



Projecte Final de Cicle

MercaLlistes



Alumne: Rubén Altur Bononad
DNI: 20945286Z
Tutor Individual: Joan Gerard Camarena Estruch
Tutor del grup: José Alfredo Murcia Andres

Dades del Projecte

Dades de l'alumne

Nom i cognoms	Rubén Altur Bononad
NIF/NIE	10014033
Curs i CF	2n DAM

Dades del projecte

Títol del projecte	MercaLlistes
Nom del tutor individual	Joan Gerard Camarena Estruch
Nom del tutor del grup	José Alfredo Murcia Andres
Resum	MercaLlistes és una aplicació basada en Flutter per al frontend i amb una base de dades a Firebase per a emmagatzemar les dades de llistes de la compra separades per supermercats per a facilitar la gestió de les compres
Abstract	MercaLlistes is an application based on Flutter for the frontend and it uses a database in Firebase for saving and managing the data of different shopping lists separated by supermarkets to help in the management of the purchase of goods
Mòduls implicats	<ul style="list-style-type: none">● Accés a Dades● Programació multimèdia i dispositius mòbils● Programació de Serveis i Processos● Disseny d'Interfícies
Data de presentació	

Índex

1. Introducció del project.	4
1.1 Descripció i tipus de projecte	4
1.2 Objectius	4
1.3 Orientacions per al desenvolupament i recursos	4
2. Disseny de la solució	6
2.1 Diagrama de comportament	6
2.2 Persistència: Diagrames Entitat-Relació. Pas a taules.	8
2.3 Planificació temporal i pressupost	9
3. Desenvolupament de la solució	9
4. Implantació de la solució	34
5. Conclusions i treballs futurs	35

1. Introducció del projecte

1.1 Descripció i tipus de projecte

MercaLlistes és una aplicació per a Android i per a escriptori, amb l'objectiu de tindre una forma d'organitzar les diferents llistes de la compra al hora d'anar a fer la compra als supermercats. Aquesta informació està emmagatzemada a una base de dades a Firebase i es pot interactuar amb ella mitjançant la interfície d'usuari feta amb Flutter.

Aquesta idea va ser proposada per el meu tutor de grup José Alfredo i a la que vaig accedir després de preguntar-li a varios coneguts com s'organitzen al hora d'anar a fer compres mitjanes i grans als supermercats.

1.2 Objectiu

El principal objectiu de l'aplicació és facilitar l'organització dels usuaris quan siga l'hora d'anar a comprar béns, i fer-ho d'una forma fàcil i atractiva per al usuari.

1.3 Orientacions per al desenvolupament i recursos

Les tecnologies i ferramentes que he utilitzat al llarg del projecte estan enumerades al següent llistat:

- **VisualStudio Code:** El meu IDE predeterminat per a desenvolupar aplicacions per la seua facilitat al hora de administrar extensions i llenguatges, en ell també he fet els testos a l'aplicació.



- **Github:** El repositori on he anat guardant el progres de l'aplicació i m'ha habilitat el poder treballar amb diferents dispositius simultàniament.



- **Firebase:** La plataforma en el núvol en la que he guardat la meua base de dades. Aquesta plataforma treballa amb col·leccions de forma pareguda a la que es gasta en els JSON, he triat aquesta perquè es molt fàcil d'entendre i de utilitzar i perquè al estar desenvolupada per Google tenia connexió directa amb la tecnologia que he gastat per al frontend.



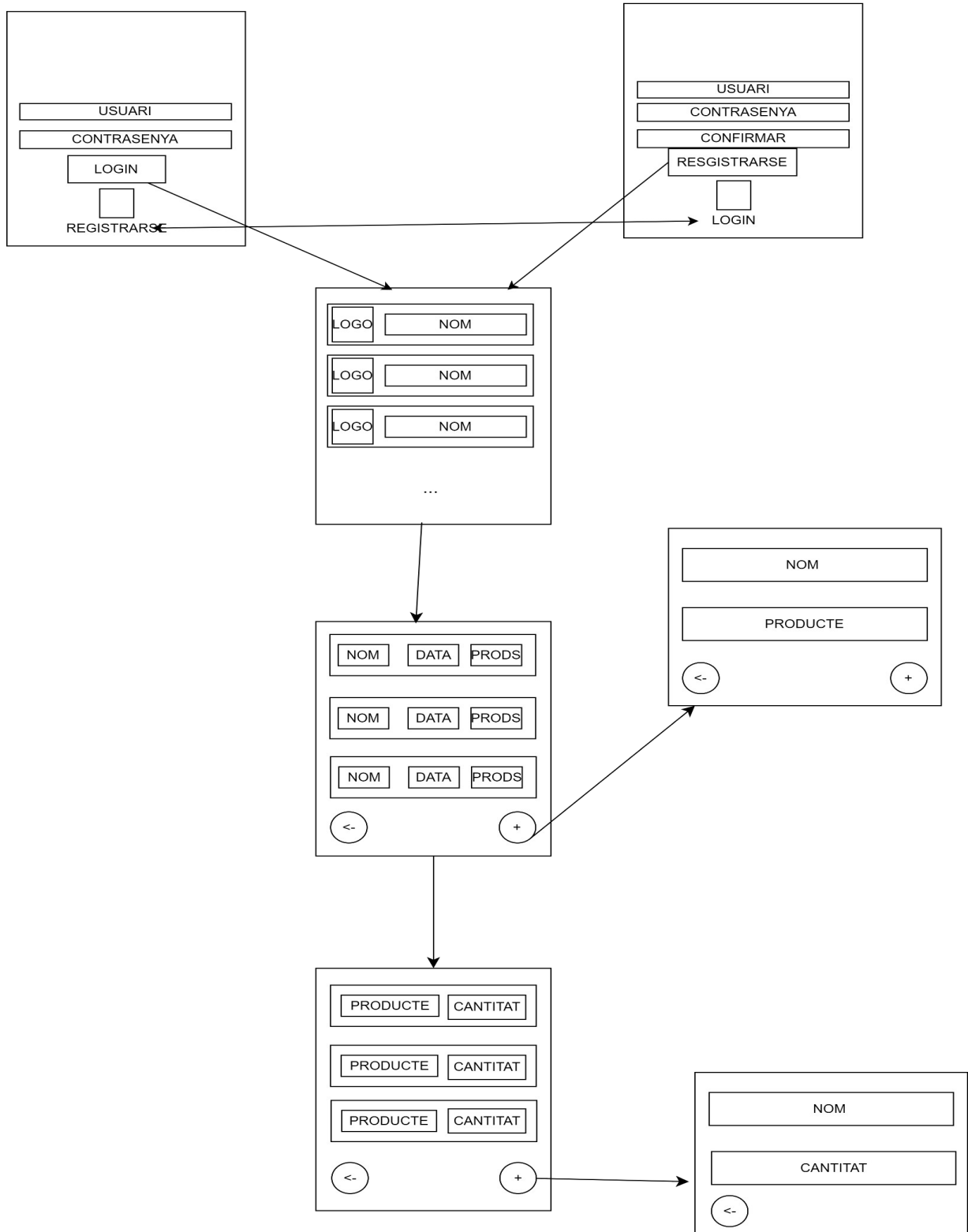
- **Flutter:** Finalment he triat Flutter per al frontend per 2 motius: És un framework amb el que estic familiaritzat al haver estat gastant-lo durant tot el segon trimestre al modul de PMDM i perquè, com bé he dit al punt anterior, està desenvolupat per Google el qual facilita molt la connexió amb la base de dades a Firebase.

2. Disseny de la solució

2.1 Diagrama de comportament

MercaLlistes està basada amb el següent esquema en ment:

- Les vistes de Log in i de registre, son les primeres vistes que apareixen quan entres a l'aplicació per primera volta, en aquestes pots fer in log in a la app o registrar-te, pots fer-ho amb usuari i contrasenya o bé amb un compte de google.
- La vista HOME en la que apareixen els diferents supermercats que hi han guardats a l'aplicació amb el corresponent logo i el número de llistes de la compra que hi han per acabar.
- La vista de selecció de llista on s'accedeix després de clicar a un dels supermercats, a aquesta vista s'observen les diferents llistes emmagatzemades a eixe supermercat, es pot veure també quants productes li queden per comprar, a quina data es van crear i quin nom li ha donat l'usuari. En aquesta vista es pot observar un botó que obri un formulari per a crear una llista nova a la que se li assigna un nom, una data i els diferents productes que es van a comprar.
- La vista de productes que apareix al clicar en la llista corresponent, en aquesta pantalla es mostraran els diferents productes que estan a la llista i la quantitat d'aquests, també s'observarà si el producte està comprat o no perquè aquest estarà taxat després de clicar en ell. En aquesta vista, igual que al anterior, també apareixerà un botó per a poder afegir productes en cas de haver-se-li oblidat al usuari.



2.2 Persistència: Firebase

Al utilitzar Firebase que és una base de dades noSQL i de la forma en la que estan guardades les dades sols hi ha col·lecció en la que s'emmagatzemen tots els documents que guarden les llistes. La col·lecció té la següent estructura:

```
{
  "id": 1,
  "nom": " ",
  "supermercat": " "
  "comprada": false,
  "data": " ",
  "usuari": " ",
  "items": [
    {
      "comprat": false,
      "nom": " ",
      "cantitat": " "
    }
  ],
}
```

Aquesta estructura comptarà: amb un id que serà igual al uid del document a la base de dades per a facilitar el treballar amb ell, un nom que li proporcionarà el usuari al hora de crear la llista, el nom del supermercat que s'assignarà automàticament quan es crea la llista, la data de creació de la llista, un booleana per a saber si està la llista comprada o no i finalment un array on es guardaran els items o productes que contindrà la llista i els seus valors com quantitat, si està comprat o no, o el nom del producte.

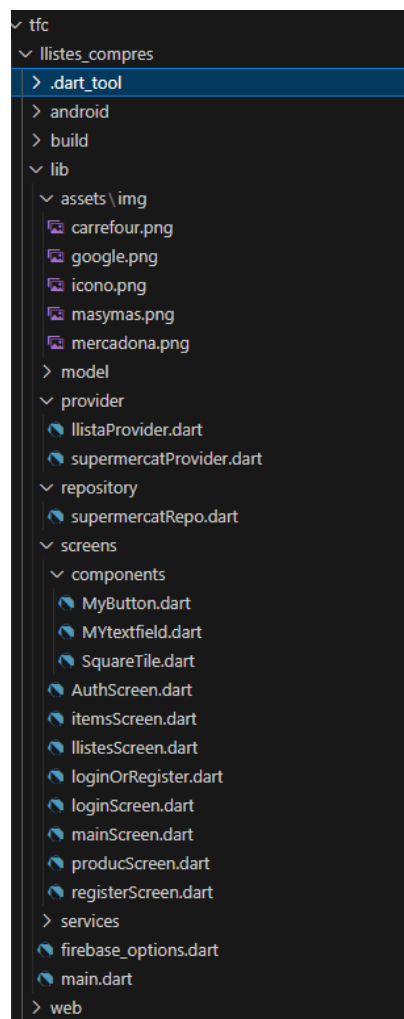
2.3 Planificació temporal

Per al projecte he treballat sense tindre objectius marcats o tasques, he funcionant avançant poc a poc cada dia després de les FCTs, el ordre ha sigut primer pensar i formular la estructura de dades i crear-la a Firebase, després fer un frontend en base aquesta i finalment fer una documentació del projecte mitjançant la memòria.

3 Desenvolupament de la solució

3.1 FLUTTER

Ací podem observar l'estructura de l'aplicació de flutter, dins de la carpeta lib està tot el codi, separat per carpetes segons quina classe de component siga, a banda també estan les carpetes de android i web per a poder executar l'aplicació en eixes plataformes



Començarem per la classe Fireservice que és la que gestiona totes les interaccions amb la base de dades amb la llibreria cloud_firestore, ací podem veure el get per a obtenir totes les llistes a la base de dades.

```
class Fireservice {
    FirebaseFirestore firestore = FirebaseFirestore.instance;

    Future<List<Map<String, dynamic>>> getLlistes() async {
        try {
            CollectionReference collection = firestore.collection("llistes");
            QuerySnapshot querySnapshot =
                await collection.get() as QuerySnapshot<Object?>;
            print(querySnapshot);

            List<Map<String, dynamic>> data;

            data = querySnapshot.docs.map((doc) {
                Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
                return data;
            }).toList();
            print(data);
            return data;
        } catch (e) {
            print("Error al obtener los documentos: $e");
            return [];
        }
    }
}
```

Seguidament ací es poden observar la resta d'operacions a un CRUD comú:

```
Future<void> addLlista(Map<String, dynamic> llista) async {
    try {
        await firestore.collection('llistes').add(llibra);
    } catch (e) {
        //
    }
}

Future<void> updateCamp( String id, String camp, Llista valorNou) async {
    try {
        await firestore.collection("llistes").doc(id).set(valorNou.toMap());
    } catch (e) {
        //
    }
}

Future<void> borrarColeccio(String colleccio, dynamic id) async {
    try {
        await firestore.collection(colleccio).doc(id).delete();
    } catch (e) {
        print("Error al actualizar el documento: $e");
    }
}
```

Per a utilitzar les funcions anteriors ho farem mitjançant el Provider que he utilitzat a tota la aplicació, aquest guardarà la llista actual amb la que estem treballant, el supermercat actual i retornarà la llista de mapes que rep del Fireservice:

```
class Llistaprovider with ChangeNotifier {  
  final fs = fireservice();  
  List<Map<String, dynamic>>? _llistaEnv;  
  String? _llistaActual = "";  
  String? superMercatActual="";  
  
  Llistaprovider() {  
    _carregaLlistes();  
  }  
  
  void _carregaLlistes() async{  
    List<Map<String, dynamic>>? jsonLlista= await fs.getLlistes();  
    _llistaEnv=jsonLlista;  
    notifyListeners();  
  }  
  
  void _carregaLlistesC(dynamic data) async{  
    List<Map<String, dynamic>> jsonLlista= await fs.getLlistes();  
    _llistaEnv=jsonLlista;  
    notifyListeners();  
  }  
  
  String get llistaActual {  
    return _llistaActual!;  
  }  
  
  List<Map<String, dynamic>>? get llistaEnv {  
    return _llistaEnv;  
  }  
  
  void addLlista(Llista llist){  
    Map<String, dynamic> llistaAPujar= llist.toMap();  
    // print(llistaAPujar);  
    fs.addLlista(llistaAPujar);  
    _carregaLlistes();  
    notifyListeners();  
  }  
  
  void updateLlista( String Id, String camp, dynamic valorNou){  
    fs.updateCamp( Id, camp, valorNou);  
    _carregaLlistes();  
    notifyListeners();  
  }  
}
```

I amb açò anem a començar a veure les vistes que utilitzarà l'aplicació. La primera es la de AuthScreen aquesta comprovarà si hi ha un usuari logejat al moment, si no hi ha ningun retornarà la pantalla de LoginOrRegister i en cas de que si que ho estigui retornarà la mainScreen

```
class AuthScreen extends StatelessWidget{  
  
  AuthScreen({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: StreamBuilder<User?>(  
        stream: FirebaseAuth.instance.authStateChanges(),  
        builder:(context, snapshot){  
          if(snapshot.hasData){  
            return mainScreen();  
          }  
          else{  
            return LoginOrRegisterScreen();  
          }  
        }  
      ), // StreamBuilder  
    ); // Scaffold  
  }  
}
```

La pantalla de LoginOrRegister es, igual que la AuthScreen, una pantalla transitòria, aquesta et redirigeixi a la pantalla de Login o la de registrar-se segons el estat de la variable booleana showLoginPage que per defecte està a true, a banda aquesta pantalla es la que gastarem per a canviar entre la de login i la de registre a la interfície gracies a la funció togglePages que posarà en true o en false la variable anteriorment mencionada cada volta que es cliqui el boto de canviar de pantalla.

```
class LoginOrRegisterScreen extends StatefulWidget {  
  const LoginOrRegisterScreen({super.key});  
  
  @override  
  State<LoginOrRegisterScreen> createState() => _LoginOrRegisterScreenState()  
}  
  
class _LoginOrRegisterScreenState extends State<LoginOrRegisterScreen> {  
  bool showLoginPage = true;  
  
  void togglePages() {  
    setState(() {  
      showLoginPage = !showLoginPage;  
    });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    if (showLoginPage) {  
      return Loginscreen(  
        onTap: togglePages,  
      );  
    } else {  
      return Registerscreen(  
        onTap: togglePages,  
      );  
    }  
  }  
}
```

Abans de mostrar les pantalles cal dir que al tindre varius widgets personalitzats que es repeten a l'aplicació els he separat a classes guardades a la carpeta components, que son els següents:

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

class MYtextfield extends StatelessWidget {
  final controller;
  final String hintText;
  final bool amagarText;

  const MYtextfield({
    super.key,
    required this.controller,
    required this.hintText,
    required this.amagarText});

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(horizontal: 25),
      child: TextField(
        controller: controller,
        obscureText: amagarText,
        decoration: InputDecoration(
          enabledBorder: OutlineInputBorder(
            borderSide: BorderSide(color: Colors.black), // Outl
          focusedBorder: OutlineInputBorder(
            borderSide: BorderSide(color: Colors.grey), // Outl
          hintText: hintText), // InputDecoration
        )); // TextField // Padding
  }
}
```

```
class MyButton extends StatelessWidget {
  MyButton({super.key, this.onTap});

  final Function()? onTap;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        padding: EdgeInsets.all(25),
        margin: EdgeInsets.symmetric(horizontal: 25),
        decoration: BoxDecoration(
          color: Color.fromARGB(255, 224, 106, 59),
          borderRadius: BorderRadius.circular(8)), //
      child: Center(
        child: Text(
          "Login",
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.bold,
            fontSize: 16,
          ), // TextStyle
        ), // Text
      ), // Center
    ); // Container // GestureDetector
  }
}
```

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

class SquareTile extends StatelessWidget {
  final String imagePath;
  const SquareTile({super.key, required this.imagePath, required this.onTap});
  final Function()? onTap;

  @override
  Widget build(BuildContext context) {
    // TODO: implement build

    return GestureDetector(
      onTap: onTap,
      child: Container(
        padding: EdgeInsets.all(20),
        decoration: BoxDecoration(
          border: Border.all(color: Colors.grey),
          borderRadius: BorderRadius.circular(16)), // BoxDecoration
      child: Image.asset(
        imagePath,
        height: 40,
      ), // Image.asset // Container
    ); // GestureDetector
  }
}
```

I amb açò passem a la primera pantalla que verdaderament mostra una interfície, la pantalla de Log in, aquesta es una pantalla a la que li passarem com a paràmetre la funció togglePages per a gastar-la en un futur. Aquesta pantalla comptarà amb 2 textfields per a introduir les dades del registre, un boto per a fer el log in amb aquestes dades, un boto per a fer el registre amb un compte de google i finalment un boto per a canviar amb la pantalla de registre en cas de no tindre compte. Finalment per a fer els registres amb mail i contrasenya utilitzarem una instancia de la classe FirebaseAuth i la funció signInWithEmailAndPassword :

```
class Loginscreen extends StatefulWidget {
  Loginscreen({super.key, required this.onTap});

  Function()? onTap;

  @override
  State<Loginscreen> createState() => _LoginscreenState();
}

class _LoginscreenState extends State<Loginscreen> {
  User? user;

  final usuariController = TextEditingController();

  final passController = TextEditingController();

  void loginUser() async {
    showDialog(
      context: context,
      builder: (context) {
        return Center(
          child: CircularProgressIndicator(),
        ); // Center
      });
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: usuariController.text, password: passController.text);
      Navigator.pop(context);
    } on FirebaseAuthException catch (e) {
      Navigator.pop(context);

      if (e.code == "user-not-found" || e.code == "wrong-password") {
        showDialog(
          context: context,
          builder: (context) {
            return AlertDialog(
              title: Text("El mail o contraseña son incorrectes"),
            ); // AlertDialog
          });
      }
    }
  }
}
```



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white10,
    body: SafeArea(
      child: Center(
        child: SingleChildScrollView(
          child: Column(mainAxisAlignment: MainAxisAlignment.center, children: [
            SizedBox(height: 50),
            //logo
            Icon(Icons.lock),
            SizedBox(height: 50),
            //welcome
            Text(
              "Benvingut",
              style: TextStyle(color: Colors.black, fontSize: 16),
            ), // Text
            SizedBox(height: 25),

            //username
            MYtextfield(
              controller: usuariController,
              hintText: "Usuari",
              amagarText: false), // MYtextfield
            SizedBox(height: 5),
            //password
            MYtextfield(
              controller: passController,
              hintText: "Contrasenya",
              amagarText: true), // MYtextfield
            //forgot pass
            SizedBox(height: 10),
            Padding(
              padding: EdgeInsets.symmetric(horizontal: 25.0),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.end,
                children: [Text("Ha olvidat la seua contrasenya?")],
              ), // Row // Padding
            ),
            SizedBox(height: 25),
            //sign in
            MyButton(
              onTap: loginUser,
            ), // MyButton
            SizedBox(height: 50),
            //continue with

            Padding(
              padding: EdgeInsets.symmetric(horizontal: 25),
              child: Row(
                children: [
                  Expanded(
                    child: Divider(
                      thickness: 0.5,
                      color: Colors.grey,
                    ), // Divider
```




```
    ], // Expanded
    Padding(
      padding: EdgeInsets.symmetric(horizontal: 10),
      child: Text(
        "Continua amb:",
        style: TextStyle(color: Colors.grey[700]),
      ), // Text
    ), // Padding
    Expanded(
      child: Divider(
        thickness: 0.5,
        color: Colors.grey,
      ), // Divider
    ), // Expanded
  ],
), // Row
), // Padding
//google login
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    SquareTile(
      onTap: () => AuthService().signInWithGoogle(),
      imagePath: 'lib/assets/img/google.png') // SquareTile
  ],
), // Row

//register
SizedBox(height: 50),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text("Encara no està registrat?"),
    SizedBox(width: 4),
    GestureDetector(
      onTap: widget.onTap,
      child: Text(
        "Registra't",
        style: TextStyle(
          color: Colors.blue, fontWeight: FontWeight.bold),
      ), // Text
    ), // GestureDetector
  ],
), // Row
), // Column
)); // SingleChildScrollView // Center // SafeArea // Scaffold
}
```

Per a fer el registre amb google he gastat una classe que he creat jo anomenada AuthService amb la que gestione tota la lògica per a fer login a un usuari amb el compte de google.

```
class AuthService {
  signinWithGoogle() async {
    try {
      final GoogleSignInAccount? gUser = await GoogleSignIn().signIn();

      final GoogleSignInAuthentication gAuth = await gUser!.authentication;

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

      return await FirebaseAuth.instance.signInWithCredential(credential);
    } catch (e) {
      print(e);
    }
  }
}
```

Al clicar en el text de Registra't et portarà al la pantalla de registre, que utilitza la mateixa estructura però amb un textfield mes per a poder confirmar la contrasenya en cas de haver fet una errata al escriure-la al camp contrasenya, a banda també canviarà la funció de log in que passarà a ser una de registre per a crear el usuari:

```
class _RegisterscreenState extends State<Registerscreen> {
  final usuariController = TextEditingController();

  final passController = TextEditingController();
  final passConfiController = TextEditingController();

  void registrarUser() async {
    try {
      if (passConfiController.text == passController.text) {
        await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: usuariController.text, password: passController.text);
      } else {
        showError("Les contrasenyes no coincidixen");
      }
    } on FirebaseAuthException catch (e) {
      Navigator.pop(context);

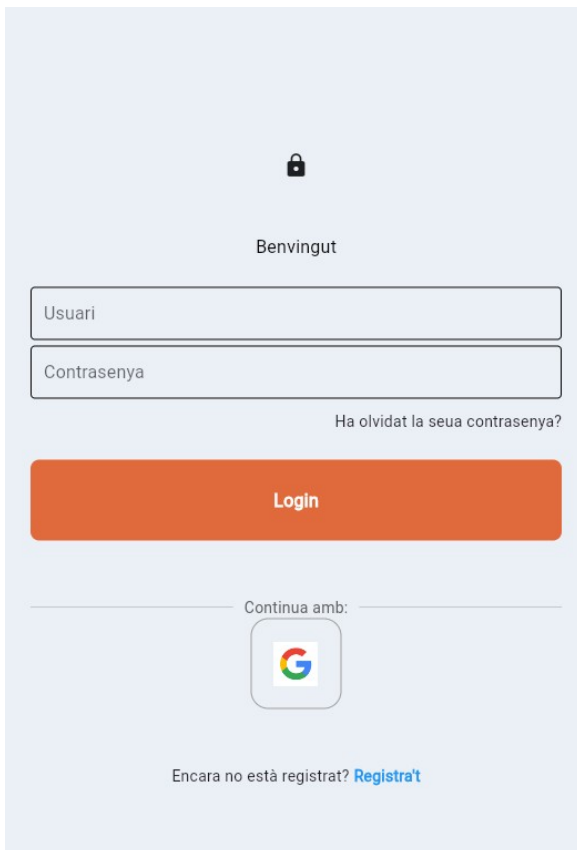
      showError(e.code);
    }
  }

  void showError(String message) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text(message),
        ); // AlertDialog
      });
  }
}
```

```
//confirmar pass
MYtextfield(
  controller: passConfiController,
  hintText: "Confirma la contraseña",
  amagarText: true), // MYtextfield
```

I de la mateixa forma que la pantalla anterior si cliques en el text de log in et portara a la pantalla de Log In.

I així es com es vorien les dos pantalles a l'aplicació:



Mockup of the login screen. At the top center is a lock icon. Below it is the text "Benvingut". There are two input fields: "Usuari" and "Contrasenya". Below the "Contrasenya" field is a link "Ha olvidat la seua contrasenya?". At the bottom is a large orange button labeled "Login". Below the button is a section titled "Continua amb:" with a Google logo icon. At the very bottom is the text "Encara no està registrat? [Registra't](#)".



Mockup of the registration screen. At the top center is a lock icon. Below it is the text "Anem a acrear un comptar". There are three input fields: "Usuari", "Contraseña", and "Confirma la contraseña". Below the "Confirma la contraseña" field is a link "Ha olvidat la seua contrasenya?". At the bottom is a large orange button labeled "Login". Below the button is a section titled "Continua con:" with a Google logo icon. At the very bottom is the text "Ja tens un comptar? [Log in](#)".

Després de entrar a la app es mostrarà la mainScreen, aquesta mostrarà a la part superior un AppBar que donarà la benvinguda al usuari mostrant el mail amb el que estan registrats i un boto per a fer log out de l'aplicació, i a la part inferior un ListView amb els diferents supermercats que hi han disponibles, el seu logo i el seu nom, al clicar en aquests supermercats et portarà a la pantalla de llistes que explicaré en un moment.

```
class mainScreen extends StatelessWidget {
  List<Map<String, String>> llistamercats = [
    {"mercat": "mercadona"},
    {"mercat": "masymas"},
    {"mercat": "carrefour"}
  ];

  mainScreen({super.key});

  final user = FirebaseAuth.instance.currentUser!;
  void logout() {
    FirebaseAuth.instance.signOut();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('benvingut ${user.email}'),
        actions: [IconButton(onPressed: logout, icon: Icon(Icons.logout))],
      ), // AppBar
      body: Center(
        child: _creaLlistaMercats(llistamercats),
      ), // Center
    ); // Scaffold
  }
}
```

```
dynamic _creaLlistaMercats(List<Map<String, String>>? values) {
  if (values == null) {
    // Si la llista és nul·la retornem un indicador de progrés
    return const [CircularProgressIndicator()];
  }

  return ListView.builder(
    itemCount: values.length,
    itemBuilder: (BuildContext context, int index) {
      if (values.isNotEmpty) {
        String? mercat = values[index]["mercat"];

        return ClickableMercatCard(mercat: mercat);
      } else {
        return Center(
          child: Text("No hi ha mercats disponibles"),
        ); // Center
      }
    },
  ); // ListView.builder
}

class ClickableMercatCard extends StatelessWidget {
  ClickableMercatCard({this.mercat, super.key});
  final String? mercat;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: ((context) => llisteScreen(supermercat: mercat))),
        ),
      child: mercatCard(mercat: mercat); // GestureDetector
    );
  }
}
```

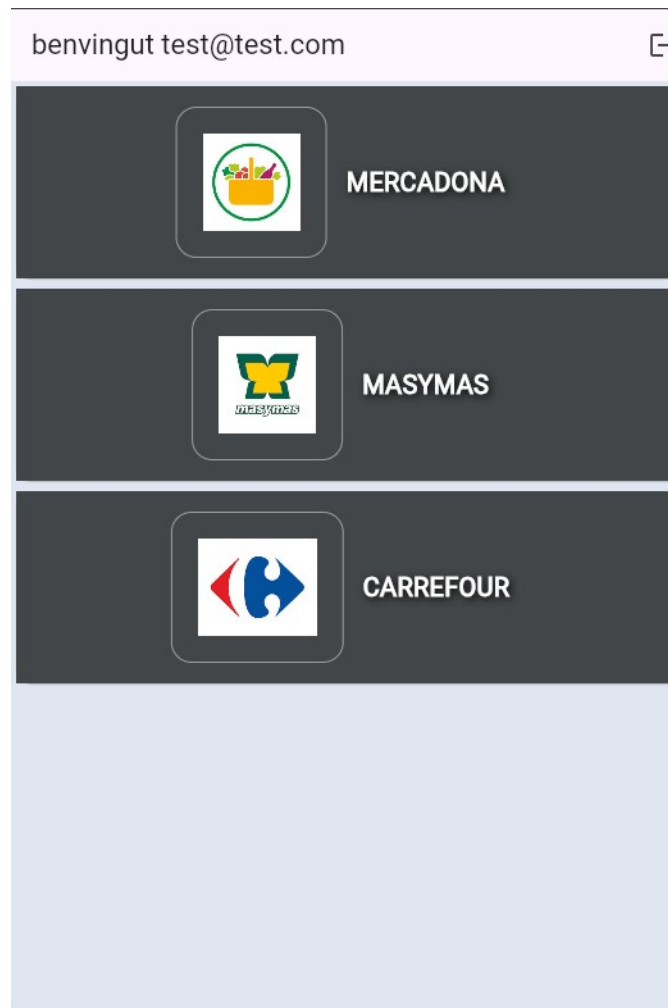
```
class mercatCard extends StatelessWidget {
  mercatCard({super.key, this.mercat});
  final String? mercat;

  @override
  Widget build(BuildContext context) {
    return Card(
      child: Container(
        height: 150,
        padding: EdgeInsets.all(16),
        decoration: BoxDecoration(
          color: const Color.fromARGB(255, 67, 70, 72),
        ), // BoxDecoration
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            padding: EdgeInsets.all(20),
            decoration: BoxDecoration(
              border: Border.all(color: Colors.grey),
              borderRadius: BorderRadius.circular(16)),
            child: Image.asset(
              'lib/assets/img/$mercat.png',
              height: 150,
            ), // Image.asset // Container

            Padding(
              padding: const EdgeInsets.all(15.0),
              child: Text(
                "${mercat?.toUpperCase()}",
                style: TextStyle(
                  fontFamily: "LeckerliOne",
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                  fontSize: 20,
                  shadows: [
                    Shadow(
                      offset: Offset(2, 2),
                      color: Colors.black,
                      blurRadius: 3), // Shadow
                  ], // TextStyle
                ), // Text
              ), // Padding
            ), // Row // Container
          ); // Card
        ]
      );
  }
}
```

Per a mostrar els supermercats he creat una classe anomenada `mercatCard` que conté el logo del supermercat i el nom, i aquesta classe es creada per la classe `ClickableMercatCard` que es la que fa que es pugui clicar, aquesta es la que farà el `Navigator.push` per a passa a la següent pantalla.

El resultat es vorà de la següent manera:



Al clicar en qualsevol dels supermercats passarem a la pantalla de llistes anomenada al projecte com `LlistesScreen` aquesta ja es més complicada que l'anterior, en aquesta pantalla es veuran les llistes que tenen cada supermercat depenent del usuari que estigui connectat, aquestes llistes mostraran: el nom que li ha proporcionat el usuari, la data de creació de la llista i la quantitat de productes que hi ha a la llista, a més a més a la part inferior hi haurà un botó per a poder afegir una llista nova, aquest obrirà un diàleg per a que el usuari introduïu el nom de la llista, els productes que contindrà i la quantitat dels mateixos.

Ací podrem veure com es fa la crida al provider per aconseguir totes les llistes i com aquestes es filtren segons l'usuari i el supermercat que hi han seleccionats actualment

```
class LlistesScreen extends StatelessWidget {
  LlistesScreen({this.supermercat, Key? key}) : super(key: key);

  final String? supermercat;
  final user = FirebaseAuth.instance.currentUser!;
  @override
  Widget build(BuildContext context) {
    var llistaProv = Provider.of<LlistaProvider>(context);
    llistaProv.superMercatActual = supermercat!;
    List<Map<String, dynamic>>? values = llistaProv.llistaEnv;

    // Filtrar las listas que pertenecen al supermercado actual
    List<Llista> listasFiltradas = values
      ?.map<(data) => Llista.fromMap(data)
        .where<(llista) =>
          llista.supermercat == supermercat && llista.usuari == user.email
        .toList() ??
      [];

    return Scaffold(
      appBar: AppBar(
        title: Text('Llistes de Compres de $supermercat'),
      ), // AppBar
      body: Center(
        child: listasFiltradas.isEmpty
          ? Text(
              "No hi han llistes per a aquest Supermercat",
              style: TextStyle(
                fontFamily: "LeckerliOne",
                fontWeight: FontWeight.bold,
                color: Colors.black,
                fontSize: 20,
              ), // TextStyle
            ) // Text
          : _creaLlistaProductes(listasFiltradas),
      ), // Center
      bottomNavigationBar: BottomAppBar(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            IconButton(
              icon: Icon(Icons.arrow_back),
              onPressed: () {
                Navigator.pop(context);
              },
            ), // IconButton
            IconButton(
              icon: Icon(Icons.add),
              onPressed: () {
                _showAddItemDialog(
                  context, llistaProv.superMercatActual, llistaProv);
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

```

    },
    ), // IconButton
  ],
), // Row
), // BottomAppBar
); // Scaffold
}

dynamic _creaLlistaProductes(List<Llista> listasFiltradas) {
  return ListView.builder(
    itemCount: listasFiltradas.length,
    itemBuilder: (BuildContext context, int index) {
      return ClickablLlistaCard(llist: listasFiltradas[index]);
    },
  ); // ListView.builder
}

void _showAddItemDialog(
  BuildContext context, String? superMActual, Llistaprovider llistaProvi) {
  final _formKey = GlobalKey<FormState>();

  Llista llistaNova = Llista.empty();
  String fecha = obtenerFechaDeHoy();
  String? nomLL = "";
  llistaNova.data = fecha;
  llistaNova.supermercat = superMActual!;
  llistaNova.usuari = user.email!;

  List<TextEditingController> _productControllers = [];
  List<TextEditingController> _quantityControllers = [];

  void _addProductField() {
    _productControllers.add(TextEditingController());
    _quantityControllers.add(TextEditingController(text: '0'));
  }

  void _removeProductField(int index) {
    _productControllers.removeAt(index);
    _quantityControllers.removeAt(index);
  }

  _addProductField(); // Añadir un campo inicial

  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Crea una llista nova'),
        content: StatefulBuilder(
          builder: (BuildContext context, StateSetter setState) {
            return Form(
              key: _formKey,
              child: SingleChildScrollView(

```

Ací està tota la lògica per a crear el diàleg de afegir llistes a la base de dades, aquest comprova si hi ha un nom per a la llista i si els productes creats tenen un nom i una cantitat



```
child: Column(  
  mainAxisAlignment: MainAxisAlignment.min,  
  children: [  
    TextFormField(  
      decoration: InputDecoration(labelText: 'Nom'),  
      onSave: (value) {  
        nomLL = value;  
      },  
      validator: (value) {  
        if (value == null || value.isEmpty) {  
          return 'Introdueix un nom';  
        }  
        return null;  
      },  
    ), // TextFormField  
    SizedBox(height: 16),  
    Text('Productes'),  
    ..._productControllers.asMap().entries.map((entry) {  
      int index = entry.key;  
      return Row(  
        children: [  
          Expanded(  
            flex: 2,  
            child: TextFormField(  
              controller: _productControllers[index],  
              decoration: InputDecoration(  
                labelText: 'Producte',  
              ), // InputDecoration  
              validator: (value) {  
                if (value == null || value.isEmpty) {  
                  return 'Introdueix un producte';  
                }  
                return null;  
              },  
            ), // TextFormField  
          ), // Expanded  
          SizedBox(width: 8),  
          Expanded(  
            flex: 1,  
            child: Row(  
              children: [  
                IconButton(  
                  icon: Icon(Icons.remove),  
                  onPressed: () {  
                    setState(() {  
                      int currentValue = int.parse(  
                        _quantityControllers[index].text);  
                      if (currentValue > 0) {  
                        currentValue--;  
                        _quantityControllers[index].text =  
                          currentValue.toString();  
                    });  
                  },  
                ), // IconButton  
              ],  
            ), // Row  
          ),  
        ],  
      ),  
    ),  
  ],  
),
```

```
), // IconButton  
Expanded(  
  child: TextFormField(  
    controller: _quantityControllers[index],  
    decoration: InputDecoration(  
      labelText: 'Quantitat',  
    ), // InputDecoration  
    keyboardType: TextInputType.number,  
    validator: (value) {  
      if (value == null || value.isEmpty) {  
        return 'Introdueix una quantitat';  
      }  
      if (int.tryParse(value) == null) {  
        return 'Introdueix un número vàlid';  
      }  
      return null;  
    },  
  ), // TextFormField  
), // Expanded  
IconButton(  
  icon: Icon(Icons.add),  
  onPressed: () {  
    setState(() {  
      int currentValue = int.parse(  
        _quantityControllers[index].text);  
      currentValue++;  
      _quantityControllers[index].text =  
        currentValue.toString();  
    });  
  },  
), // IconButton  
],  
), // Row  
), // Expanded  
IconButton(  
  icon: Icon(Icons.remove_circle),  
  onPressed: () {  
    setState(() {  
      _removeProductField(index);  
    });  
  },  
), // IconButton  
],  
); // Row  
)).toList(),  
SizedBox(height: 16),  
ElevatedButton(  
  onPressed: () {  
    setState(() {  
      _addProductField();  
    });  
  },  
),
```




```
        _addProductField();
    });
},
    child: Text('Afegeix Producte'),
  ), // ElevatedButton
],
), // Column
), // SingleChildScrollView
); // Form
},
), // StatefulBuilder
actions: [
  TextButton(
    child: Text('Cancel·la'),
    onPressed: () {
      Navigator.of(context).pop();
    },
  ), // TextButton
  TextButton(
    child: Text('Afegeix'),
    onPressed: () {
      if (_formKey.currentState!.validate()) {
        _formKey.currentState!.save();

        List<Map<String, dynamic>> products = [];
        for (int i = 0; i < _productControllers.length; i++) {
          products.add({
            'comprat': false,
            'nom': _productControllers[i].text,
            'cantitat': _quantityControllers[i].text,
          });
        }
        llistaNova.nom = nomLL!;
        llistaNova.items = products;

        llistaProvi.addLlista(llibraNova);

        Navigator.of(context).pop();
      }
    },
  ),
],
);
```

```
class ClickablLlistaCard extends StatelessWidget {  
  ClickablLlistaCard({required this.llist, super.key});  
  final Llista llist;  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: () {  
        Navigator.push(  
          context,  
          MaterialPageRoute(  
            builder: ((context) => itemsScreen(  
              llista: llist,  
            )))); // itemsScreen // MaterialPage  
      },  
      child: llistaCard(llist: llist)); // GestureDetect  
    }  
  }  
}
```

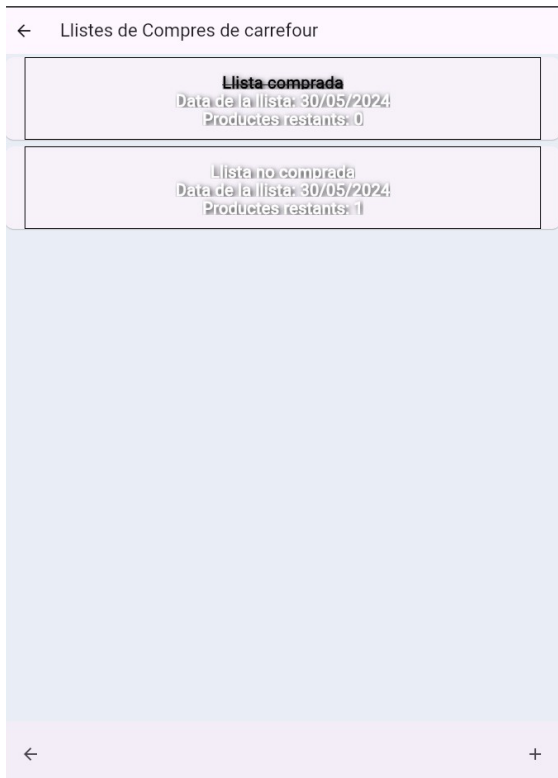
Aquesta es la classe ClickableLListaCard que es la que es crida al hora de crear el listView, aquesta té el mètode onTap que navega a la pròxima pantalla que serà la de items, finalment aquesta classe crearà dins d'ella un widget de la classe llistaCard que veurem a continuació

La classe llistaCard és la que contindrà la informació de la llista, mostrara el nom que canviarà de color segons si estan comprats tots els productes o no, la data de creació i quants productes queden per comprar.

```
class llistaCard extends StatelessWidget {
  llistaCard({super.key, required this.llista});
  final llista llista;

  @override
  Widget build(BuildContext context) {
    llista.productesComprats;
    return Card(
      child: Container(
        height: 100,
        padding: EdgeInsets.all(16),
        margin: EdgeInsets.symmetric(horizontal: 25),
        decoration: BoxDecoration(
          border: Border.all(
            color: Colors.black,
          ), // Border.all
        ), // BoxDecoration
      child: Column(
        children: [
          Expanded(
            flex: 1,
            child: Text(
              llista.nom,
              style: TextStyle(
                decoration:
                  llista.comprada ? TextDecoration.lineThrough : null,
                fontFamily: "LeckerliOne",
                fontWeight: FontWeight.bold,
                color: llista.comprada ? Colors.black : Colors.white,
                fontSize: 20,
                shadows: [
                  Shadow(
                    offset: Offset(2, 2),
                    color: Colors.black,
                    blurRadius: 3,
                  ), // Shadow
                ],
              ), // TextStyle
            ), // Text
          ), // Expanded
          Expanded(
            flex: 1,
            child: Text(
              "Data de la llista: ${llista.data}",
              style: TextStyle(
                fontFamily: "LeckerliOne",
                fontWeight: FontWeight.bold,
                color: Colors.white,
                fontSize: 20,
                shadows: [
                  Shadow(
                    offset: Offset(2, 2),
                    color: Colors.black,
                    blurRadius: 3,
                  ), // Shadow
                ],
              ), // TextStyle
            ), // Text
          ), // Expanded
          Expanded(
            flex: 1,
            child: Text(
              "Productes restants: ${llista.productesComprats()}",
              style: TextStyle(
                fontFamily: "LeckerliOne",
                fontWeight: FontWeight.bold,
                color: Colors.white,
                fontSize: 20,
                shadows: [
                  Shadow(
                    offset: Offset(2, 2),
                    color: Colors.black,
                    blurRadius: 3,
                  ), // Shadow
                ],
              ), // TextStyle
            ), // Text
          ), // Expanded
        ],
      ),
    );
  }
}
```

Finalment a la interfície és veurà de la següent manera:



I per a finalitzar tenim la ultima pantalla: la dels Productes, aquesta té una estructura molt similar a la anterior ja que es el mateix concepte: un ListView que contindrà els items i aquesta canviaran segons si estan comprats o no.

Ací tenim el Scaffold principal que tindrà les AppBar i BottomBar, i al centre contindrà, segons si la llista està plena o buida, un text mostrant que no hi han productes o el list view amb els productes.

```
class itemsScreen extends StatefulWidget {
  itemsScreen({super.key, required this.llista});
  final Llista llista;

  @override
  State<itemsScreen> createState() => _itemsScreenState();
}

class _itemsScreenState extends State<itemsScreen> {
  @override
  Widget build(BuildContext context) {
    List<Item> items = Item.fromList(widget.llista.items);

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.llista.nom),
      ), // AppBar
      body: Center(
        child: items.isEmpty
          ? Text(
              "No hi han productes per a aquesta llista",
              style: TextStyle(
                fontFamily: "LeckerliOne",
                fontWeight: FontWeight.bold,
                color: Colors.black,
                fontSize: 20,
              ), // TextStyle
            ) // Text
          : _creaLlistaItems(items),
      ), // Center
      bottomNavigationBar: BottomAppBar(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            IconButton(
              icon: Icon(Icons.arrow_back),
              onPressed: () {
                Navigator.pop(context);
              },
            ), // IconButton
          ],
        ), // Row
      ), // BottomAppBar
    ); // Scaffold
  }
}
```

Aquesta es la funció que utilitze per a crear el ListView, igual que en les anterior pantalles, crida a una classe Clickable per a gestionar el clic sobre el Item i aquesta classe crida a la classe que contindrà la informació.

```
dynamic _creallistaItems(final List<Item> items) {  
    return ListView.builder(  
        itemCount: items.length,  
        itemBuilder: (BuildContext context, int index) {  
            return ClickableItemCard(  
                item: items[index],  
                llista: widget.llista,  
            ); // ClickableItemCard  
        },  
    ); // ListView.builder  
}  
]  
  
class ClickableItemCard extends StatefulWidget {  
    ClickableItemCard({required this.llista, super.key, required this.item});  
    final Llista llista;  
    final Item item;  
  
    @override  
    State<ClickableItemCard> createState() => _ClickableItemCardState();  
}  
  
class _ClickableItemCardState extends State<ClickableItemCard> {  
    @override  
    Widget build(BuildContext context) {  
        return GestureDetector(  
            onTap: () {setState(() {  
                actualitzarEstatItem(context);  
            })};  
        ),  
        child: ItemCard(  
            llista: widget.llista,  
            item: widget.item,  
        )); // ItemCard // GestureDetector  
    }  
}
```

Al clicar en un producte es cridarà aquesta funció que actualitzarà la base de dades posant al contrari del que estava el camp comprat d'eixe item en concret per a que la app el mostre com toca:

```
actualitzarEstatItem(BuildContext context) {  
  List<dynamic> llistaItemsA = widget.llist.items;  
  List<dynamic> llistaItemsN = [];  
  var llistaProvi = Provider.of<Llistaprovider>(context, listen: false);  
  widget.item.comprat = !widget.item.comprat;  
  
  for (dynamic listItem in llistaItemsA) {  
    if (listItem["nom"] == widget.item.nom) {  
      llistaItemsN.add(widget.item.toMap());  
    } else {  
      llistaItemsN.add(listItem);  
    }  
  }  
}  
  
widget.llist.items = llistaItemsN;  
  
llistaProvi.updateLlista(widget.llist.id, "items", widget.llist);  
}
```

I per a concluir, al igual que en la pantalla anterior, està la classe ItemCard, que es la que mostra la quantitat i el nom del producte, si el camp comprat està en false aquest apareixerà normal però si està en true el text estarà taxat i en el fondo en gris per a denotar que aquest està comprat d'una forma visual i fàcil:

```
class ItemCard extends StatelessWidget {
  const ItemCard({
    super.key,
    required this.item,
    required this.llista,
  });
  final Item item;
  final Llista llista;

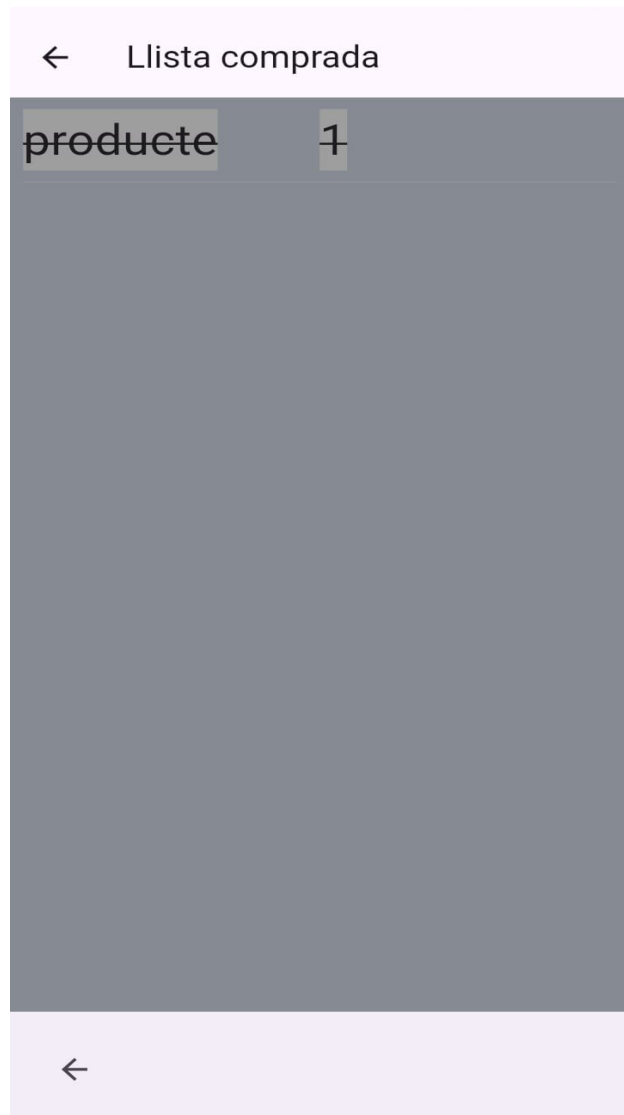
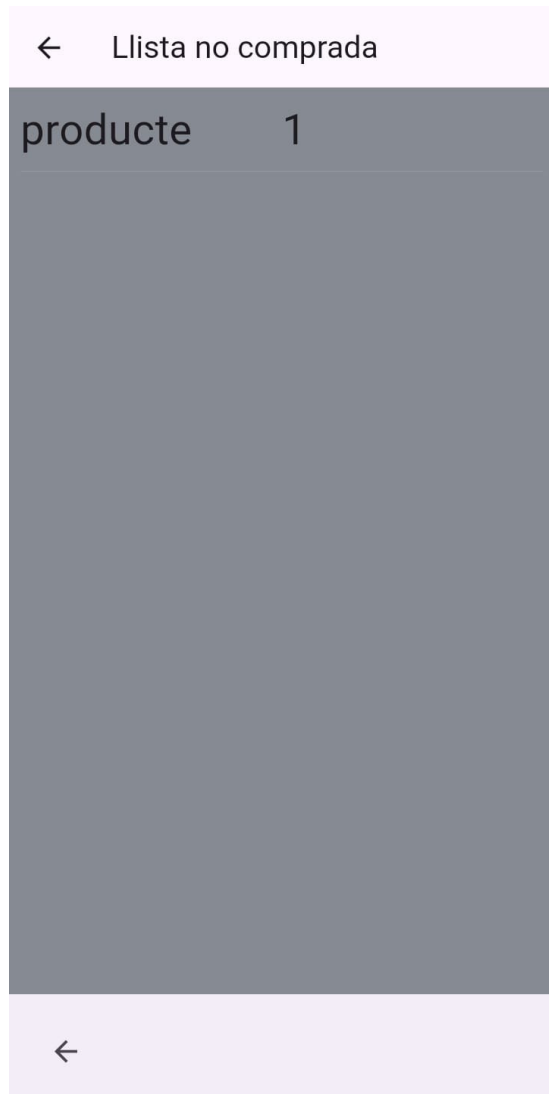
  @override
  Widget build(BuildContext context) {
    print(item);
    return Center(
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(8.0),

          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,

                children: [
                  Expanded(
                    child: Text(
                      item.nom,

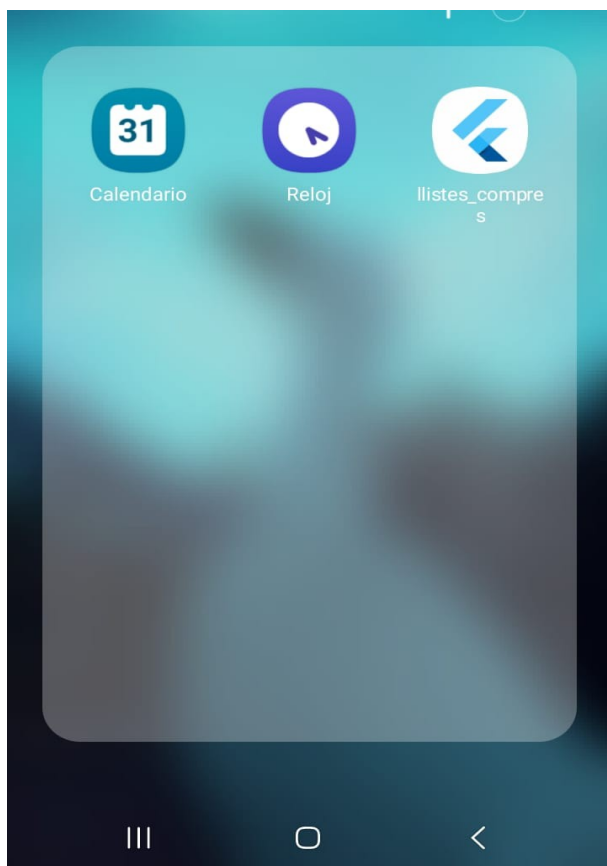
                      style: TextStyle(
                        decoration:
                          item.comprat ? TextDecoration.lineThrough : null,
                        fontSize: 30,
                        backgroundColor: item.comprat ? Colors.grey : null,
                      ), // TextStyle
                    ), // Text
                  ), // Expanded
                  Expanded(
                    child: Text(item.cantitat ,
                      style: TextStyle(
                        decoration:
                          item.comprat ? TextDecoration.lineThrough : null,
                        fontSize: 30,
                        backgroundColor: item.comprat ? Colors.grey : null,
                      ), // TextStyle
                    ), // Text
                  ), // Expanded
                ],
              ), // Row
              Divider(
                thickness: 0.5,
                color: Colors.grey,
              ), // Divider
            ],
          ), // Column
        ), // Padding
      ), // SingleChildScrollView
    ); // Center
  }
}
```


El resultat es el següent:



4 Implantació de la solució

Per a la instal·lació de la aplicació a he decidit fer una APK del projecte i instal·lar-la al meu dispositiu mòbil, i encara que sorgixen alguns problemes amb la interfície, l'aplicació funciona perfectament i es completament utilitzable. Per a instal·lar la apk es igual de fàcil que amb qualsevol altra fer clic i ella mateixa s'instal·larà



5 Conclusions i treballs futurs

Per a confluïr he de dir que en general el desenvolupament de l'aplicació en si no ha sigut molt complicat, les complicacions han aparegut al hora de implementar ferramentes externes com la base de dades o el log in en Google, però tot açò ha fet que ho tingui mes clar en un futur en cas de tornar a necessitar-ho.

Dificultats

- Canviar la forma de guardar les dades
- La habilitació del registre amb Google.
- La actualització de les dades de forma dinàmica a la interfície

Futurs Treballs

- Habilitar la creació de nous supermercats per part del usuari
- Millorar la interfície
- Habilitar un pont entre l'aplicació i les plataformes de compres online dels supermercats
- Crear llistes multiusuari

Bibliografia

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

<https://chat.openai.com/>

<https://www.youtube.com/@CodigoCorrecto>

<https://youtu.be/4fucdtPwTWI?si=4gJttjcdSXG9lsrT>

https://pub.dev/packages/google_sign_in_web

<https://stackoverflow.com/>