



POMELO'S

ÍNDICE

Contenido

ÍNDICE	1
Contenido	1
1 Introducción	2
1.1 Personal involucrado	2
2 Descripción general	2
2.1 Perspectiva de la aplicación	2
2.2 Alcance	2
2.3 Funcionalidad de la aplicación	3
2.4 Evolución previsible del sistema	6
2 Requisitos específicos	7
3.1 Base de datos (SQLite)	7
3.2 Requisitos funcionales	10
BIBLIOGRAFÍA	11



1 Introducción

Este documento resume las especificaciones de la aplicación en Android Pomelo's . La aplicación está enfocada en el muestreo de prendas de vestir y por ello la interfaz estará inspirada en la interfaz de Instagram porque es una interfaz muy estandarizada por el público joven además de ser intuitiva y muy visual.

1.1 Personal involucrado

Antonio Martínez Clemente

Fernando Cárdenas Bravo

Roberto Cerezo Moreno

2 Descripción general

2.1 Perspectiva de la aplicación

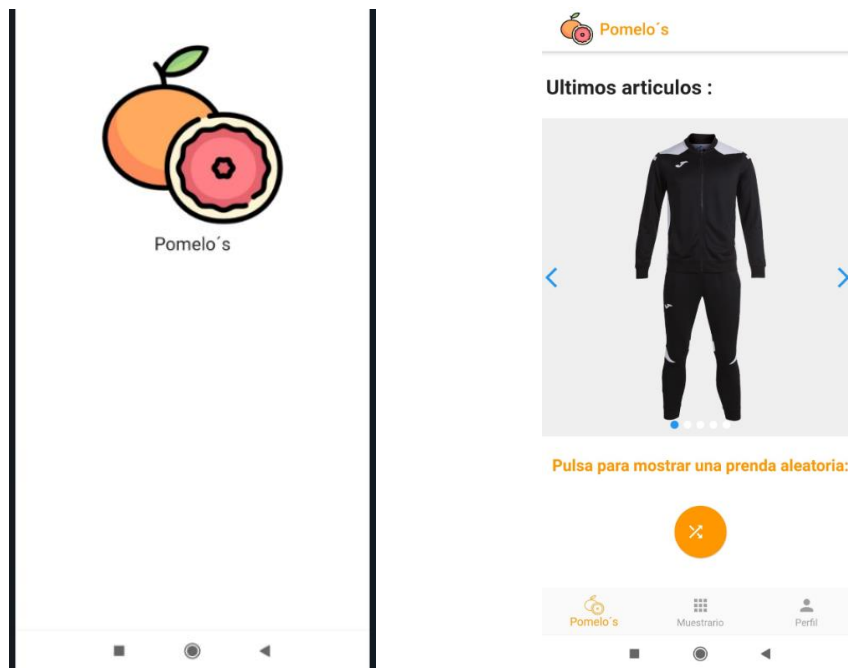
Es un producto plenamente funcional y no necesita de aplicaciones de terceros para poder desarrollar un correcto funcionamiento. Aunque para desarrollar la aplicación hemos utilizado librerías de terceros como SQLite (utilizada para el desarrollo de la creación de la base de datos local que contiene la aplicación)

2.2 Alcance

Esta aplicación está enfocada a pequeños comercios y artistas, con la aplicación se podrán promocionar y darse a conocer. Para los usuarios de la aplicación será fácil de utilizar ya que es una interfaz familiar y accesible.



2.3 Funcionalidad de la aplicación



Al iniciar la aplicación lo primero que vemos será la pantalla de carga de la izquierda.

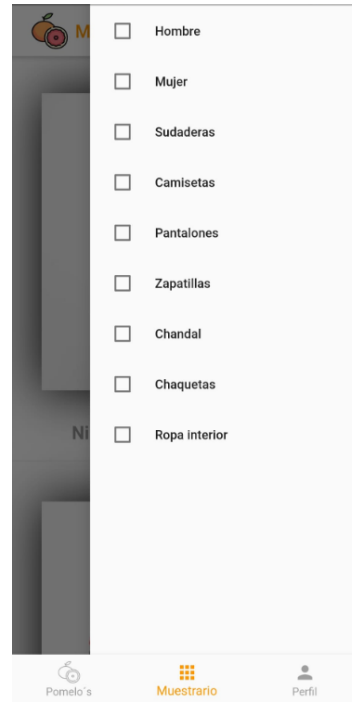
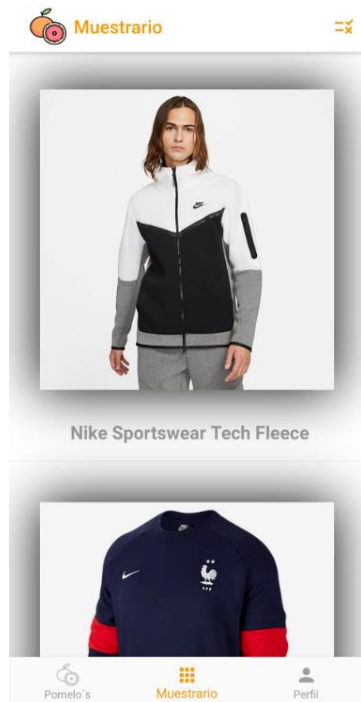
Después veremos la página principal de la aplicación. Esta pestaña consta de dos partes, la parte de arriba en la cual veremos 5 prendas que rotan automáticamente o podremos desplazarnos con las flechas que podemos ver a izquierda y derecha. La segunda parte de la página es el botón naranja que se sitúa en la parte inferior, este botón nos redireccionará a una prenda aleatoria. Esta funcionalidad es muy interesante ya que al mostrarnos una prenda al azar podríamos fijarnos en prendas que no visualizaríamos en un caso normal.

Como podemos ver en la parte inferior de la página principal (imagen izquierda) se sitúa el menú de navegación.

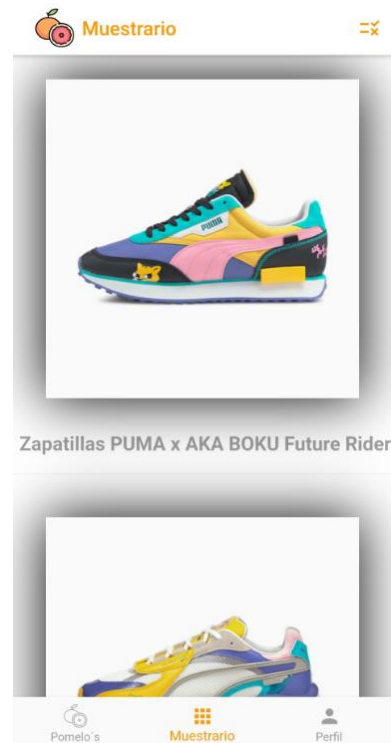
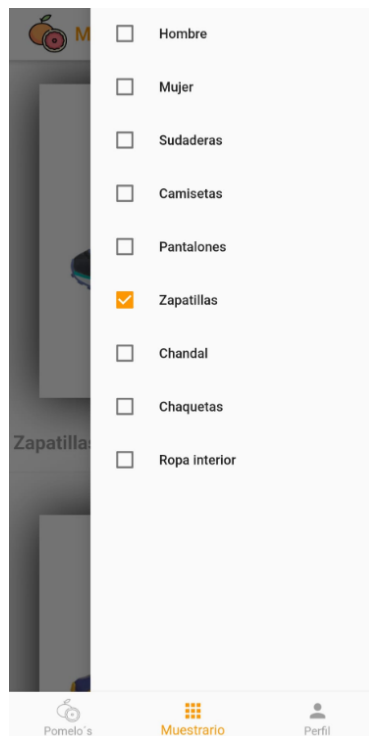


El primer símbolo corresponde a la página principal de la aplicación, el segundo corresponde al muestrario el cual contienen todas las prendas y por último tenemos el perfil el cual contendrá todas las prendas que hayamos seleccionado como favoritas.





Estas dos imágenes corresponden al muestrario. Para navegar por ella tendremos que deslizar hacia arriba y nos irán saliendo diferentes prendas. Como podemos ver esta página contiene distintos filtros los cuales aparecerán si pulsamos el botón situado en la parte superior derecha. Al activar estos filtros cambiará nuestra página de muestreo apareciendo únicamente las prendas seleccionadas.



Si seleccionamos una de las imágenes del muestrario nos redireccionará a la página de detalle de la prenda.

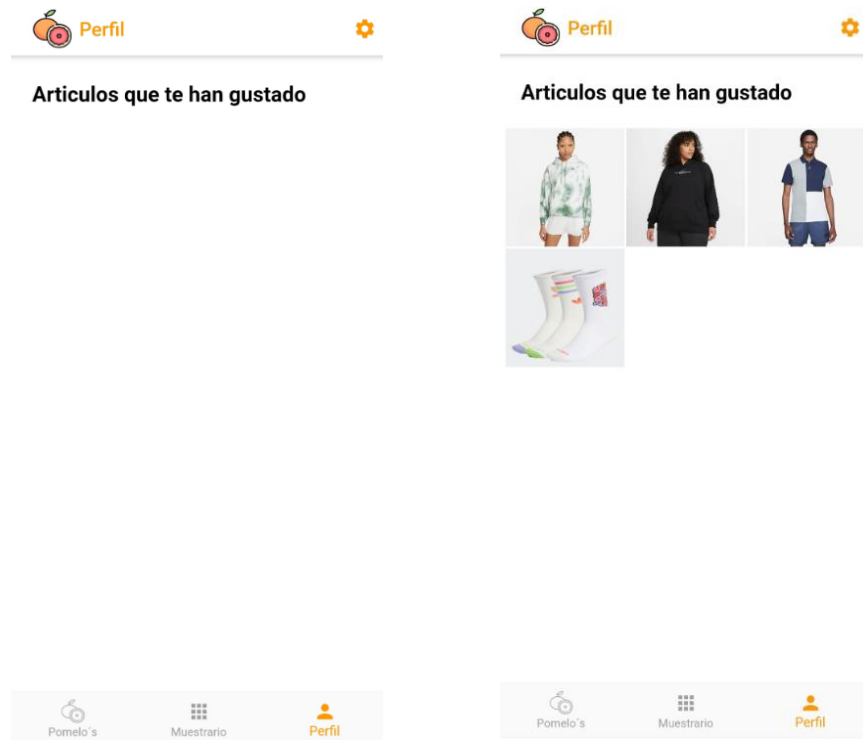


En esta página podemos consultar toda la información de la prenda, podemos ver la descripción de la prenda y en la parte inferior contiene un link el cual nos redireccionará a la página web donde podemos comprar la prenda.

En la parte superior izquierda tenemos un símbolo en forma de corazón, con este botón podemos seleccionar la prenda en favoritos (en la imagen de la izquierda la prenda estaría seleccionada como favorita).

Podemos ver también una flecha por encima del corazón la cual nos redireccionará a la página anterior.





Aquí vemos la página del perfil que consta de un área donde podemos ver todas las prendas previamente marcadas como favoritas. También podemos acceder a estas fotos pulsándolas para quitarlas de favoritos o ver la información.

2.4 Evolución previsible del sistema

La evolución lógica de este sistema es terminar siendo una aplicación de venta de ropa, siempre intentándonos enfocar en los pequeños comercios, añadiendo una pasarela de pago con un API como el de PayPal se podría vender las prendas con facilidad y seguridad. Evidentemente si esto acabara pasando habría que desarrollar un método de encriptar toda esa información para que el sistema fuera seguro.











3 Requisitos específicos

La aplicación está desarrollada en Flutter un lenguaje desarrollado por Google. También hemos contado con librerías de terceros como SQLite con la cual hemos desarrollado la base de datos local que contiene la aplicación.

Para poder ejecutar la aplicación es necesario tener el SDK de Flutter y de Dart configurado. También tenemos que obtener los paquetes a través del terminal escribiendo *flutter packages get*. Por ultimo nos dirigiremos a la clase *pubspec.yaml* y pulsaremos *Pub get* y *Pub udgrade*.

3.1 Base de datos (SQLite)

La base de datos consiste en una tabla prendas con varios campos.

Students			
	ID	integer	
	CATEGORIA	integer	
	DESCRIPCION	text	
	NOMBRE	text	
	LINK	text	
	FAVORITO	integer	
	GENERO	integer	

El campo Categoría es un número, relacionando un numero con una categoría

- 1.Sudadera
- 2.Camiseta
- 3.Pantalones
- 4.Zapatillas
- 5.Chándal
- 6.Ropa Interior

El campo favorito también es numérico siendo 0 no favorito y 1 favorito. De manera parecida funciona genero siendo 0 Hombre, 1 Mujer y 3 Unisex.

Para la inicialización de la base de datos tenemos este método




```
Future<Database> ini() async {
  Future<Database> datos = openDatabase(
    join(await getDatabasesPath(), 'prendas_database.db'),

    onCreate: (datos, version) {

      return datos.execute(
        "CREATE TABLE prendas(id INTEGER PRIMARY KEY, nombre TEXT, categoria INTEGER,descripcion TEXT,link TEXT,favorito INTEGER,genero INTEGER)",
      );
    },

    version: 1,
  );
}
```

Este método inicia la base de datos si en ese dispositivo es la primera vez que se ejecuta la aplicación crearía la tabla prendas.

Sino simplemente el método carga la base de datos.

```
Future<Database> cargar() async{
  Future<Database> db = openDatabase(

    join(await getDatabasesPath(), 'prendas_database.db')
  ,);
  return db;
}
```

También tenemos el método cargar que pasaremos a otros métodos para poder cargar la base de datos.

```
Future<List<Prenda>> obtenerdatos() async {

  Database db = await cargar();
  final List<Map<String, dynamic>> maps = await db.query('prendas');
  return List.generate(maps.length, (i) {
    return Prenda(maps[i]['id'], maps[i]['categoria'], maps[i]['descripcion'], maps[i]['favorito'], maps[i]['genero'], maps[i]['nombre'], maps[i]['link']);
  }); // List.generate
}
```

El método obtener datos obtiene todas las filas de la tabla prendas y las retorna en forma de lista. Con este método sacaremos todas las prendas en el muestrario.



```

void rellenar() async{
    prendas=await obtenerdatos();
}

Future<void> updatePrenda(Prenda p) async {
    // Obtiene una referencia de la base de datos
    Database u = await cargar();

    // Actualiza el Dog dado
    await u.update(
        'prendas',
        p.toMap(),

        where: "id = ?",
        whereArgs: [p.id],
    );
}

Future<void> insertPrenda(Prenda p) async {

    Database db = await cargar();

    await db.insert(
        'prendas',
        p.toMap(),
        conflictAlgorithm: ConflictAlgorithm.ignore,
    );
}

```

El método rellenar inserta todas las prendas en una lista la encargada de guardar las prendas en distintas posiciones y pudiendo acceder a ellas fácilmente.

UpdatePrenda es un método encargado de editar una prenda en la base de datos para obtener la persistencia que buscamos para ello se la tendremos que pasar como parámetro (ejemplo: guardar las prendas en favoritos).

Y por último insertPrenda el cual le pasamos una Prenda como parámetro y insertaremos esta prenda como parámetros. Si el id de la prenda es el mismo la prenda no se sobrescribirá.



```

List<Prenda> tFav(){
    rellenar();
    List<Prenda> fav=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].favorito==1){
            fav.add(prendas[i]);
        }
    }
    return fav;
}

List<Prenda> fHom(){
    List<Prenda> hom=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].genero==0){
            hom.add(prendas[i]);
        }
    }
    return hom;
}

List<Prenda> fMuj(){
    List<Prenda> mu=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].genero==1){
            mu.add(prendas[i]);
        }
    }
    return mu;
}

List<Prenda> fSud(){
    List<Prenda> sud=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==1){
            sud.add(prendas[i]);
        }
    }
    return sud;
}

List<Prenda> fcam(){
    List<Prenda> cam=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==2){
            cam.add(prendas[i]);
        }
    }
    return cam;
}

List<Prenda> fpan(){
    List<Prenda> pan=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==3){
            pan.add(prendas[i]);
        }
    }
    return pan;
}

List<Prenda> fzap(){
    List<Prenda> zap=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==4){
            zap.add(prendas[i]);
        }
    }
    return zap;
}

List<Prenda> fcha(){
    List<Prenda> cha=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==5){
            cha.add(prendas[i]);
        }
    }
    return cha;
}

List<Prenda> fri(){
    List<Prenda> ri=<Prenda>[];
    for (var i = 0; i < prendas.length; i++) {
        if(prendas[i].categoria==6){
            ri.add(prendas[i]);
        }
    }
    return ri;
}

```

Estos métodos son los encargados de obtener sublistas las cuales se utilizaran para cargar las prendas de los filtros seleccionados en el muestrario y las favoritas en el perfil.



3.2 Requisitos funcionales

```
class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return new SplashScreen(
      seconds: 5,
      navigateAfterSeconds: new App(),
      image: new Image.asset('assets/images/pomelo.png'),
      title: Text(
        "Pomelo's",
        style: TextStyle(fontSize: 20),
      ), // Text
      backgroundColor: Colors.white,
      loaderColor: Colors.transparent,
      photoSize: 100,
    ); // SplashScreen
  }
}
```

Este fragmento de código pertenece a la creación de la pantalla de carga al inicio de la aplicación.

```
void meGusta(bool newValue,int num) => setState(() {
  megusta = newValue;
  if (megusta==true){
    a.todas[num].favorito=1;
    a.updateDog(a.todas[num]);
  }else{
    a.todas[num].favorito=0;
    a.updateDog(a.todas[num]);
  }
});
```

Gracias a este método al pulsar en el botón de favoritos se cambia el valor de favorito en la base de datos para poder mostrar la prenda en la página del perfil las prendas marcadas como favoritas.



```

class Swiper5 extends StatelessWidget {
  final List<Producto> todos;
  Swiper5({Key key, @required this.todos}) : super(key: key);

  Widget build(BuildContext context) {
    return Swiper(
      itemBuilder: (BuildContext context, int index) {
        return InkWell(
          child: Container(
            child: Image.asset('assets/images/pic${productos[index]}.jpeg'),
          ), // Container
          onTap: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => DetailScreen(productos[index], data),
              ), // MaterialPageRoute
            );
          },
        ); // InkWell
      },
      autoplay: true,
      itemCount: 5,
      //layout: SwiperLayout.STACK,
      scrollDirection: Axis.horizontal,
      pagination: new SwiperPagination(),
      control: new SwiperControl(),
    ); // Swiper
  }
}

```

Este código forma parte de la página del home de la aplicación. Pertenece al swiper que muestra los últimos artículos.

```

Container(
  padding: EdgeInsets.only(top: 30),
  alignment: Alignment.center,
  width: double.maxFinite,
  child: Text(
    data.todos[id].nombre,
    style: TextStyle(fontSize: 20, color: Colors.black38, fontWeight: FontWeight.w700),
    textAlign: TextAlign.center,
  ), // Text
), // Container
Container(
  padding: EdgeInsets.only(top: 30, left: 15, right: 15),
  alignment: Alignment.centerLeft,
  width: double.maxFinite,
  child: Text(
    "Descripción: " + data.todos[id].descripcion,
    style: TextStyle(fontSize: 15, color: Colors.black, fontWeight: FontWeight.w400),
    textAlign: TextAlign.start,
  ), // Text
), // Container

```



```

Container(
  padding: EdgeInsets.only(top: 30, left: 15, right: 15, bottom: 50),
  alignment: Alignment.center,
  width: double.maxFinite,
  child: RichText(
    text: TextSpan(
      children: [
        TextSpan(
          style: TextStyle(fontSize: 15, color: Colors.black, fontWeight: FontWeight.w400),
          text: "Link: ",
        ), // TextSpan
        TextSpan(
          style: TextStyle(fontSize: 15, color: Colors.blue, fontWeight: FontWeight.w400, decoration: TextDecoration.underline),
          text: "Ir a la página",
          recognizer: TapGestureRecognizer().onTap = () async {
            var url = data.todos[id].link;
            if(await canLaunch(url)){
              await launch(url);
            } else{
              throw "No se puede cargar la url";
            }
          }
        ) // TextSpan
      ]
    ) // TextSpan
  ), // RichText
), // Container

```

En estos dos fragmentos de código aparecen la creación de los container del nombre del artículo, de la descripción de este y la creación del campo del link, el cual se obtiene desde la lista que contiene todas las prendas almacenadas.



BIBLIOGRAFÍA

FLUTTER: <https://flutter.dev/docs>

SQLITE: <https://sqlite.org/docs.html>

SQLITE EN FLUTTER: <https://esflutter.dev/docs/cookbook/persistence>

DART: <https://dart.dev/guides>

