DYNAMIC GROCERY LIST APP

SYSTEM DESIGN DOCUMENT (SDD)

CS 4850 02

Spring 2024

Professor Perry

February 13, 2024

SP-5 Red

Table of Contents

1.0 Introduction

1.1 Purpose of the SDD

The goal of this System Design Document (SDD) is to describe in detail the design of a mobile grocery list application. The document covers the system's structure, database, user authentication, and real-time features. It will be updated as the project evolves. The SDD will address security and privacy, ensuring user data is safe, logins are secure, and communication is private for a trustworthy shopping experience. By the end of the document, the reader should have enough information to build the application as proposed.

2.0 General Overview and Design Guidelines

2.1 General Overview

The goal of our grocery app is to make grocery list management easy and collaborative. Flutter and dart are being used to ensure the application will be able to run on IOS and Android firmware and Google Firebase for its authentication and database. The design focuses on being user-friendly and adaptable as the project evolves. We want the app to be responsive and easy to use for users.

2.2 Assumptions/Constraints/Risks

2.2.1 Assumptions

The grocery app will require internet connection for real-time updates and sharing features to function effectively. It is assumed that users will have an Android or IOS device to utilize the application. Dependencies include the continued support and compatibility of Google Firebase for database management and user authentication. It is also assumed that users will have a basic understanding of using mobile applications and will engage with the app in a manner consistent with typical grocery shopping behavior. Potential changes in functionality may be required based on user feedback and evolving market trends in grocery app usage. Overall, these assumptions provide a foundational understanding for the project's development and user engagement scenarios.

2.2.2 Constraints

The grocery app has several constraints in its design. Users need compatible smartphones, an Android or IOS device, and a stable internet connection for the app to work effectively, especially for real-time updates. The app relies on a wireless connection with Google Firebase. These constraints also include a limit of 10 Gigabytes of storage for the free Firebase out application will be utilizing.

2.2.3 Risks

Potential risks for the application include unauthorized access to data stored in users lists. However, this is mitigated by Firebase encrypting data in transit via HTTPS, as well as a layer of encryption on data at rest in their servers. Another possible risk in terms of unauthorized access is the potential of a user being able to access I unauthorized list via the application, to help prevent this potential issue, the application will be utilizing a strict set of rules and permissions to determine who has access to what lists.

3.0 Design Considerations

3.1 Goals and Guidelines

The guidelines of the grocery application have several key points: ensuring the app is fast and responsive, ensuring a user-friendly UI, and ensuring the code of the application is commented and clean to ensure readability. By following these guidelines, we aim to ensure that this application's production will be efficient and result in a reliable application.

3.2 Development Methods & Contingencies

The grocery application is being developed with Flutter and Dart, the approach we are mostly using is object-oriented due to our focus on ensuring flexible design. This approach should allow us to take advantage of Flutter and Dart flexible and lightweight framework to ensure a consistent and smooth development process while dividing the project into multiple smaller parts for use to complete.

If any situations arise that will have a large effect on the development of the application, the team will meet either in person or online in Microsoft Teams to discuss the issue, propose solutions, and work together to resolve the complications in a timely manner.

3.3 Architectural Strategies

For this project, we will use the Dart programming language to develop the application. We will use Flutter to design the User Interface and Firebase will manage the database for the application’s data. Firebase also provides the features to manage user authentication and any security concerns for the application. Firebase also supports adding communication mechanisms such as notifications to update users on changes made to their lists.

4.0 System Architecture and Architecture Design

4.1 Logical View

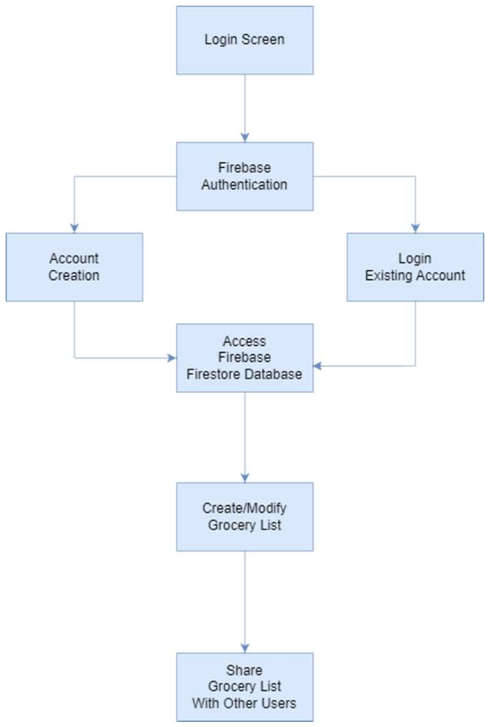

Login Screen
Firebase Authentication
Account Creation
Login Existing Account
Access Firebase Firestore Database
Create/Modify Grocery List
Share Grocery List With Other Users

Fig. 4.1.1 Diagram of the application’s logical view

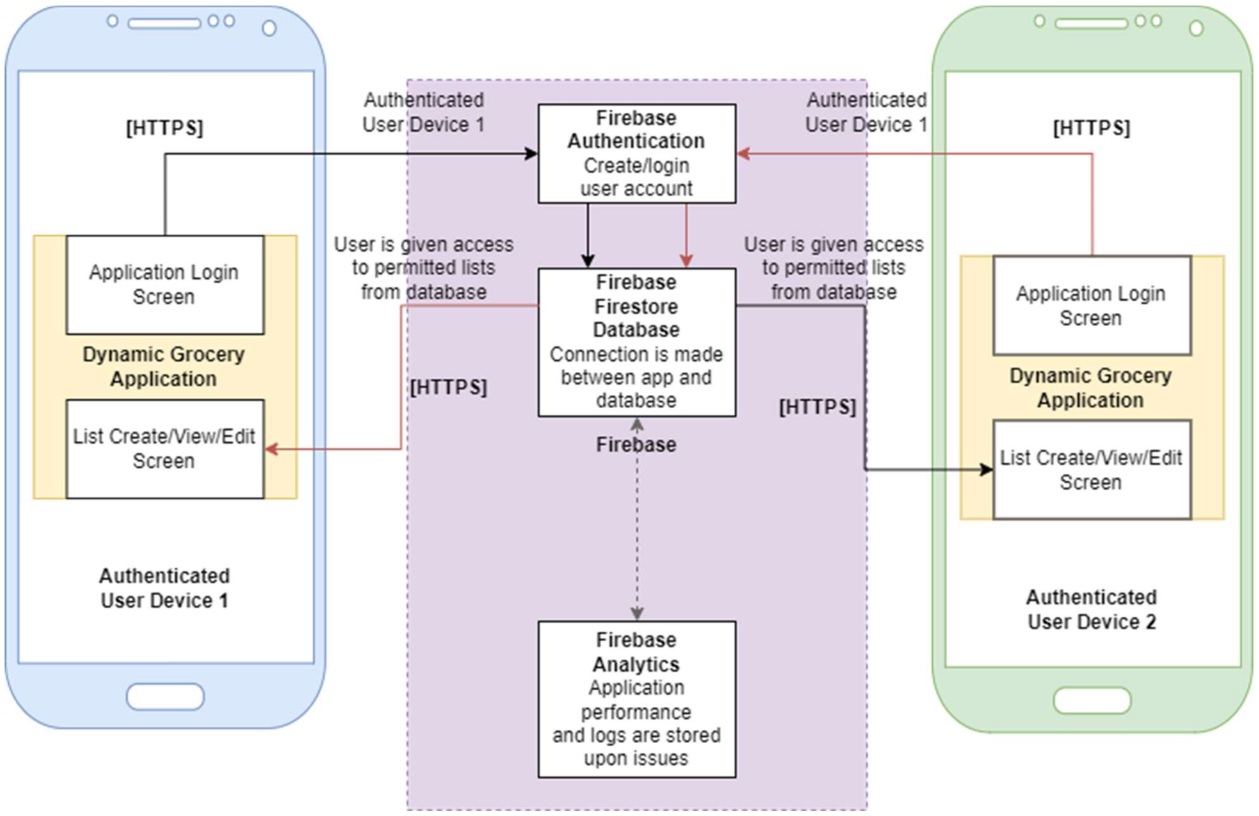
4.1.1 Login and Password

In the grocery app, the login and password will serve as a primary means of authentication in verifying user accounts. Users will be able to create login username and password to access their account and use the app's main features.

4.2 Software Architecture

The software architecture for the Dynamic Grocery App revolves around simplicity and functionality. Utilizing the Flutter framework for the front end and Firebase for authentication and real-time database storage, the app ensures a seamless user experience. Dart serves as the primary language, with the operating system targeting either Android or iOS. The architecture prioritizes scalability and real-time collaboration, aiming for a straightforward and effective app experience.

After a user is signed in, they are given access to Firebase's Firestore database through their UI, where they can create, edit, and share grocery lists.



[HTTPS]
Application Login Screen
Dynamic Grocery Application
List Create/View/Edit Screen
Authenticated User Device 1
Authenticated User Device 1
Firebase Authentication Create/login user account
Authenticated User Device 1
User is given access to permitted lists from database
[HTTPS]
Firebase Firestore Database Connection is made between app and database
User is given access to permitted lists from database
[HTTPS]
Firebase
Firebase Analytics Application performance and logs are stored upon issues
[HTTPS]
Application Login Screen
Dynamic Grocery Application
List Create/View/Edit Screen
Authenticated User Device 2

Fig. 4.2.0.1 Diagram of the application's process flow, made by Cole Young

4.2.1 Security Software Architecture

The Security Software Architecture of our grocery app prioritizes user data protection. To verify user identity, the app will leverage Firebase Authentication, ensuring secure access. Once logged in, users receive specific permissions based on their roles, to allow the ability to edit or view lists. For secure communication and data exchange with the database, the app will rely on Firebase's built-in HTTPS, ensuring the confidentiality and integrity of the information being sent and retrieved. For analytics

Firebase's built-in analytics and monitoring will be utilized to ensure users are receiving a high level of performance.

4.2.2 Performance Software Architecture

To keep our grocery app running smoothly and reliably, we use Flutter for a quick and efficient user interface and Firebase Firestore Database for fast data access. However, since we are utilizing the free Firebase access plan, the database we are utilizing could go down and we have limited control over it. If this server were to go down, the application will be unable to do live updates to and from the database.

4.3 Information Architecture

4.3.1 Records Management

4.3.1.1 Data

The users will supply information to the application such as login information when first creating an account. They will also enter the names of the grocery lists and any items they wish to add to their lists. All user data will be entered through the graphical user interface developed for the application. Grocery list items are stored as an array of strings, these arrays are stored in a collection of lists that are paired with a user ID. The owner of each list can then assign editor or viewer permission to other users through their email through the users' collection.

4.3.1.2 How Data is Stored in the Database

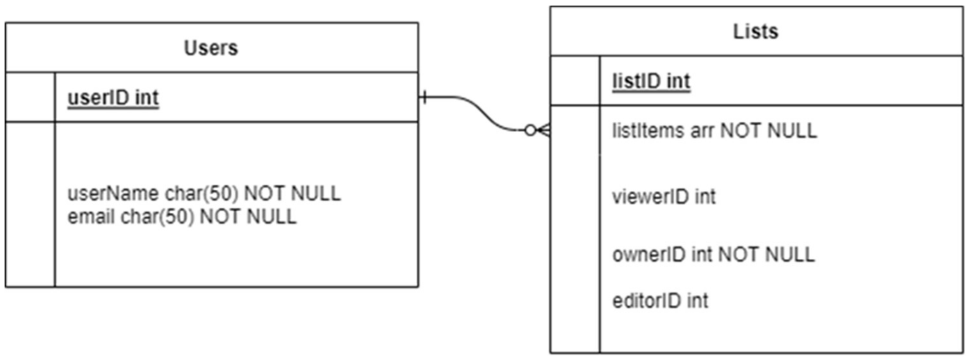


Fig. 4.4.1.2.1 Database Design Diagram

4.4 Security Architecture

In our grocery list app, Firebase Authentication ensures that only authenticated users can use the application by managing sign-ups and logins securely. Firebase Firestore rules control data access, specifying who can read or write data based on user roles. For instance, owners have broader access compared to editors or viewers. These rules prevent unauthorized access, securing user data and maintaining data integrity. Together, Firebase Authentication and Firestore rules create a simple yet effective security system for our application.

4.5 System Architecture Diagram

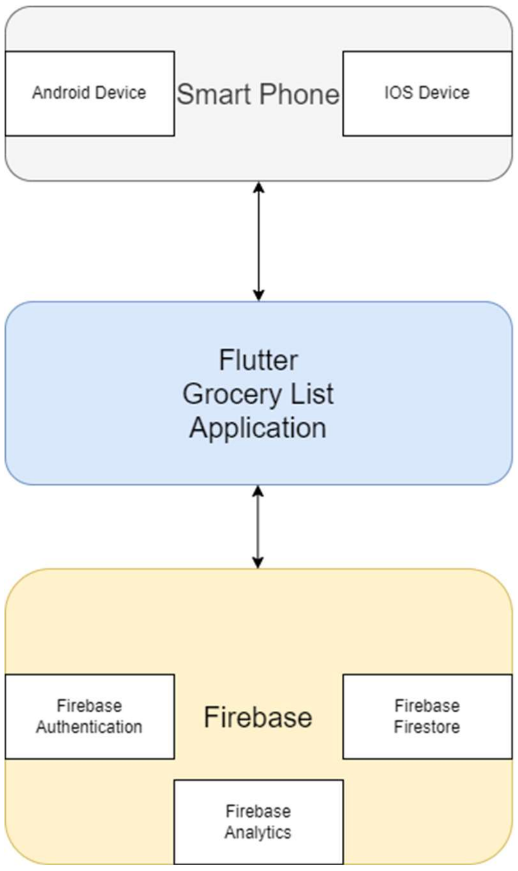


Fig. 4.8.1 Reference for information flow between smartphone, application, and Firebase

5.0 System Design

5.1 User Interface Requirements

The user interface design should be intuitive, aesthetically pleasing, and responsive. Our design includes four main views: the login page, the page containing all the user's lists, another page to allow users to edit and view the contents of as well as share the list with another user, and finally a view to allow users to search for keywords in their grocery lists.

5.2 Database Design

The database is managed by Firebase's Cloud Firestore, a noSQL database. The diagram in 4.4.1.2 shows the data stored in the database and how the tables interact.

5.3 User Interface Design

The following diagrams are examples of how the application's user interface will look.

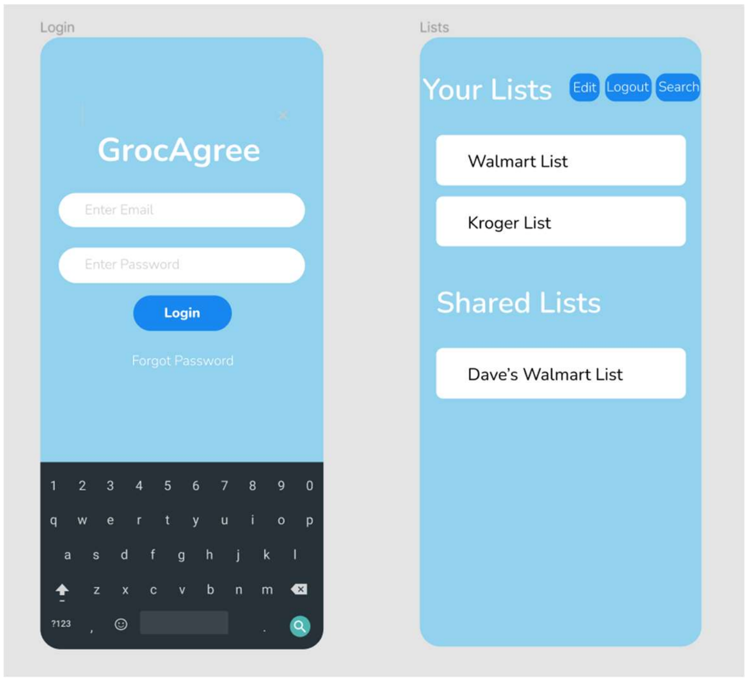


Fig. 5.3.1 Design of login screen and grocery list view

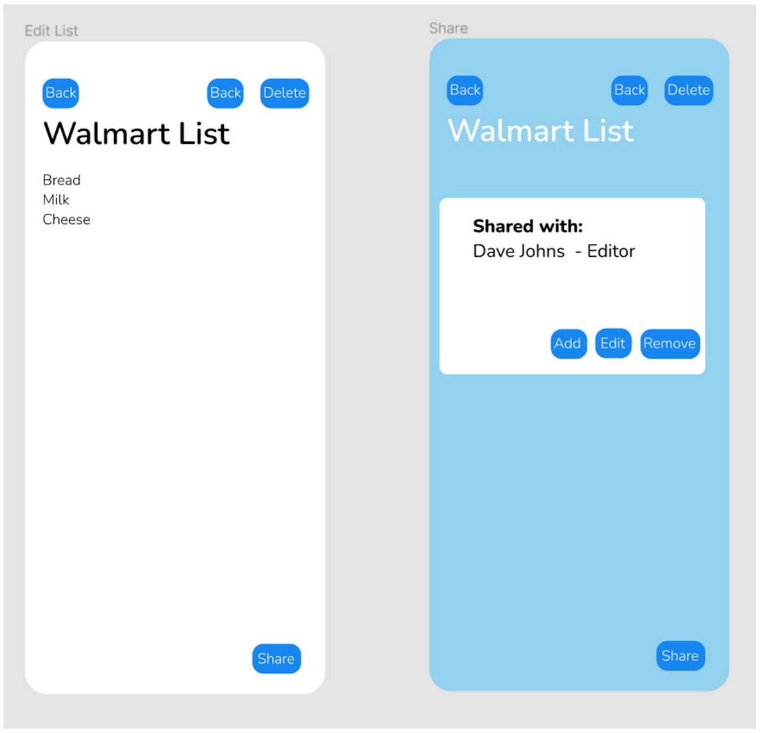


Fig. 5.3.2 Design of grocery list and share option

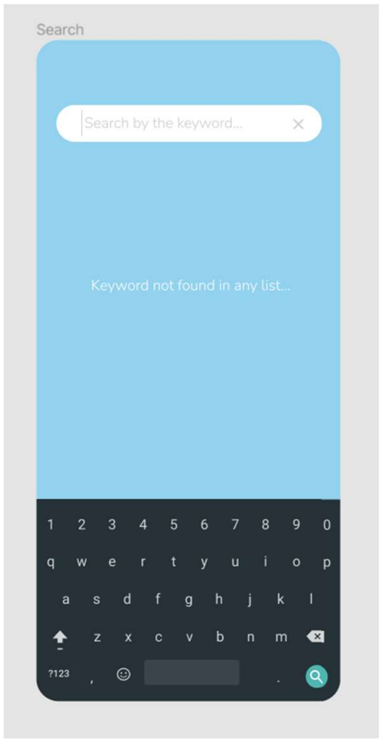


Fig. 5.3.3 Design of search function

6.0 Operational Scenarios

Scenario:

- User1 will create a grocery list and share it with a friend, User2.

User1 Actions:

- User1 will log into their account.
- User1 will navigate and use the “Create New List” button.
- User1 will add desired items to the list.
- After the list is complete, user1 can share it with user2.

System Actions:

- The grocery list will be saved in the history of a user1’s account.
- The recipient, user2, of the list will be sent a notification indicating the list has been shared with them.

User2 Actions:

- User2 receives the notification that the list has been shared with them.
- User2 can now select to view the list shared from user1 on their grocery application.

System Actions:

- The application displays the shared grocery list in real-time to user2.
- User2 is now able to see and interact with the list as they desire (add, delete, or edit).

User2 Actions:

- User2 needs additional items, so they add more to the list.

- User2 marks all items that they already have.

System Actions:

- The application will update the shared list in real-time, showing user1 all the changes user2 made.
- User1 will be sent a notification that the list has been modified.

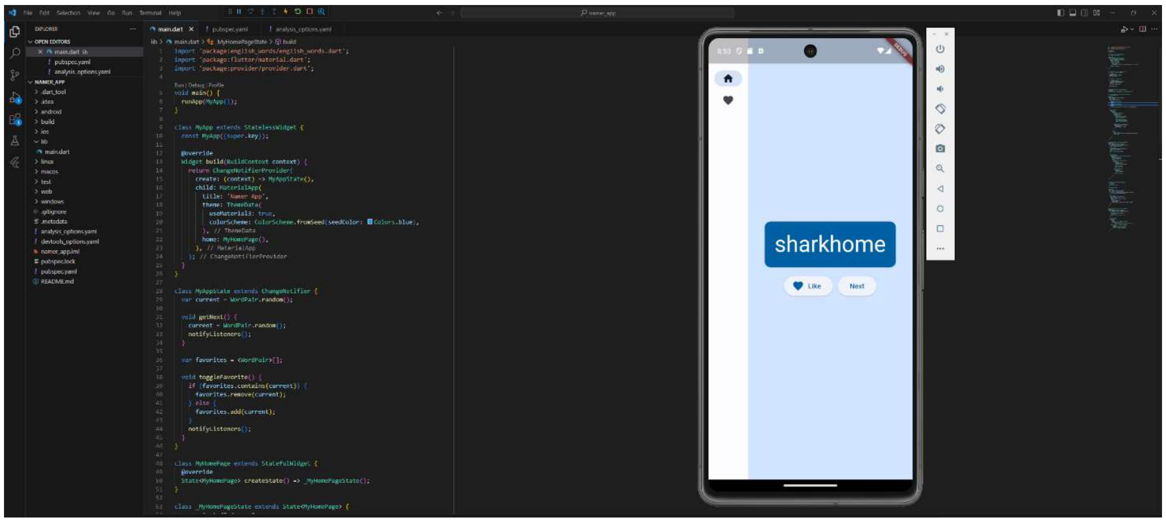
User1 Actions:

- User1 sees the notification and opens the application.
- User1 is now able to view the changes made by user2 in real-time.

System Results:

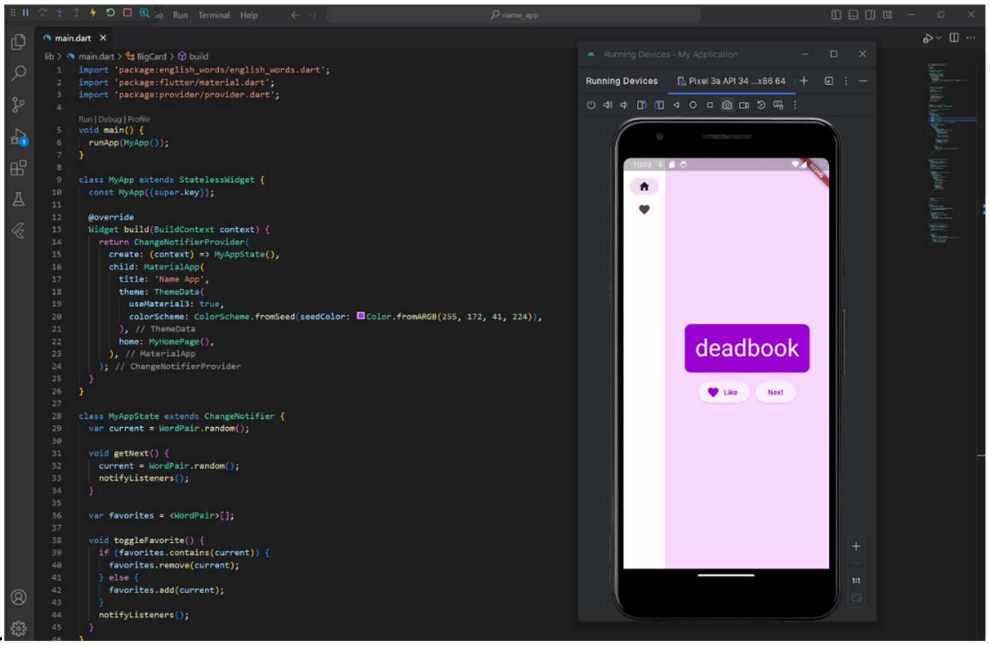
- Now both users can view the latest version of the shared grocery list with all the updates synchronized in real-time.

APPENDICES



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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MyAppState(),
      child: MaterialApp(
        title: 'Namer App',
        theme: ThemeData(
          useMaterial3: true,
          colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
        ), // ThemeData
        home: MyHomePage(),
      ), // MaterialApp
    ); // ChangeNotifierProvider
  }
}

class MyAppState extends ChangeNotifier {
  var current = WordPair.random();

  void getNext() {
    current = WordPair.random();
    notifyListeners();
  }

  var favorites = <WordPair>[];

  void toggleFavorite() {
    if (favorites.contains(current)) {
      favorites.remove(current);
    } else {
      favorites.add(current);
    }
    notifyListeners();
  }
}

class MyHomePage extends StatefulWidget {
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
```

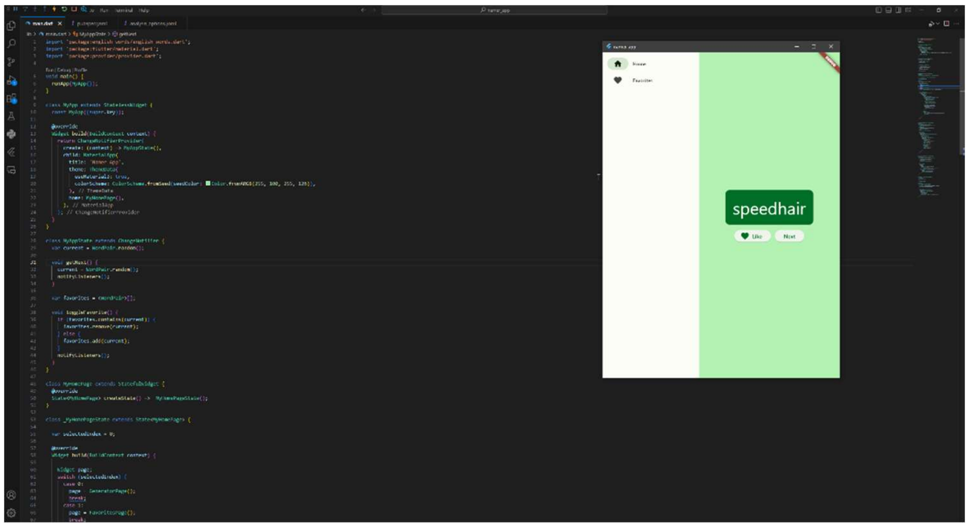
Cole Young completed Flutter tutorial screenshot.


Run
Terminal
Help
name_app
main.dart
lib > main.dart > BigCard > build
Run | Debug | Profile
Running Devices - My Application
Running Devices
Pixel 3a API 34 ...x86 64
deadbook
Like
Next

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MyAppState(),
      child: MaterialApp(
        title: 'Name App',
        theme: ThemeData(
          useMaterial3: true,
          colorScheme: ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 172, 41, 224)),
        ), // ThemeData
        home: MyHomePage(),
      ), // MaterialApp
    ); // ChangeNotifierProvider
  }
}

class MyAppState extends ChangeNotifier {
  var current = WordPair.random();

  void getNext() {
    current = WordPair.random();
    notifyListeners();
  }

  var favorites = <WordPair>[];

  void toggleFavorite() {
    if (favorites.contains(current)) {
      favorites.remove(current);
    } else {
      favorites.add(current);
    }
    notifyListeners();
  }
}
```

.

Elizabeth Krummert completed Flutter tutorial screenshot.



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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => MyAppState(),
      child: MaterialApp(
        title: 'Namer App',
        theme: ThemeData(
          useMaterial3: true,
          colorScheme: ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 100, 255, 120)),
        ), // ThemeData
        home: MyHomePage(),
      ), // MaterialApp
    ); // ChangeNotifierProvider
  }
}

class MyAppState extends ChangeNotifier {
  var current = WordPair.random();

  void getNext() {
    current = WordPair.random();
    notifyListeners();
  }

  var favorites = <WordPair>[];

  void toggleFavorite() {
    if (favorites.contains(current)) {
      favorites.remove(current);
    } else {
      favorites.add(current);
    }
    notifyListeners();
  }
}

class MyHomePage extends StatefulWidget {
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

  var selectedIndex = 0;

  @override
  Widget build(BuildContext context) {

    Widget page;
    switch (selectedIndex) {
      case 0:
        page = GeneratorPage();
        break;
      case 1:
        page = FavoritesPage();
        break;
```

Aaron Figueroa completed Flutter tutorial screenshot.