio de trabajo ayuda a enfocarse mejor y a ser más productivo.

Además, es necesario que cuente con las mejores herramientas y mantenerse actualizado con las últimas tecnologías, ya que es muy fácil quedarse desactualizado.

Para este proyecto, se definirán todos los medios que se va a utilizar, tanto humanos, hardware y software.

Humanos



Daniel Aranda Maestro

Desarrollador de aplicaciones móviles con 4 años de experiencia.

Tengo amplios conocimientos en bases de datos, infraestructuras y servidores para el desarrollo de cualquier tipo de aplicación.

Hardware



El espacio de trabajo cuenta con un monitor de 34 pulgadas ultrawide, portátil Acer, portátil MacBook Pro, móvil Huawei Honor 8x, móvil iPhone 11, teclado y ratón.



Software

Para poder desarrollar un proyecto es necesario hacer uso de algunos programas informáticos que hacen posible la ejecución de tareas específicas dentro de un ordenador.

Para nuestro proyecto los softwares necesarios son:

1. Flutter

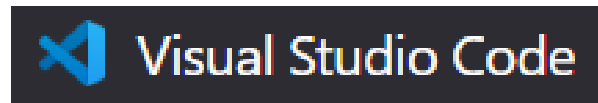


Flutter es un framework multiplataforma creado por Google y que compila en nativo. Permite crear aplicación más rápido, diseñar hermosas apps y publicarlas en cualquier tipo de plataforma.

Sus principales características son:

- **Desarrollo Rápido:** Trae tu aplicación a la vida en cuestión de milisegundos con Hot Reload. Utilice un completo conjunto de widgets totalmente personalizables para crear interfaces nativas en cuestión de minutos.
- **UI Expresiva y Flexible:** Monta rápidamente funcionalidades con el foco en la experiencia de usuario nativa. La arquitectura en capas permite una completa personalización, que resultan en un renderizado increíblemente rápido y diseños expresivos y flexibles.
- **Rendimiento Nativo:** Los widgets de Flutter incorporan todas las diferencias críticas entre plataformas, como el scrolling, navegación, iconos y fuentes para proporcionar un rendimiento totalmente nativo tanto en iOS como en Android.

2. Visual Studio Code



Es uno de los editores de código más potentes que existe, pudiéndose utilizar en Windows, macOS y Linux. Otorga innumerables posibilidades gracias a la infinidad de extensiones que dispone.

Es una de las principales elecciones para desarrolladores web y javascript, pero para nuestro proyecto en Flutter sirve perfectamente. Se acopla perfectamente a cualquier lenguaje de programación y dispone de extensiones específicas para dichos lenguajes.

Estas son algunas extensiones que se utilizará en este proyecto:

- Flutter
- Dart
- Dart Data Class Generator
- Dart Union Class Generator
- Flutter Intl
- Awesome Flutter Snippets
- Material Icon Theme
- GitHub
- Better Comments
- Image preview

3. Android Studio / Xcode



Android Studio y Xcode son dos entornos de desarrollo (IDE) para el desarrollo de aplicaciones móviles.

Android Studio se utiliza para el desarrollo de dispositivos Android en nativo con lenguajes de programación Java y Kotlin. Usa un sistema de compilación flexible basado en Gradle.

Xcode se utiliza para el desarrollo de dispositivos de Apple, sus lenguajes de programación son Swift y Objective-C entre otros.

En estos IDEs se suelen desarrollar las aplicaciones en nativo con dichos lenguajes, esto significa que si nuestro proyecto se desarrollara en nativo se tendrían que realizar dos proyectos uno para Android y otro para iOS. Sin embargo, este proyecto está realizado en Flutter y no necesitaremos desarrollar dos proyectos sino que con el mismo código podremos exportarlo para ambas plataformas.

Únicamente utilizaremos estos entornos para las configuraciones nativas de nuestro proyecto en Flutter ya que como se mencionó anteriormente compila en nativo y es por ello que requiere de ciertas configuraciones tanto en Android como iOS.

4. Firebase



Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Su finalidad es facilitar el desarrollo, crecimiento, monetización y análisis de las aplicaciones.

Sus servicios más destacados son:

- Realtime Database: bases de datos en tiempo real, son No-SQL y almacena los datos como JSON. Firebase envía automáticamente eventos a las aplicaciones cuando los datos cambian.
- Autenticación de usuarios: ofrece un sistema de autenticación que permite tanto el registro como el acceso utilizando perfiles de otras plataformas externas.
- Almacenamiento en la nube: sistema de almacenamiento donde los desarrolladores pueden guardar los ficheros de sus aplicaciones y sincronizarlos. Este almacenamiento es de gran ayuda para tratar archivos de los usuarios (por ejemplo, fotografías que hayan subido).
- Crash Reporting: útil para mantener y mejorar la calidad de la app, hay que prestar especial atención a los fallos, por lo que los seguimientos de errores son clave para poder actuar y solucionarlos.
- Cloud Messaging: útil para envío de notificaciones y mensajes a diversos usuarios en tiempo real.

Y otros muchos más servicios. Resulta muy completa y con grandes y numerosos beneficios, aunque hay que mencionar que hay que tener cuidado con la escalabilidad de Firebase porque la capa gratuita es muy limitada.

5. GitHub



GitHub es un servicio basado en la nube que aloja un sistema de control de versiones llamado Git.

Permite crear y descargar repositorios. Además de poder administrar los proyectos con el sistema de control Git, es muy útil para trabajar en equipo ya que cada uno puede ir trabajando en diferentes ramas del proyecto y luego hacer merge para fusionar las ramas.

Los comandos más comunes para trabajar con git son:

- Git init
- Git status
- Git branch
- Git checkout “rama”
- Git add .
- Git commit -m “mensaje”
- Git pull
- Git push

El link al repositorio de este proyecto es:

https://github.com/Anullos/luna_fashion

Planificación del proyecto

La planificación del proyecto es una de las tareas más importantes que se ha de realizar y la primera. Consiste en llevar a cabo un plan de acción para todas las fases que de un proyecto. Por ello, se definirá un proceso de organización sistemático de las diferentes tareas a realizar, así como de los recursos necesarios para llevarlas a cabo.

El objetivo principal de este es la realización del mismo y asegurar los mejores resultados.

Análisis de requerimientos

El análisis es la primera fase a ejecutar de un proyecto, es donde a partir de una idea o concepto se aplicará los conocimientos necesarios para poder llevarlo a cabo. En esta etapa, se define las posibles soluciones y necesidades, al igual que los posibles inconvenientes. Es conveniente realizar además un estudio de viabilidad para saber si será posible y viable su realización.

1. Concepto

Crear una aplicación, donde los usuarios escoger productos de la tienda y realizar sus propios pedidos, se deberá conocer el estado del pedido para que el usuario sepa cuando ha de ir a recoger su pedido a la tienda.

2. Necesidades y posibles inconvenientes

Para poder realizar este proyecto necesitaremos establecer en que *lenguaje de programación* y sobre que *tecnología* lo vamos a desarrollar. Al ser un desarrollo móvil es posible que se necesite realizarlo en dos proyectos en el caso que se decida desarrollarlo en nativo. Otra opción sería usar una tecnología híbrida para poder realizarlo desde un solo proyecto para ambas plataformas (Android e iOS).

El siguiente paso será elegir el *backend* de la aplicación, esto es muy importante para no delegar a la aplicación toda la lógica de negocio y la responsabilidad.

Definir la base de datos, que sea sencilla poder realizar futuras migraciones en caso de escalabilidad. Al igual que definir todos los modelos se necesitarán.

Se tendrá en bastante consideración cual será el tiempo disponible para la realización de dicho proyecto, lo que influirá bastante en la toma de decisiones sobre cuál será la mejor opción para desarrollarlo.

3. Selección de la solución idónea

Como se mencionó anteriormente, el tiempo es un factor clave en la toma de decisiones para escoger las mejores tecnologías en base a nuestras necesidades.

Así que, el framework elegido ha sido *Flutter*, esto nos dará la ventaja de poder desde un mismo proyecto exportar la aplicación en Android e iOS sin necesidad de perder el tiempo en desarrollar dos proyectos bajo dichas plataformas.

Además, se utilizará *Firebase* como backend y la base de datos que ya tiene integrada en su plataforma ya que es una de las mejores opciones para proyectos pequeños y de rápida ejecución. Pudiendo migrarse de forma sencilla en caso de que la aplicación escale de manera significativa.

Cronograma del proyecto

Se establecerá los plazos de ejecución de cada actividad. A medida que el proyecto avanza y con la ayuda de paneles de control, se deberá verificar el progreso del proyecto, la calidad y el cumplimiento de las especificaciones iniciales. De esta manera podremos tener mayor conocimiento si se está estableciendo los plazos definidos.

Este cronograma será una de las metodologías ágiles que servirá, en especial utilizaremos scrum que se basa en sprints de trabajo cortos y estructurados.

El tiempo estimado para la realización de este proyecto es de 7 semanas:

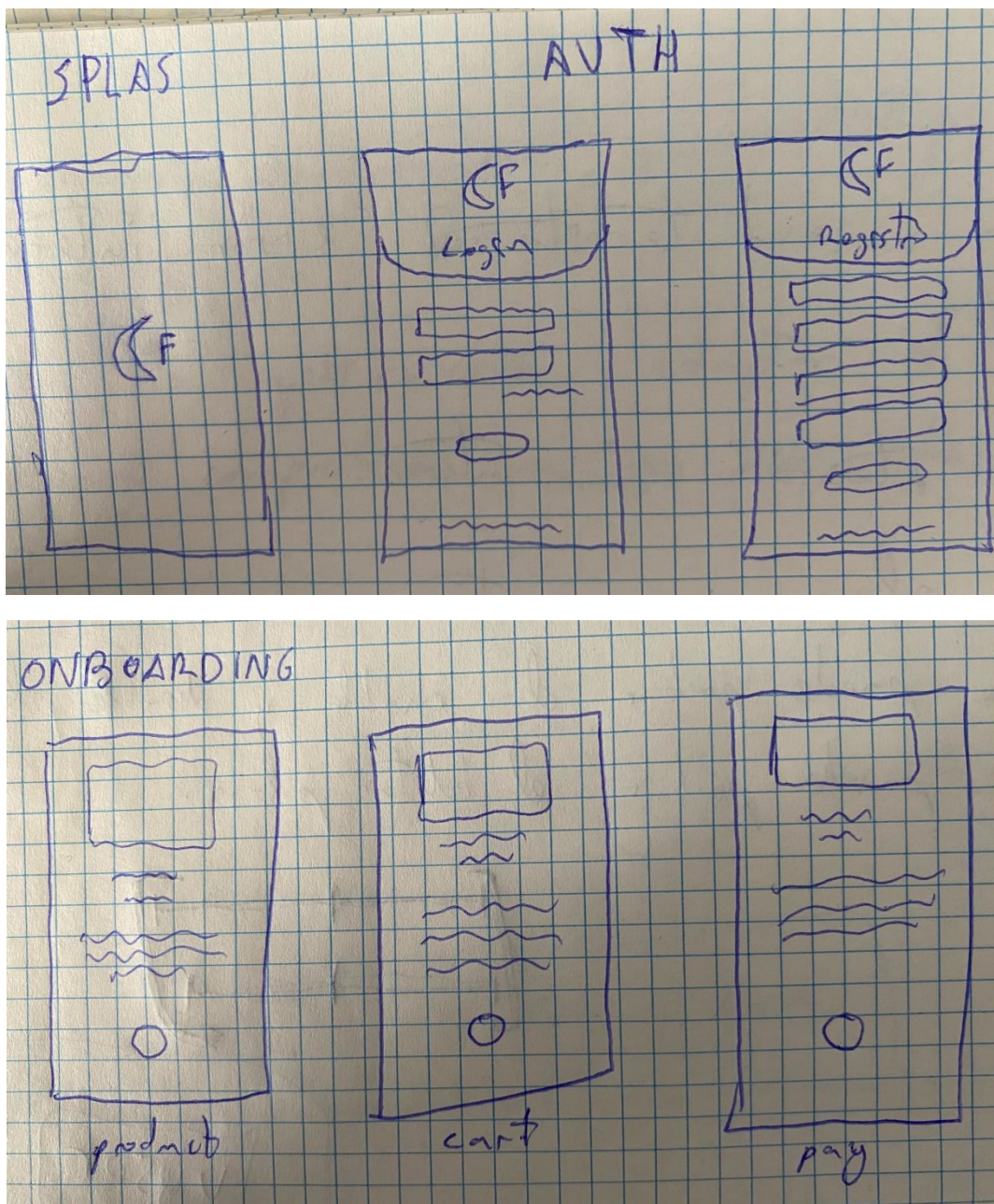
- Semana 1: Diseño básico y recopilación de los recursos del proyecto.
- Semana 2: Proceso de autenticación y onboarding.
- Semana 3: Funcionalidad rol de Administrador (pantallas, listar productos, crear y borrar productos, listas pedidos de los usuarios y modificación del estado de los pedidos).
- Semana 4: Funcionalidad rol de Usuario (listar productos, llenar carrito de compra, enviar pedido y lista de sus pedidos).
- Semana 5: Funcionalidad de notificaciones push para el administrador y mejoras varias de las interfaces.
- Semana 6: Realización de pruebas unitarias y de verificación. Correcciones que se pudieran producir. Y prueba de aceptación.
- Semana 7: Realización de memoria técnica.

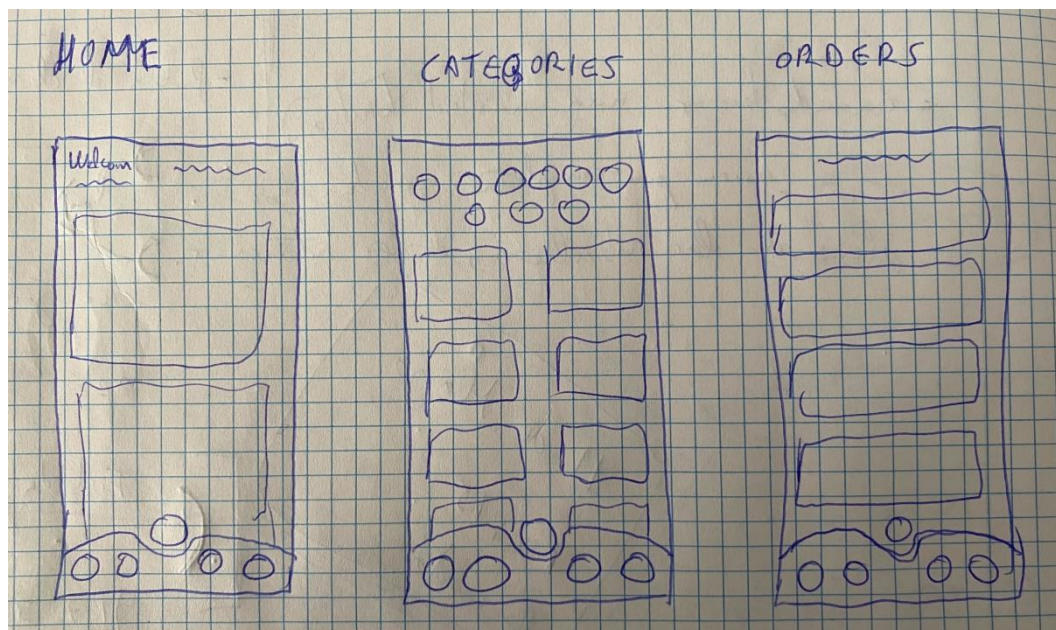
Ejecución de las tareas asignadas

En este apartado, se verá alguno de los puntos más destacados para la realización de las tareas que se planificaron.

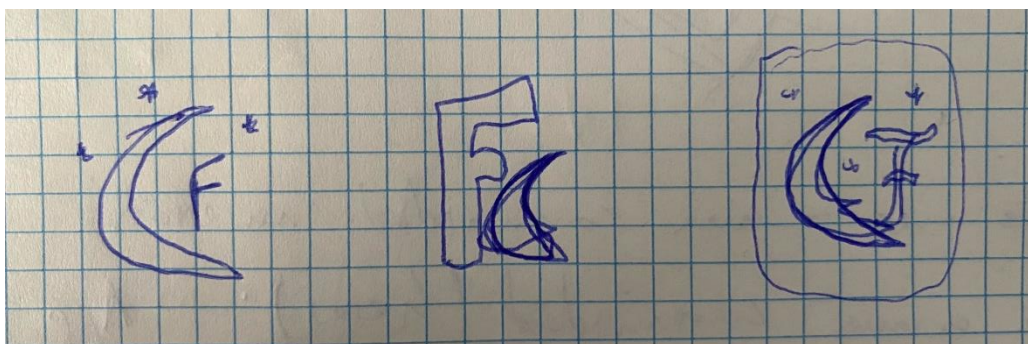
1. Semana 1: Diseño básico y recopilación de los recursos del proyecto

Primeros bocetos de cómo serían algunas pantallas de la aplicación.

























































Crear un logo para la aplicación y que de imagen a la marca.



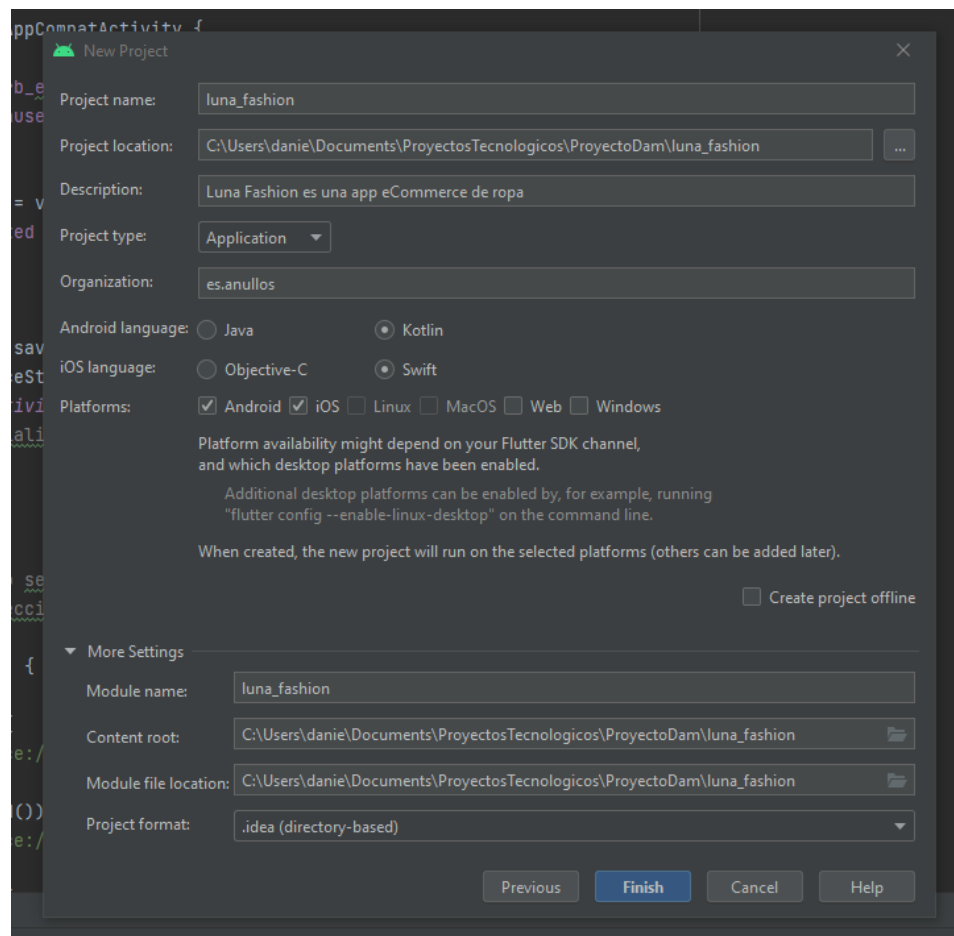
Recopilar imágenes para los productos de la aplicación.

quipo > Documentos > Curso DAM > 2ºCurso > Proyecto > project > recursos

Buscar

 camisa_azul.png Archivo PNG 281 KB	 camisa_azul_corta.png Archivo PNG 215 KB	 camisa_azul_pro.png Archivo PNG 310 KB
 camiseta_blanca.png Archivo PNG 33,9 KB	 camiseta_gris.png Archivo PNG 500 KB	 camiseta_gris_pro.png Archivo PNG 1,35 MB
 camiseta_roja.png Archivo PNG 1,00 MB	 camiseta_roja_pro.png Archivo PNG 170 KB	 camiseta_rosa.png Archivo PNG 63,0 KB
 camiseta_sin_mangas.png Archivo PNG 35,7 KB	 camiseta_sin_mangas_amarilla.png Archivo PNG 43,2 KB	 camiseta_sin_mangas_negra.png Archivo PNG 13,4 KB
 cap.png Archivo PNG 28,9 KB	 falda_dos.png Archivo PNG 740 KB	 falda_tres.png Archivo PNG 341 KB
 falda_uno.png Archivo PNG 2,52 MB	 gorra_cinco.png Archivo PNG 386 KB	 gorra_cuatro.png Archivo PNG 1,51 MB
 gorra_dos.png Archivo PNG 183 KB	 gorra_seis.png Archivo PNG 341 KB	 gorra_tres.png Archivo PNG 222 KB
 gorra_uno.png Archivo PNG 641 KB	 pantalon_cinco.png Archivo PNG 805 KB	 pantalon_cuatro.png Archivo PNG 393 KB
 pantalon_dos.png Archivo PNG 725 KB	 pantalon_seis.png Archivo PNG 292 KB	 pantalon_tres.png Archivo PNG 728 KB
 pantalon_uno.png Archivo PNG 5,59 MB	 polo_mujer.png Archivo PNG 858 KB	 polo_rojo.png Archivo PNG 196 KB
 sudadera_cuatro.png Archivo PNG 253 KB	 sudadera_dos.png Archivo PNG 135 KB	 sudadera_tres.png Archivo PNG 165 KB
 sudadera_uno.png Archivo PNG 132 KB	 vestido_cuatro.png Archivo PNG 700 KB	 vestido_dos.png Archivo PNG 132 KB
 vestido_tres.png Archivo PNG 92,8 KB	 vestido_uno.png Archivo PNG 85,7 KB	 zapatos_cinco.png Archivo PNG 322 KB
 zapatos_cuatro.png Archivo PNG 202 KB	 zapatos_dos.png Archivo PNG 116 KB	 zapatos_seis.png Archivo PNG 688 KB
 zapatos_siete.png Archivo PNG 202 KB	 zapatos_tres.png Archivo PNG 200 KB	 zapatos_uno.png Archivo PNG 513 KB
 cap.png Archivo PNG 28,9 KB	 dress.png Archivo PNG 3,08 KB	 informal.png Archivo PNG 7,23 KB
 jeans.png Archivo PNG 2,56 KB	 shoe.png Archivo PNG 3,24 KB	 tshirt.png Archivo PNG 2,35 KB
 tienda.jpg Archivo JPG 77,2 KB		

Creación del proyecto en Flutter.

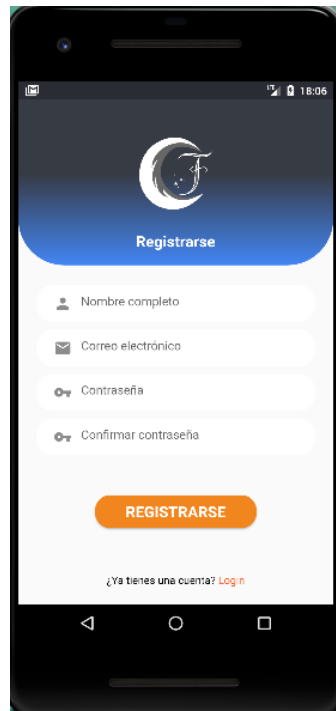
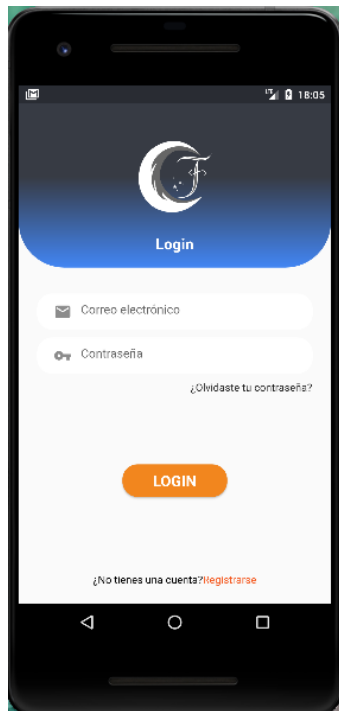


Creación del proyecto en Firebase.

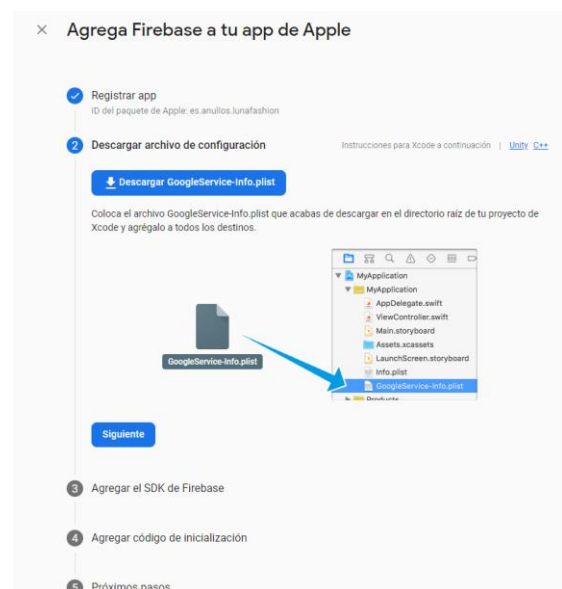
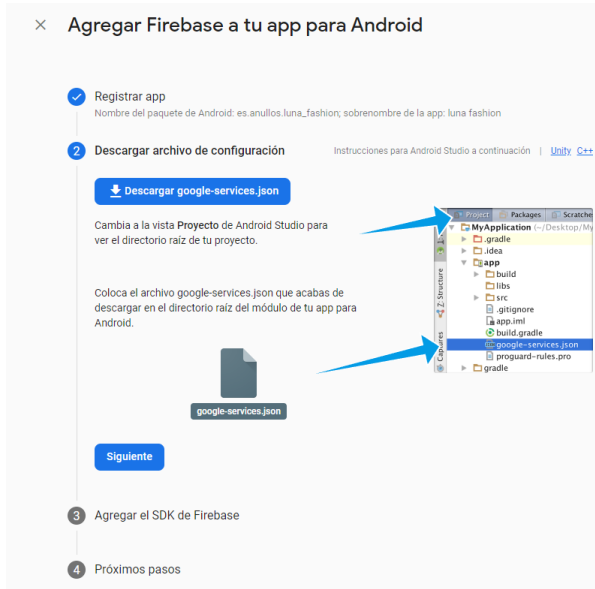


2. Semana 2: Proceso de autenticación y onboarding

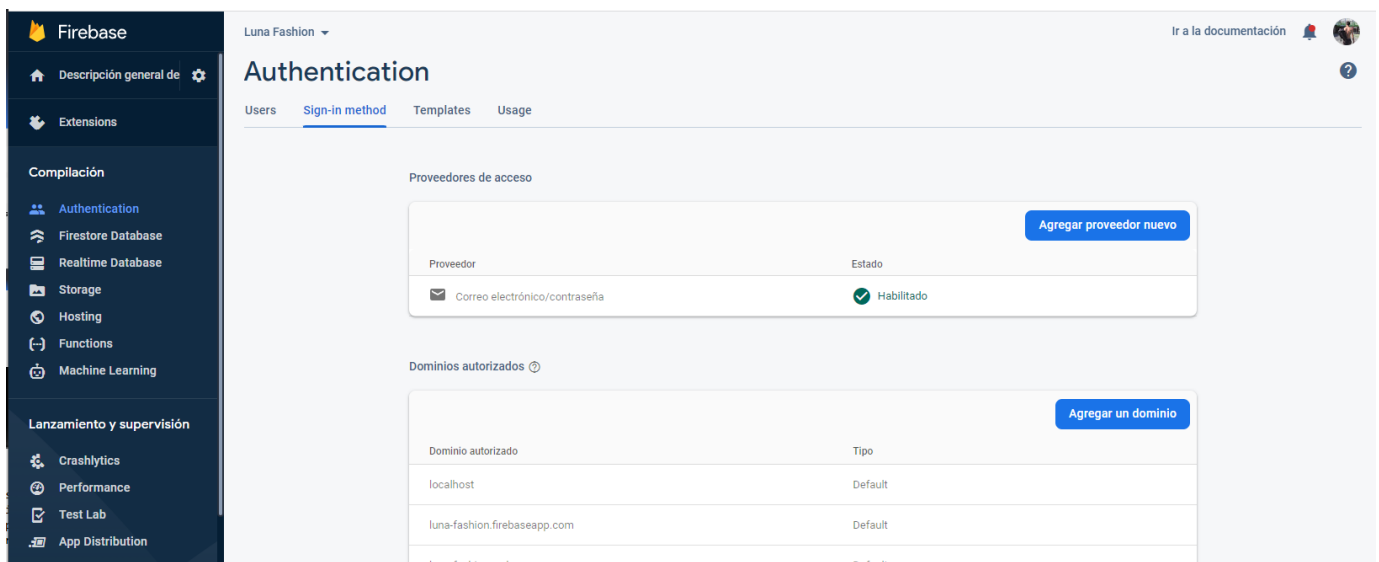
Se empieza a maquetar las pantallas pertinentes de la autenticación y del proceso del onboarding.



A continuación, desde Firebase, agregamos la configuración tanto para Android como para iOS.



Se habilita el método de Sign-in con el correo electrónico.



Se agregan las dependencias que se utilizarán de Firebase a nuestro proyecto en el archivo *pubspec.yaml*.

```
#firebase
firebase_core: ^1.14.0
firebase_auth: ^3.3.12
firebase_storage: ^10.2.10
firebase_analytics: ^9.1.3
firebase_crashlytics: ^2.6.0
firebase_messaging: ^11.2.14
cloud_firestore: ^3.1.11
firebase_remote_config: ^2.0.5
```

Se prepara la configuración del proyecto para admitir la internacionalización.

Archivo: pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
```

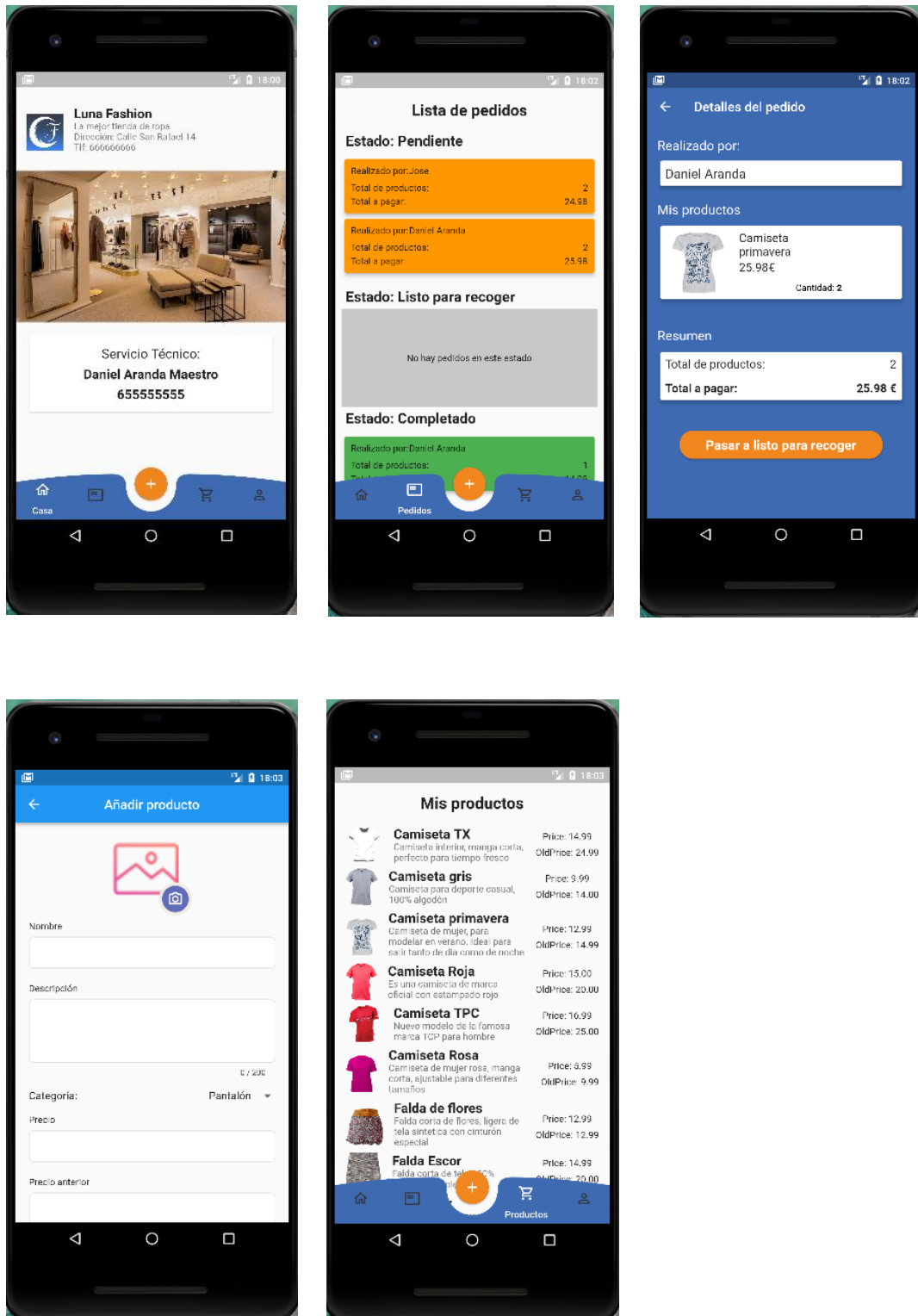
```
flutter_intl:
  enabled: true
  main_locale: es
  arb_dir: lib/src/shared/presentation/l10n
  output_dir: lib/src/shared/presentation/l10n/generated
```

Archivo: Main.dart

```
localizationsDelegates: const [
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
  GlobalCupertinoLocalizations.delegate,
  S.delegate,
],
supportedLocales: S.delegate.supportedLocales,
```

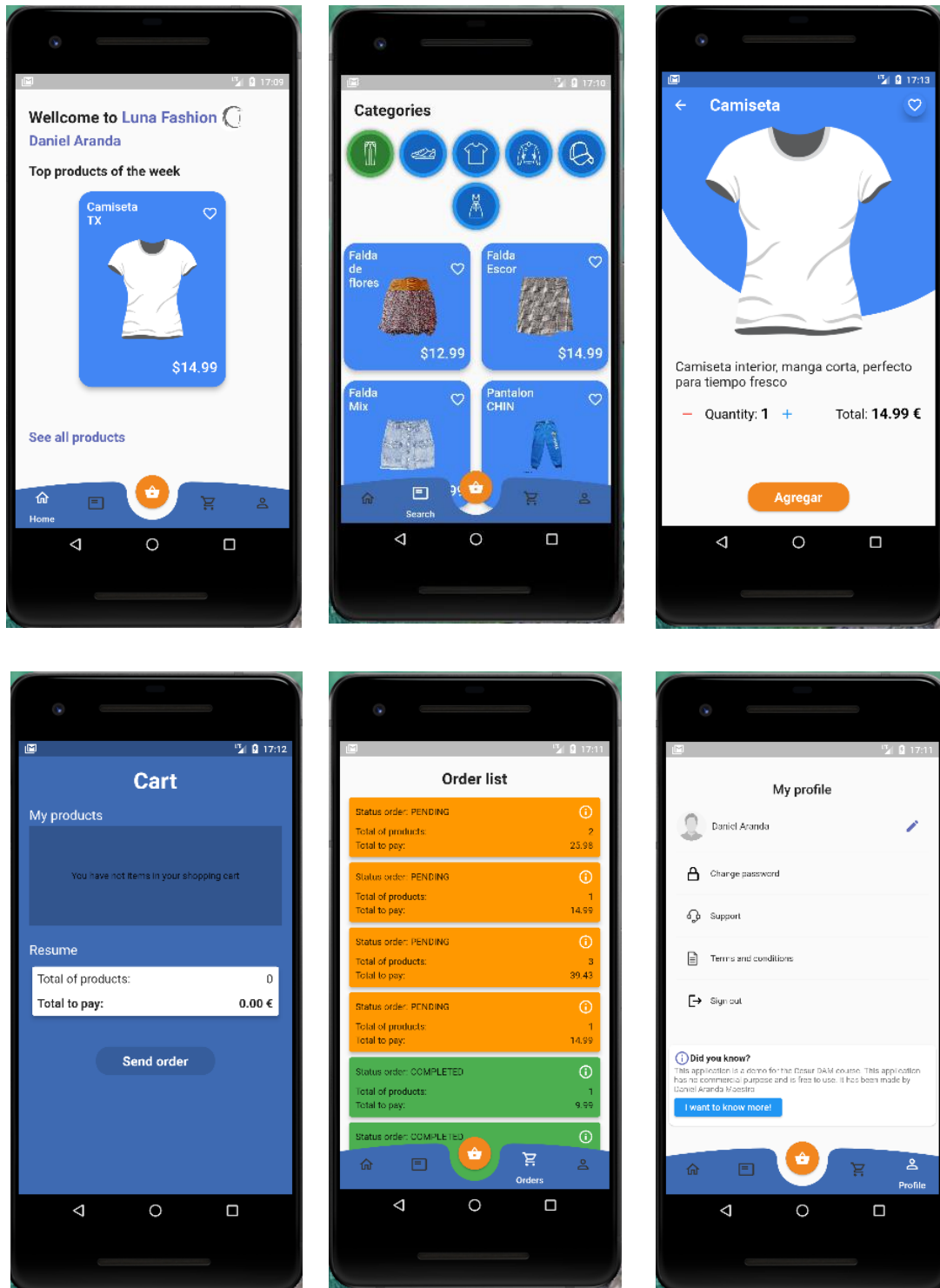
3. Semana 3: Funcionalidad rol de Administrador

Maquetación de las pantallas pertinentes del administrador junto con toda su implementación.



4. Semana 4: Funcionalidad rol de Usuario

Maquetación de las pantallas pertinentes del usuario junto con toda su implementación.



5. Semana 5: Funcionalidad de notificaciones push

Para la implementación de las notificaciones push se creará un servicio de notificaciones push y lo inicializaremos cuando se autentique un Administrador.

Archivo: notification_push_service.dart

```
static Future initialize() async {
  await messaging.requestPermission(
    alert: true,
    announcement: false,
    badge: true,
    carPlay: false,
    criticalAlert: false,
    provisional: false,
    sound: true,
  );
  //Push notifications
  await FirebaseMessaging.instance.subscribeToTopic('ORDERS');

  //Handlers
  FirebaseMessaging.onBackgroundMessage(_backgroundHandler);
  FirebaseMessaging.onMessage.listen(_onMessageHandler);
  FirebaseMessaging.onMessageOpenedApp.listen(_onMessageOpenApp);
}
```

Archivo: home_page.dart

```
@override
Widget build(BuildContext context) {
  final user = ref.watch(userController).user;
  if (user != null &&
    user.role.toString() == UserRoleType.admin().toString()) {
    PushNotificationService.initialize();
    ref.watch(ordersAdminStream);
    ref.watch(productsList);
  }
  if (user != null &&
    user.role.toString() == UserRoleType.user().toString()) {
    ref.watch(ordersList);
    ref.watch(productsStream);
  }
}
```

Así, cuando el usuario realiza el pedido si todo fue satisfactorio se llama al servicio directamente desde la implementación.

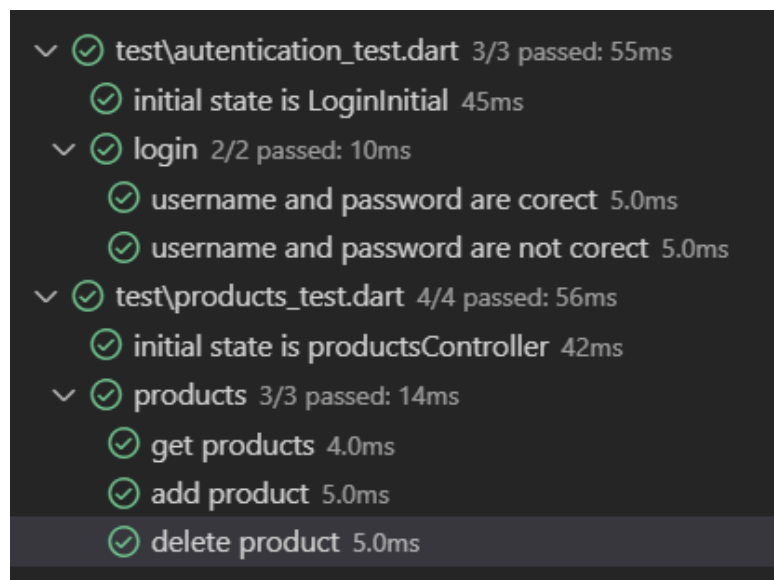
```
    }, SetOptions(merge: true));  
  
    await sendNotification(products.length, user);  
  
    return ResultOr.success();  
  } catch (e, f) {
```

Y al administrador les llegan las notificaciones.



6. Semana 6: Realización de tests y correcciones

Hay que realizar test unitarios para cada controlador que tenemos en nuestro proyecto, además de todas aquellas funciones que se usan. Con esto, se asegura que la aplicación cumplirá con buena calidad y libre de errores. Muy útil para cuando la aplicación va creciendo y se incorporan nuevas funcionalidades. Tras una modificación del proyecto al lanzar los tests podremos verificar si todos los tests continúan funcionando o si algún test ha fallado por alguna razón.



Cabe destacar que en Flutter existen los tests de *Widgets* que nos garantizará que se están mostrando exactamente los elementos que queremos que se muestren por pantalla.

Por último, mencionar que también se podría realizar tests de integración para comprobar que los módulos en su conjunto funcionan correctamente, por ejemplo, un proceso de autenticación, o al navegar a una nueva pantalla esta cargue los datos de los productos.

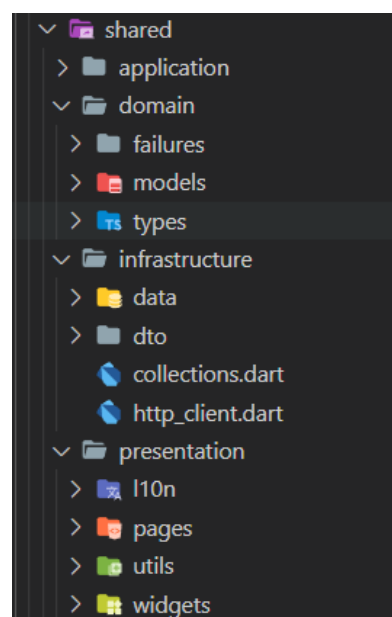
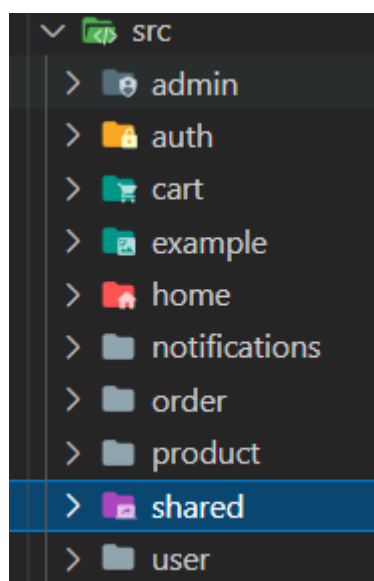
7. Semana 7: Realización de memoria técnica

La realización de una memoria técnica al igual que los manuales de usuario son documentos muy importantes que ayudan a conocer mejor el proyecto y como ha sido su evolución. Es realmente útil sobre todo para la incorporación de nuevas personas al proyecto.

Puntos a destacar del proyecto

Existen varios puntos significativos de cómo ha sido realizado el proyecto, como por ejemplo su estructura de directorios, el uso de los archivos, la inyección de dependencias, los enumeradores, las extensiones, etc....

- La estructura de carpetas: Se distribuyen entre funcionalidades y dentro de cada funcionalidad podemos encontrar la capa de application, domain, infrastructure y presentation. Este tipo de arquitectura se le conoce como *Clean Arquitecture*, distinguir cada capa que forma nuestra aplicación, de esta manera resulta mucho más escalable y mantenible el código.



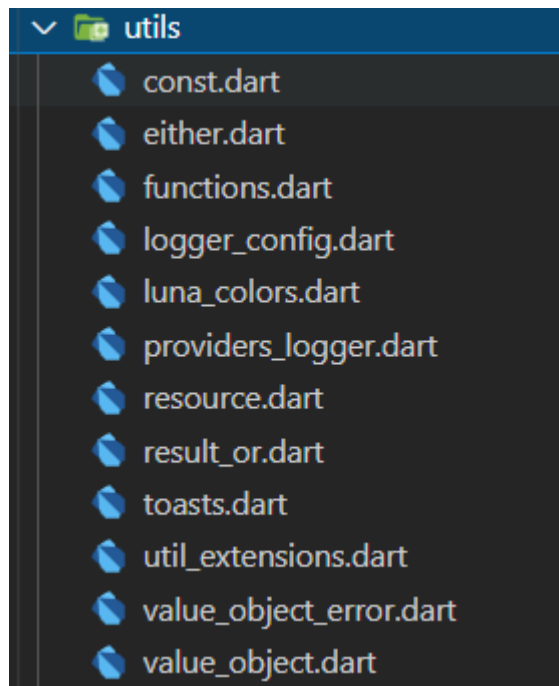
- El uso de unions: Se caracteriza por crear una clase abstracta de lo que vendría siendo los enums, permitiendo utilizar métodos y herencias del mismo.

```
abstract class CategoryProductType {  
    const CategoryProductType();  
    factory CategoryProductType.shoes() = CategoryProductTypeShoes;  
    factory CategoryProductType.tshirts() = CategoryProductTypeTshirts;  
    factory CategoryProductType.pants() = CategoryProductTypePants;  
    factory CategoryProductType.sweatshirts() = CategoryProductTypeSweatshirts;  
    factory CategoryProductType.caps() = CategoryProductTypeCaps;  
    factory CategoryProductType.dresses() = CategoryProductTypeDresses;  
  
    void when({  
        required void Function(CategoryProductTypeShoes) shoes,  
        required void Function(CategoryProductTypeTshirts) tshirts,  
        required void Function(CategoryProductTypePants) pants,  
        required void Function(CategoryProductTypeSweatshirts) sweatshirts,  
        required void Function(CategoryProductTypeCaps) caps,  
        required void Function(CategoryProductTypeDresses) dresses,  
    }) {  
    }
```

- El uso de extensiones: Es una buena forma de crear y utilizar métodos que extienden de otras clases, para que la propia clase no obtenga toda la responsabilidad y no se mezclen capas de nuestra arquitectura.

```
extension DescriptionFailureTranslation on DescriptionFailure {  
    String toTranslation(BuildContext context) {  
        if (this is DescriptionFailureEmpty) {  
            return S.of(context).empty;  
        }  
  
        if (this is DescriptionFailureInvalid) {  
            return S.of(context).invalidEmail;  
        }  
  
        if (this is DescriptionFailureTooLong) {  
            return S.of(context).tooLong;  
        }  
  
        return S.of(context).empty;  
    }  
}
```

- La carpeta utils: Se encuentran todos los archivos de utilidades para nuestro proyecto, como son: las constantes, los loggers, configuraciones, etc...



- El manejo del enrutador: La aplicación cuenta con un generador de rutas al que le hemos asignado todas las rutas personalizadas que nuestra aplicación navegará entre las distintas pantallas.

```
onGenerateRoute: AppRouter.onGenerateRoute,  
initialRoute: splashRoute,
```

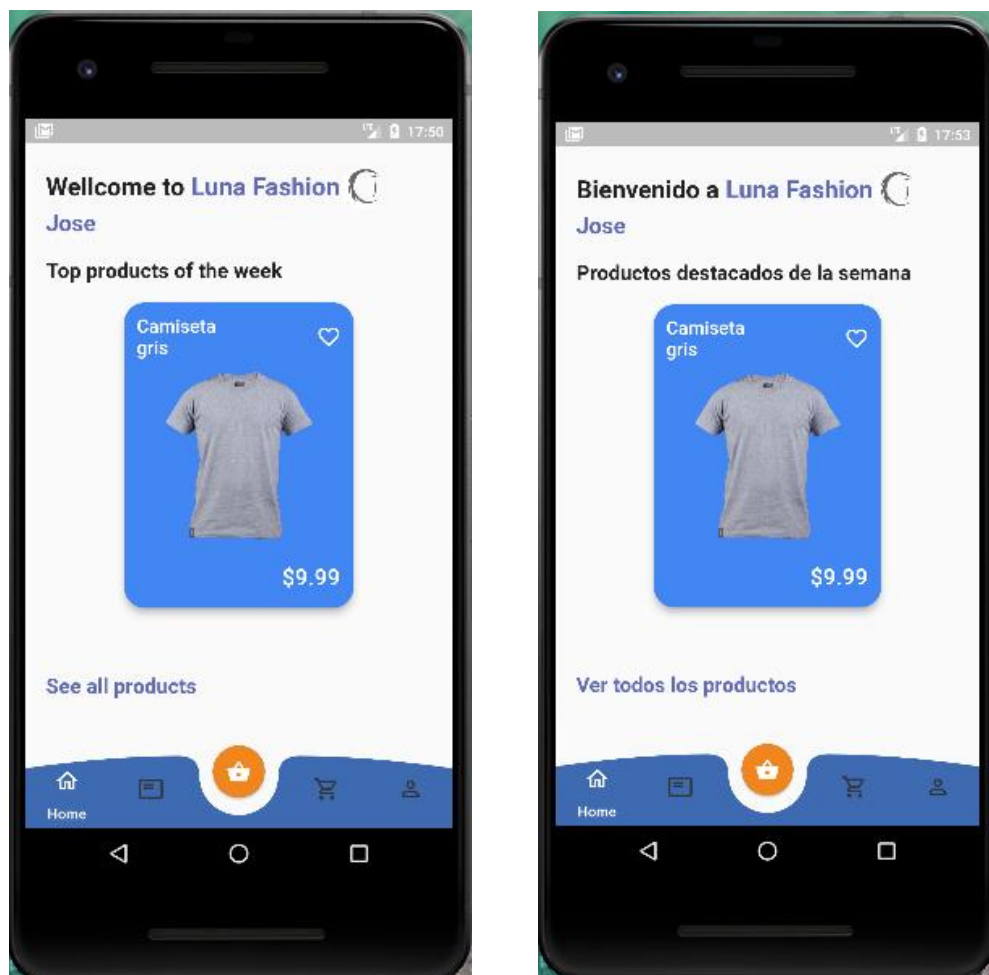
- El manejador de estados: Se usa *Riverpod*, un potente manejador de estados, evolucionado de *Provider*. Una de sus mayores ventajas es en como maneja la inyección de dependencias con solo envolver la clase *MyApp* en una clase especial de *Riverpod*. Además de la facilidad de instanciar los proveedores en los propios widgets, lo que lo hace mucho más limpio que otros manejadores de estados.



```
runZonedGuarded(  
  () => runApp(  
    const ProviderScope(  
      observers: [  
        ProvidersLogger(),  
      ],  
      child: MyApp(),  
    ), // ProviderScope  
  ),  
)
```

- Internalización: El uso de la internalización es algo fundamental si queremos que nuestra aplicación se muestre en diferentes idiomas. Es bastante sencillo de implementar y *visual studio code* dispone de una extensión que lo facilita aún más autogenerando los propios archivos necesarios para ello. Únicamente hay que agregar unas líneas de código en el archivo *pubspec.yaml* y en el *main.dart* que ya mostró anteriormente.

Prueba de internalización en inglés y en español.



- Las reglas de programar: Desde las últimas versiones de Flutter, cuenta con su propio estilo de reglas y recomendaciones para su desarrollo. Se trata el archivo *analysis_options.yaml* en el que además puedes agregar nuevas reglas o ignorar/excluir algunas de ellas. Es una de las cosas más importantes a seguir cuando se trabaja en equipo para que todo el equipo desarrolle de forma uniforme todo el proyecto.

Archivo: analysis_options.yaml

```
# section below, it can also be suppressed for a single
# or a specific dart file by using the `// ignore: name_of_lint`
# `// ignore_for_file: name_of_lint` syntax on the line
# producing the lint.
rules:
  prefer_relative_imports: true
  # avoid_print: false # Uncomment to disable the `avoid_print` rule
  # prefer_single_quotes: true # Uncomment to enable the `prefer_single_quotes` rule
```

Resultado

```
lib > main.dart > ...
You, hace 38 segundos | 1 author (You)
1 import 'dart:async';
2
3 import 'package:firebase_analytics/firebase_analytics.dart';
4 import 'package:firebase_core/firebase_core.dart';
5 import 'package:firebase_crashlytics/firebase_crashlytics.dart';
6 import 'package:flutter/material.dart';
7 import 'package:flutter_localizations/flutter_localizations.dart';
8 import 'package:flutter_riverpod/flutter_riverpod.dart';
9
10 import 'src/shared/presentation/l10n/generated/l10n.dart';
11 import 'src/shared/presentation/utils/logger_config.dart';
12
13 Run | Debug | Profile
14 Future<void> main() async {
```


Fase de pruebas

Cuando la aplicación ya está prácticamente desarrollada comienza la hora de realizar la fase de pruebas. En ella consiste en probar todos los componentes de la aplicación y verificar que todo funciona correctamente.

Desde el comienzo se estuvo realizando test unitarios para comprobar que lo que se iba desarrollando no presentaban errores en el proyecto, dando así una calidad en el proyecto mayor como se ha mostrado anteriormente.

Posteriormente comienzan las pruebas funcionales, se centran en los requisitos que se estableció, verificando así el resultado de una acción. Son muy parecidas a las pruebas de integración pero con más valor esperado.

Las pruebas integrales, se han realizado con varios usuarios ajenos al desarrollador para verificar que el comportamiento del usuario y el flujo de la aplicación funciona de forma correcta y esperada.

La prueba de aceptación fue realizada con el tutor para verificar que la aplicación satisface todos los requisitos mencionados en el comienzo de la memoria y del anteproyecto.

Además, existen más tipos de pruebas como son las pruebas de rendimiento y las de humo, que aunque no se haya utilizado en este proyecto no termina de ser importante para proyectos de mayor envergadura.

Conclusiones

Como conclusión final del proyecto, cabe decir que es una aplicación completa para realizar pedidos de una tienda con su propio backend independiente y realizada con buena arquitectura siguiendo patrones de diseño que lo harán escalable y mantenible de forma muy sencilla.

Posibles mejoras

Una aplicación siempre se puede mejorar, ya sea en la interfaz gráfica del usuario para mejorar la UI (interfaz de usuario) o UX (experiencia de usuario), en agregar nuevas funcionalidades que le den más atractivo a la aplicación, o incluso refactorizar el código mejorando así su legibilidad y su proceso de compilación para obtener mejores rendimientos.

En este proyecto en particular se podrían mejorar los siguientes puntos:

- Mejorar el diseño de la pantalla home del usuario.
- Mejorar el diseño de los ítems de los productos para agregar el precio antiguo.
- Agregar forma de pago al realizar un pedido.
- Agregar la funcionalidad del usuario para editar su perfil y el cambio de contraseña.
- Mejorar el diseño de los pedidos del usuario para mejorar su organización o a través del uso de filtros dependiendo del estado actual del pedido.

Apéndices

Comandos a tener en cuenta para utilizar y/o exportar el proyecto:

- Flutter clean: Borra todos los directorios que autogenera Flutter cuando descargamos las dependencias, además de borrar las carpetas del build del proyecto.

```
Deleting build... 30,9s
Deleting .dart_tool... 237ms
Deleting .packages... 2ms
Deleting Generated.xcconfig... 1ms
Deleting flutter_export_environment.sh... 0ms
Deleting .flutter-plugins-dependencies... 7ms
Deleting .flutter-plugins... 1ms
```

- Flutter pub get: Descarga todas las dependencias necesarias de nuestro proyecto.

```
Running "flutter pub get"
```

- Flutter build apk: Genera un ejecutable para Android. Un detalle a destacar es que a partir del 2021 para subirlo a la tienda de la *Play Store* hay que subirlo en formato bundle. Para ello habría que ejecutar el comando “Flutter build appbundle”.
- Flutter build ios: Genera un ejecutable para iOS. Un detalle a destacar es que no es posible compartir el ejecutable, para ello es necesario utilizar la herramienta de Apple que se llama *TestFlight*, donde será necesario agregar al usuario como tester para poder hacer uso y probar la aplicación.

Para instalar la aplicación sólo es necesario instalar el ejecutable de la aplicación y comenzar a usarlo.

Para el usuario, la aplicación resulta muy sencilla e intuitiva, nada más que abre la aplicación basta con realizar el registro y ya se autentica de forma automática donde en la misma pantalla del home, podrá agregar los productos al carrito y realizar los pedidos.

Además, desde el perfil del usuario, si presiona el botón de “¡Quiero saber más!” navegará a la página web de Daniel Aranda Maestro, donde podrá obtener más información sobre su profesión y sus proyectos.

Referencias bibliográficas

Flutter:

<https://esflutter.dev/>

Gestión de paquetes oficiales de Flutter:

<https://pub.dev/>

Uso de Internalización para Flutter:

<https://docs.flutter.dev/development/accessibility-and-localization/internationalization>

Uso del gestor de estados Riverpod:

<https://medium.com/flutter-community/riverpod-a-deep-dive-on-the-surface-e12a0559bcf5>

LottieFiles:

<https://lottiefiles.com/>

Firebase:

<https://firebase.google.com/?hl=es>

Documentación oficial de Firebase para Flutter:

<https://firebase.flutter.dev/>

Código de errores de Firebase:

<https://firebase.google.com/docs/reference/js/v8/firebase.auth.Auth#createuserwithemailandpassword>

Registro con Firebase:

<https://firebase.flutter.dev/docs/auth/usage/>