Proyecto Final - Aplicación de recetas

Descripción general del proyecto

Este proyecto es una aplicación móvil que permite a los usuarios gestionar, descubrir y crear recetas de cocina de manera fácil y organizada. Diseñada para amantes de la cocina de cualquier nivel, la app ofrece funciones como generación de listas de compras, favoritos y categorización personalizada de recetas. Los usuarios pueden guardar recetas con ingredientes, pasos detallados, fotos y datos como tiempo de preparación y dificultad. Toda la información se encuentra alojada en la nube en la base de datos de Firebase.

Explicación de funciones implementadas

1. Controladores
    a. Categorias

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import '../model/category.dart';


class CategoryController {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<List<Category>> fetchCategory() async {
    List<Category> categorias = [];

    try {
      QuerySnapshot snapshot = await _firestore.collection('categories').get();

      for(var doc in snapshot.docs)
      {
        Category category = Category.fromJson(doc.data() as Map<String, dynamic>);
        categorias.add(category);
      }
    }catch(e)
    {
      print('Error al leer las categorias: $e');
    }

    return categorias;
  }
}
```

En este apartado se obtienen todas las categorías de la base de datos de Firebase. Se puede observar que la clase CategoryController tiene una función fetchCategory que devuelve una lista de categorías. Si ocurre algún error se devuelve una lista vacía.

b. Favoritos

```dart
import 'package:shared_preferences/shared_preferences.dart';

class FavoritesController {
  static const _keyFavorites = 'favorites';

  static Future<List<String>> getFavorites() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getStringList(_keyFavorites) ?? [];
  }

  static Future<void> addFavorite(String recipeID) async {
    final prefs = await SharedPreferences.getInstance();
    final favorites = await getFavorites();
    if(!favorites.contains(recipeID)) {
      favorites.add(recipeID);
      await prefs.setStringList(_keyFavorites, favorites);
    }
  }

  static Future<void> deleteRecipe(String recipeID) async {
    final prefs = await SharedPreferences.getInstance();
    final favorites = await getFavorites();
    favorites.remove(recipeID);
    prefs.setStringList(_keyFavorites, favorites);
  }

  static Future<bool> isfavorite(String recipeID) async{
    final favorites = await getFavorites();
    return favorites.contains(recipeID);
  }
}
```

Para el controlador de favoritos se puede observar que se han utilizado SharedPreferences. Mediante este método se han declarado una serie de métodos para guardar el id de la receta en una lista de cadenas. Para las diferentes funcionalidades implementadas se pueden observar las funciones de obtener los favoritos, añadir uno nuevo, eliminarlo y comprobar si es favorito.

c. Ingredientes

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:recipe_app/model/ingredients.dart';
import 'package:recipe_app/utils/stringExtensions.dart';

class IngredientController {

  Future<List<Ingredient>> fetchIngredientsBySearch(String query) async {
    var snapshot = await FirebaseFirestore.instance
        .collection('ingredients')
        .where('name', isGreaterThanOrEqualTo: query.capitalize())
        .where('name', isLessThanOrEqualTo: query.capitalize() + '\uf8ff')
        .get();

    return snapshot.docs.map((doc) {
      return Ingredient.fromJson(doc.data());
    }).toList();
  }
```

```dart
Future<void> updateIngredientsStock(Ingredient ingredient, int value) async {
  try {
    final querySnapshot = await FirebaseFirestore.instance
        .collection('ingredients')
        .where('id', isEqualTo: ingredient.id)
        .get();

    if (querySnapshot.docs.isNotEmpty) {
      final doc = querySnapshot.docs.first;

      final currentStock = doc.data()['stock'] ?? 0;
      await FirebaseFirestore.instance
          .collection('ingredients')
          .doc(doc.id)
          .update({'stock': currentStock + value});

      final recipesSnapshot = await FirebaseFirestore.instance
          .collection('recipes')
          .get();

      for (var recipeDoc in recipesSnapshot.docs) {
        final ingredientsList = List<Map<String, dynamic>>.from(recipeDoc['ingredients']);

        final hasIngredient = ingredientsList.any((ing) => ing['id'] == ingredient.id);

        if (hasIngredient) {
          final updatedIngredientsList = ingredientsList.map((ing) {
            if (ing['id'] == ingredient.id) {
              ing['stock'] = currentStock + value;
            }
            return ing;
          }).toList();

          await FirebaseFirestore.instance
              .collection('recipes')
              .doc(recipeDoc.id)
              .update({'ingredients': updatedIngredientsList});
        }
      }
    }
  } catch (e) {
    print('Error al modificar stock de ingrediente: $e');
  }
}
}
```

En este caso hay dos funciones, una para mostrar los ingredientes por búsqueda y otra para modificar los ingredientes en la base de datos. En el caso de la modificación como los datos de los ingredientes se encuentran tanto en la colección de ingredientes como de recetas habrá que hacer una modificación doble para mantener los datos sincronizados y coherentes.

La función de modificación su único propósito es el de modificar los datos de stock del ingrediente, sin modificar las demás propiedades del mismo.

d. Recetas

```dart
class RecipeController {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;


  String generateID() {
    var rand = Random();
    return 'recipe_${DateTime.now().millisecondsSinceEpoch}_${rand.nextInt(1000)}';
  }

  Future<List<Recipe>> fetchRecipes() async {
    List<Recipe> recipes = [];

    try {
      QuerySnapshot snapshot = await _firestore.collection('recipes').get();

      for(var doc in snapshot.docs)
      {
        Recipe recipe = Recipe.fromJson(doc.data() as Map<String, dynamic>);
        recipes.add(recipe);
      }
    }catch(e)
    {
      print('Error al leer las recetas: $e');
    }

    return recipes;
  }
```

```dart
Future<bool> addRecipe(Recipe recipe) async {
  try {
    String id = recipe.id.isEmpty ? generateID() : recipe.id;
    await _firestore.collection('recipes').doc(id).set(recipe.toJson());
    return true;
  }catch(e)
  {
    return false;
  }
}

Future<void> deleteRecipe(String id) async {
  try {
    await _firestore.collection('recipes').doc(id).delete();
  }catch(e)
  {
    print('Error al eliminar receta: $e');
  }
}

Future<bool> updateRecipe(Recipe recipe) async {
  try {
    await _firestore.collection('recipes').doc(recipe.id).update(recipe.toJson());
    return true;
  }catch(e) {
    return false;
  }
}
```

En el caso del controlador de receta se aplica un CRUD completo. Desde la muestra de recetas hasta la creación, modificación y eliminación de las mismas, todo gestionado a través de Firebase.

2. Modelos
    a. Categorias

```dart
class Category {
  final int _id;
  final String _name;
  final String _description;

  Category({
    required int id,
    required String name,
    required String description,
  })  : _name = name,
        _description = description,
        _id = id;

  String get name => _name;
  String get description => _description;
  int get id => _id;


  //Get information from Firebase
  factory Category.fromJson(Map<String, dynamic> data)
  {
    return Category(
        id: data['id'],
        name: data['name'] ?? '',
        description: data['description'] ?? '');
  }

  //Save information to Firebase
  Map<String, dynamic> toJson()
  {
    return {
      'id' : _id,
      'name' : _name,
      'description' : _description,
    };
  }

  @override
  bool operator ==(Object other) {
    if (identical(this, other)) return true;
    if (other is! Category) return false;
    return other.id == id && other.name == name;
  }

  @override
  int get hashCode => id.hashCode ^ name.hashCode;
}
```

El modelo de Categoria con sus variables, constructor y getters. Aparte de eso tenemos dos

funciones extra para la transformación de datos de forma que firebase puede leerlo y guardarlo correctamente. Además se han sobreescrito dos funcionalidades básicas de clase para la comparación correcta entre clases (el operator == y el hashcode).

b. Ingredientes

```dart
class Ingredient {
  final int _id;
  final String _name;
  final String _description;
  final int _stock;

  Ingredient({
    required int id,
    required String name,
    required String description,
    required int stock
  })  : _name = name,
        _description = description,
        _id = id,
        _stock = stock;

  String get name => _name;
  String get description => _description;
  int get id => _id;
  int get stock => _stock;

  @override
  bool operator ==(Object other) {
    if (identical(this, other)) return true;
    if (other is! Ingredient) return false;
    return other.id == id;
  }

  @override
  // TODO: implement hashCode
  int get hashCode => id.hashCode;


  //Get information from Firebase
  factory Ingredient.fromJson(Map<String, dynamic> data)
  {
    return Ingredient(
        id: data['id'],
        name: data['name'] ?? '',
        description: data['description'] ?? '',
        stock: data['stock']
    );
  }
```

Se ha seguido el mismo procedimiento que para la clase de Categorías.

c. Recetas

```dart
class Recipe {
  final String _id;
  final String _title;
  final String _description;
  final List<Ingredient> _ingredients;
  final Map<Ingredient, int> _ingredientQuantities;
  final List<String> _preparation_steps;
  final int _time;
  final Difficulty _difficulty;
  final String _image;
  final Category _category;


  Recipe({
    required String id,
    required String title,
    required String description,
    required List<Ingredient> ingredients,
    required Map<Ingredient, int> ingredientQuantities,
    required List<String> preparation_steps,
    required int time,
    required Difficulty difficulty,
    required String image,
    required Category category
  })  : _id = id,
        _title= title,
        _description = description,
        _ingredients = ingredients,
        _ingredientQuantities = ingredientQuantities,
        _preparation_steps = preparation_steps,
        _time = time,
        _difficulty = difficulty,
        _image = image,
        _category = category;

  String get id => _id;
  String get title => _title;
  String get description => _description;
  List<Ingredient> get ingredients => _ingredients;
  Map<Ingredient, int> get ingredientQuantities => _ingredientQuantities;
  List<String> get preparation_steps => _preparation_steps;
  int get time => _time;
  Difficulty get difficulty => _difficulty;
  String get image => _image;
  Category get category => _category;
```

```dart
//Get information from Firebase
factory Recipe.fromJson(Map<String, dynamic> data)
{
  var ingredientQuantities = (data['ingredient_quantities'] as Map<String, dynamic>?)
      ?.map((key, value) => MapEntry(
      Ingredient.fromJson(Map<String, dynamic>.from(data['ingredients'].firstWhere((ingredient) => ingredient['id'] == int.parse(key)))),
      value as int)) ?? {};  // MapEntry

  return Recipe(
      id: data['id'],
      title: data['title'] ?? '',
      description: data['description'] ?? '',
      ingredients: (data['ingredients'] as List<dynamic>?)
          ?.where((item) => item is Map<String, dynamic>)
          .map((item) => Ingredient.fromJson(item as Map<String, dynamic>))
          .toList() ??
          [],
      ingredientQuantities: ingredientQuantities,
      preparation_steps: List<String>.from(data['preparation_steps'] ?? []),
      time: data['time'] ?? 0,
      difficulty: Difficulty.values.firstWhere((e) => e.toString() == 'Difficulty.${data['difficulty']}',
          orElse: () => Difficulty.low),
      image: data['image'] ?? '',
      category: Category.fromJson(Map<String, dynamic>.from(data['category'] ?? {}))
  );  // Recipe
}

//Save information to Firebase
Map<String, dynamic> toJson()
{
  return {
    'id' : _id,
    'title' : _title,
    'description' : _description,
    'ingredients' : _ingredients.map((ingredient) => ingredient.toJson()).toList(),
    'ingredient_quantities': _ingredientQuantities.map((key, value) => MapEntry(key.id.toString(), value)),
    'preparation_steps' : _preparation_steps,
    'time' : _time,
    'difficulty' : _difficulty.name,
    'image' : _image,
    'category' : _category.toJson()
  };
}
```

```dart
  // Método para crear un objeto Product desde una cadena JSON
  factory Recipe.fromJsonString(String jsonString) {
    return Recipe.fromJson(json.decode(jsonString));
  }

  // Método para convertir el objeto Product en una cadena JSON
  String toJsonString() {
    return json.encode(toJson());
  }
}


extension DifficultyExtension on Difficulty {
  String get displayName {
    switch (this) {
      case Difficulty.high:
        return 'High';
      case Difficulty.medium:
        return 'Medium';
      case Difficulty.low:
        return 'Low';
      default:
        return '';
    }
  }
}
```

La receta es la clase creada más completa. Desde la creación de variables, getters y constructores, también se han añadido los convertidores para leer los datos en Firebase y para terminar se ha añadido una pequeña funcionalidad para transformar la clase de manera legible para que el Navigator 2.0 puede leer los argumentos pasados como un JSON STRING.

3. Navegación

Para el sistema de navegación se ha utilizado el Navigator 2.0 creando las clase de RouteInformationParse y AppRouterDelegate.

a. RouteInformationParser

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

import '../model/recipe.dart';

class AppRouteInformationParser extends RouteInformationParser<RouteSettings> {
  @override
  Future<RouteSettings> parseRouteInformation(RouteInformation routeInformation) async {

    final uri = routeInformation.uri;
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'login') return RouteSettings(name: '/login');
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'register') return RouteSettings(name: '/register');
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'recipes') return RouteSettings(name: '/recipes');
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'newrecipe') return RouteSettings(name: '/newrecipe');
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'favorites') return RouteSettings(name: '/favorites');
    if(uri.pathSegments.length == 2 && uri.pathSegments[0] == 'detailrecipe') {
      final recipeJsonString = uri.pathSegments[1];
      final recipe = Recipe.fromJsonString(recipeJsonString);
      return RouteSettings(name: '/recipedetail', arguments: recipe);
    }
    if(uri.pathSegments.length == 2 && uri.pathSegments[0] == 'updaterecipe') {
      final recipeJsonString = uri.pathSegments[1];
      final recipe = Recipe.fromJsonString(recipeJsonString);
      return RouteSettings(name: '/updaterecipe', arguments: recipe);
    }
    if(uri.pathSegments.length == 1 && uri.pathSegments[0] == 'shopping') return RouteSettings(name: '/shopping');
    /*if(uri.pathSegments.length == 2 && uri.pathSegments[0] == 'productDetail'){
      final productJsonString = uri.pathSegments[1];
      final product = Product.fromJsonString(productJsonString);
      return RouteSettings(name: '/productDetail', arguments: product);
    }*/
    return RouteSettings(name: '/');
  }
```

```dart
  @override
  RouteInformation? restoreRouteInformation(RouteSettings configuration)
  {
    if (configuration.name == '/recipedetail') {
      final recipe = configuration.arguments as Recipe;
      final recipeJsonString = recipe.toJsonString();
      return RouteInformation(uri: Uri.parse('/productDetail/$recipeJsonString'));
    }

    if (configuration.name == '/updaterecipe') {
      final recipe = configuration.arguments as Recipe;
      final recipeJsonString = recipe.toJsonString();
      return RouteInformation(uri: Uri.parse('/updaterecipe/$recipeJsonString'));
    }
    return RouteInformation(uri: Uri.parse(configuration.name ?? '/'));
  }
}
```

En el routeInformationParser se han creado todas las rutas. Para las rutas que necesiten un argumento en específico se le ha pasado dicho argumento.

b. AppRouterDelegate

```dart
class AppRouterDelegate extends RouterDelegate<RouteSettings>
    with ChangeNotifier, PopNavigatorRouterDelegateMixin<RouteSettings> {
  final GlobalKey<NavigatorState> navigatorKey;
  RouteSettings? _currentRoute = RouteSettings(name: '/');

  AppRouterDelegate() : navigatorKey = GlobalKey<NavigatorState>();

  RouteSettings? get currentConfiguration => _currentRoute;

  void _setNewRoutePath(RouteSettings settings) {
    _currentRoute = settings;
    notifyListeners();
  }

  @override
  Future<void> setNewRoutePath(RouteSettings configuration) async {
    _setNewRoutePath(configuration);
  }

  @override
  Widget build(BuildContext context) {
    return Navigator(
      key: navigatorKey,
      pages: [
       // if(_currentRoute?.name == '/')
          CustomTransitionPage(key: ValueKey('IntroScreen'),
              child: IntroScreen()),  // CustomTransitionPage
        if(_currentRoute?.name == '/login')
          CustomTransitionPage(key: ValueKey('LoginScreen'),child: LoginScreen()),
        if(_currentRoute?.name == '/register')
          CustomTransitionPage(key: ValueKey('RegisterScreen'),child: Registerscreen()),
        if(_currentRoute?.name == '/recipes')
          CustomTransitionPage(key: ValueKey('RecipesScreen'), child: RecipesScreen()),
        if(_currentRoute?.name == '/newrecipe')
          CustomTransitionPage(key: ValueKey('NewRecipeScreen'), child: NewRecipeScreen()),
        if(_currentRoute?.name == '/recipedetail')
          CustomTransitionPage(key: ValueKey('DetailScreen'), child: DetailScreen(_currentRoute?.arguments as Recipe)),
        if(_currentRoute?.name == '/updaterecipe')
          CustomTransitionPage(key: ValueKey('UpdateRecipeScreen'), child: UpdateRecipeScreen(_currentRoute?.arguments as Recipe)),
        if(_currentRoute?.name == '/shopping')
          CustomTransitionPage(key: ValueKey('ShoppingScreen'), child: ShoppingScreen()),
        if(_currentRoute?.name == '/favorites')
          CustomTransitionPage(key: ValueKey('FavoritesScreen'), child: FavoritesScreen()),
      ],
      onPopPage: (route, result) {
        if (!route.didPop(result)) {
          return false;
        }

        if(_currentRoute?.name == '/register' || _currentRoute?.name == '/recipes')
        {
          _setNewRoutePath(RouteSettings(name: '/login'));
        }
        else if(_currentRoute?.name == '/login') {
           exit(0);
        }
        else if(_currentRoute?.name == '/newrecipe' || _currentRoute?.name == '/recipedetail' || _currentRoute?.name == '/shopping' ||
            _currentRoute?.name == '/favorites')
        {
          _setNewRoutePath(RouteSettings(name: '/recipes'));
        }
        else if(_currentRoute?.name == '/updaterecipe') {
          _setNewRoutePath(RouteSettings(name: '/recipedetail', arguments: _currentRoute?.arguments as Recipe));
        }

        return true;
      },
    );  // Navigator
  }
}
```

```dart
class CustomTransitionPage extends Page {
  final Widget child;

  const CustomTransitionPage({required LocalKey key,required  this.child})
  : super(key: key);
  @override
  Route createRoute(BuildContext context) {
    return PageRouteBuilder(
        settings: this,
        pageBuilder: (context, animation, secondaryAnimation) => child,
        transitionsBuilder: (context, animation, secondaryAnimation, child) {
          return FadeTransition(opacity: animation, child: child,);
        }
    );  // PageRouteBuilder
  }
}
```

Este apartado es para redirigir al usuario a la página pertinente según la ruta en la que se encuentre. En algunos casos se le pasa argumentos a la ruta. Aparte de eso se ha diseñado una transición personalizada para cada cambio de ruta.

4. Pantallas

Aquí se han definido todas las pantallas utilizadas a lo largo del proyecto. Al tratarse de crear la interfaz se ha utilizado mucho código, por lo tanto, se explicara como se ha hecho sin mostrar el código como tal. Se mostrarán funciones específicas utilizadas en cada página.

a. Detalle

En la página de detalle se muestra la interfaz de detalle de una receta, por lo tanto a esta página se le pasa por argumento una receta. Aparte de eso se han utilizado las siguientes funciones para la gestión de favoritos.

```dart
Future<void> _verifyFavorite() async {
  final isFavorite = await FavoritesController.isfavorite(widget.recipe.id);
  setState(() {
    _isFavorite = isFavorite;
  });
}

Future<void> _switchFavorite() async {
  if(_isFavorite) {
    await FavoritesController.deleteRecipe(widget.recipe.id);
  } else {
    await FavoritesController.addFavorite(widget.recipe.id);
  }
  _verifyFavorite();
}
```

De esta forma mediante el uso del controlador de favoritos se han guardado las recetas

favoritas de los usuarios. Aparte de eso en la interfaz de detalle se ha mostrado todos las propiedades de la receta excepto su ID.

b. Favoritos

En esta pantalla se ha enseñado la lista de favoritos. Se han implementado las siguientes funciones para guardar las recetas favoritas en una lista.

```
Future<void> _getRecipes() async {
  final recipes = await recipeController.fetchRecipes();
  final favoriteIds = await FavoritesController.getFavorites();

  final favoriteRecipes = recipes.where((recipe) => favoriteIds.contains(recipe.id)).toList();

  setState(() {
    _recipesList = favoriteRecipes;
  });
}

void _deleteRecipeById(String id) {
  FavoritesController.deleteRecipe(id);
  setState(() {
    _recipesList.removeWhere((recipe) => recipe.id == id);
  });
}
```

c. Intro

La página de Introducción es una Splash Screen con la cual pasados 3 segundos se pasa a la página de Login.

```
@override
void initState() {
  super.initState();

  _timer = Timer(Duration(seconds: 3), () {
    final routerDelegate = Router.of(context).routerDelegate as AppRouterDelegate;
    routerDelegate.setNewRoutePath(RouteSettings(name: '/login'));
  });  // Timer
}
```

```
@override
void dispose() {
  _timer?.cancel();
  super.dispose();

}
```

d. Login

La página de Login es la pantalla para autentificarse y tener acceso total a todas las funcionalidades de la aplicación. Se tiene la opción de iniciar sesión con Google o por metodos tradicionales (email y contraseña).

```
onPressed: () async {
  final email = emailController.text.trim();
  final password = passwordController.text.trim();
  final user = await _authService.loginWithEmail(email, password);
  if(user == null)
  {
    showDialog(context: context, builder: (BuildContext builder) {
      return AlertDialog(
        title: Text('Login Failed'),
        content: Text('Invalid username or password. Please try again.'),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK'))  // TextButton
        ],
      );  // AlertDialog
    });
  }
  else {
    final routerDelegate = Router.of(context).routerDelegate as AppRouterDelegate;
    routerDelegate.setNewRoutePath(RouteSettings(name: '/recipes'));
  }
},
```

```
onPressed: () async {
  final user = await _authService.loginWithGoogle();
  if(user == null)
  {
    showDialog(context: context, builder: (BuildContext builder) {
      return AlertDialog(
        title: Text('Login Failed'),
        content: Text('Invalid login with Google. Please try again.'),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK'))  // TextButton
        ],
      );  // AlertDialog
    });
  }
  else
  {
    final routerDelegate = Router.of(context).routerDelegate as AppRouterDelegate;
    routerDelegate.setNewRoutePath(RouteSettings(name: '/recipes'));
  }
},
```

e. Receta Nueva

Esta página es de las páginas más desarrolladas y que más trabajo me ha llevado a nivel de código. Se trata de añadir una receta nueva validando de forma correcta. Algunas de las funciones más destacadas son las siguientes:

```
Future<List<Ingredient>> _getSuggestions(String query) async{
  return await ingredientController.fetchIngredientsBySearch(query);
}

Future<void> _getCategories() async {
  final fetchedCategories = await categoryController.fetchCategory();
  setState(() {
    _categoriesList = fetchedCategories;
  });
}
```

La obtención de ingredientes por búsqueda y de categorías.

```
void _addIngredients(Ingredient ingredient){
  if(!_selectedIngredients.contains(ingredient))
  {
    TextEditingController quantityController = TextEditingController();
    showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: Text('Set quantity for ${ingredient.name}'),
            content: TextField(
              controller: quantityController,
              keyboardType: TextInputType.number,
              decoration: InputDecoration(
                labelText: 'Quantity',
                prefixIcon: Icon(Icons.numbers),
                border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(8)
                ),  // OutlineInputBorder
              )  // InputDecoration
            ),  // TextField
            actions: [
              TextButton(
                  onPressed: () {
                    int quantity = int.tryParse(quantityController.text) ?? 1;
                    setState(() {
                      _selectedIngredients.add(ingredient);
                      _quantitiesIngredients[ingredient] = quantity;
                    });
                    Navigator.pop(context);
                  },
                  child: Text('Add'))  // TextButton
            ],
          );  // AlertDialog
        });
  }
```

```
else {
  showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
        title: Text('Error adding ingredient'),
        content: Text('The ingredient ${ingredient.name} is already in the list'),
        actions: [
          TextButton(
              onPressed: () {
                Navigator.pop(context);
              },
              child: Text('OK'))  // TextButton
        ],
        );  // AlertDialog
      });
  }
}
```

Función para añadir una lista de ingredientes junto a su cantidad.

```
void _addPreparationSteps()
{
  setState(() {
    _preparationSteps.add('');
  });
}

void _removePreparationSteps(int index)
{
  setState(() {
    _preparationSteps.removeAt(index);
  });
}

void _updatePreparationStep(int index, String step){
  setState(() {
    _preparationSteps[index] = step;
  });
}
```

Funciones para añadir, eliminar o modificar pasos de preparación.

```dart
Future<void> _pickImage() async {
  final XFile? pickedImage = await _imagePicker.pickImage(source: ImageSource.gallery);

  if(pickedImage != null) {
    setState(() {
      selectedImage = File(pickedImage.path);
    });
  }

}
```

Función para escoger imagen de tu galería

```dart
void showErrorMessage(String message, BuildContext context)
{
  showDialog(context: context, builder: (BuildContext builder) {
    return AlertDialog(
      title: Text('New Recipe entry error'),
      content: Text(message),
      actions: [
        TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK'))  // TextButton
      ],
    );  // AlertDialog
  });
}
```

Mensajes de error para mostrar problemas al añadir una receta nueva.

f. Recetas

Parecida a la lista de favoritos, se muestran todas las recetas en una lista. Las funciones mas importantes son las siguientes:

```dart
Future<void> _getRecipes() async {
  final recipes = await recipeController.fetchRecipes();
  setState(() {
    _recipesList = recipes;
  });
}

void _deleteRecipeById(String id) {
  FavoritesController.deleteRecipe(id);
  setState(() {
    _recipesList.removeWhere((recipe) => recipe.id == id);
  });
}
```

g. Registro

Página de registro. Se introduce email y contraseña y se guarda en la base de datos.

```dart
void showErrorMessage(String message, BuildContext context)
{
  showDialog(context: context, builder: (BuildContext builder) {
    return AlertDialog(
      title: Text('Registration Failed'),
      content: Text(message),
      actions: [
        TextButton(
            onPressed: () {
              Navigator.pop(context);
            },
            child: Text('OK'))  // TextButton
      ],
    );  // AlertDialog
  });
}

bool isEmailValid(String email) {
  String emailPattern = r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+";
  RegExp regExp = new RegExp(emailPattern);
  return regExp.hasMatch(email);
}
```

Se comprueba que el email tenga el formato de email y también en otro apartado se comprueba que la contraseña tenga mínimo 6 caracteres. Aparte de eso se ha creado una vez mas el mensaje de error para dar información al usuario.

h. Compra

Página para la compra de ingredientes seleccionando una o más recetas. De las funciones más importantes tenemos las siguientes:

```
Future<void> _getRecipes() async {
  final recipes = await recipeController.fetchRecipes();
  setState(() {
    _recipesList = recipes;
  });
}

void _handleRecipeSelection(Recipe recipe, bool? isSelected) {
  setState(() {
    if (isSelected == true) {
      _selectedRecipes.add(recipe);
    } else {
      _selectedRecipes.remove(recipe);
    }
  });
}
```

Aquí se recogen todas las recetas y se comprueba si las casillas de checkbox de cada recetas están seleccionadas o no.

```dart
Future<void> _updateIngredientsForSelectedRecipes(
    List<Recipe> selectedRecipes, int value) async {
  try {
    final Set<int> uniqueIngredientsID = {};
    final Map<int, Ingredient> uniqueIngredientsMap = {};

    for (Recipe recipe in selectedRecipes) {
      for (Ingredient ingredient in recipe.ingredients) {
        if (!uniqueIngredientsID.contains(ingredient.id)) {
          uniqueIngredientsID.add(ingredient.id);
          uniqueIngredientsMap[ingredient.id] = ingredient;
        }
      }
    }

    for (int ingredientID in uniqueIngredientsID) {
      final ingredient = uniqueIngredientsMap[ingredientID];

      if (ingredient != null) {
        await ingredientController.updateIngredientsStock(ingredient, value);
      }

      _getRecipes();
    }
  } catch (e) {
    print('Error al actualizar los stocks de ingredientes: $e');
  }
}
```

En este apartado se añade stock a los ingredientes de las recetas seleccionadas.

i. Modificar Receta

Mismo proceso que para añadir recetas cambiando pocas cosas.

5. Utilidad

a. Servicio de Autentificación

Servicio de Autentificación para la pagina de Login y Registro. Se han implementado las siguiente funciones:

```dart
Future<User?> registerWithEmail(String email, String password) async {
  try {
    UserCredential result = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password);
    return result.user;
  }catch(e) {
    print('Error al registrarse: $e');
    return null;
  }
}

Future<User?> loginWithEmail(String email, String password) async {
  try {
    UserCredential result = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password);
    return result.user;
  }catch(e)
  {
    print('Error al iniciar sesion con usuario y contraseña: $e');
    return null;
  }
}
```

Registro con email y contraseña y asimismo Login con email y contraseña.

```dart
Future<User?> loginWithGoogle() async {
  try {
    final GoogleSignInAccount? googleUser = await _googleSignIn.signIn();

    if(googleUser == null) return null;

    final GoogleSignInAuthentication googleAuth = await googleUser.authentication;

    final AuthCredential credential = GoogleAuthProvider.credential(
      accessToken: googleAuth.accessToken,
      idToken: googleAuth.idToken
    );
    final UserCredential result = await _auth.signInWithCredential(credential);
    return result.user;
  }catch(e)
  {
    print('Error al iniciar sesion con google: $e');
    return null;
  }
}

Future<void> logout() async {
  try{
    await _auth.signOut();
    await _googleSignIn.signOut();
  }catch(e)
  {
    print('Error al cerrar sesión: $e');
  }
}
}
```

Login con Google y logout total de la Firebase y de la aplicación.

b. Procesamiento de imágenes

Funciones para el procesamiento de imagenes. Se pasa de imagen a base64, de base64 a widget de Flutter y de base64 a fichero.

```dart
static Future<String> imageFileToBase64(File imageFile) async {
  try {
    final bytes = await imageFile.readAsBytes();
    return base64Encode(bytes);
  }catch(e)
  {
    throw Exception('Error converting image to Base64: $e');
  }
}

static Widget base64ToImageWidget(String base64String) {
  try {
    final Uint8List imageBytes = base64Decode(base64String);
    return Container(
      width: double.infinity,
      constraints: BoxConstraints(
        maxHeight: 300,
      ),  // BoxConstraints
      child: Image.memory(
        imageBytes,
        width: double.infinity,
        fit: BoxFit.cover,
      ),  // Image.memory
    );  // Container
  }catch(e) {
    throw Exception("Error converting base64 String to Image Widget: $e");
  }
}
```

```dart
static Future<File> base64ToFile(String base64String) async {
  try {
    final Uint8List imageBytes = base64Decode(base64String);

    final directory = await getTemporaryDirectory();
    final String path = '${directory.path}/image.png'; // Ruta tempora

    final File imageFile = File(path);

    await imageFile.writeAsBytes(imageBytes);

    return imageFile;
  } catch (e) {
    throw Exception("Error converting Base64 String to File: $e");
  }
}
```

c. Extensiones de cadenas

Creación de una extensión de String para poner en mayúscula la primera letra de cada palabra.

```dart
extension StringExtension on String {
  String capitalize() {
    return this
        .split(' ')
        .map((word) => word.isNotEmpty
        ? '${word[0].toUpperCase()}${word.substring(1).toLowerCase()}'
        : '')
        .join(' ');
  }
}
```

6. Widgets

a. Item de receta

El ítem de receta se le pasa a la lista de recetas y favoritos en las páginas de Recetas y Favoritos correspondientes. Por lo tanto las listas de favoritos y recetas tienen ambas la misma interfaz.

b. Item de compra

El ítem de compra tiene el mismo procedimiento que el ítem de receta pero aplicándolo a la lista de compra de la página de Compra.