Fruit Fairy

A Project Report
Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in Software Engineering

By
Thuan Chau, George Michael Cuevas, Yonatan Greenblum, Xuxiang Huang
05/2021

Copyright © 2021
Thuan Chau, George Michael Cuevas, Yonatan Greenblum, Xuxiang Huang
ALL RIGHTS RESERVED

APPROVED FOR THE COLLEGE OF ENGINEERING

Chung Wen Tsao

Professor Chung-Wen (Albert) Tsao, Project Advisor

Professor Rod Fatoohi, Instructor

Dr. Xiao Su, Computer Engineering Department Chair

ABSTRACT

Fruit Fairy

By Thuan Chau, George Michael Cuevas, Yonatan Greenblum, Xuxiang Huang

Fruit Fairy, a new mobile application, enables home gardeners and anyone in possession of fruit and vegetables to donate to their local charities. The goals of the application are to decrease food waste and to ensure that more of the population has access to a healthier way of life.

Fruit Fairy’s impact on society can be instrumental in solving global, economic, social, and environmental issues. On the economic scale, Fruit Fairy can help solve feeding the poor and alleviate some of the government financing towards the same goal. Regarding the social aspect, people will be more involved in helping the needy which might make them more socially involved in other aspects of life and have a positive effect on society. The environmental impact involved with our application is that hopefully less industrialized farming will be needed to prevent hunger.

Fruit Fairy is made up of two types of users that are needed for the applications success. The first are home gardeners who are in search of a way to donate their excess fruits and vegetables and the second are local charities that are looking for donations of fruits and vegetables. Charities can compile a list of fruits and vegetables they need, and users can donate from their garden excess fruits and vegetables. The application matches specific donors to a list of recommended charities and the user can choose the charity he wishes to donate to. Our application offers the home gardener to have his fruit and vegetables collected and harvested by the charity.

Acknowledgments

The project team would like to express our gratitude to our project advisor Prof. Chung-Wen (Albert) Tsao who guided and supported us in creating our application. Prof. Albert guided us to focus on user experience in developing the application. The team also wishes to thank San Jose State University and its professors in teaching us on how to work as a team and teach us the needed technical skills to create our application.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

George Michael Cuevas/Yonatan Greenblum

1.1 Project Goals and Objectives

The main project goal of Fruit Fairy was to decrease food waste by making the process of donating fruits and vegetables more applicable. This goal is even more essential in the time of a pandemic where people are suffering more than usual. Our objective was to create a user-friendly application that would be a platform to connect home gardeners and local charities. Home gardeners can donate their excess of fruits and vegetables through the application and charities are able to designate in the application what types of fruits and vegetables they need.

1.2 Problem and Motivation

The idea for our project came from walking around our local communities and seeing multiple fruits and vegetables lying on people's front yards and going to waste. Our application puts a stop to this behavior because food waste is a major concern worldwide. According to foodprint.org, America wastes over forty percent of its food and much of this food is perfectly edible. Food waste has a large price tag on the economy as well.

3.2 Project Application and Impact

Our application has a global, economic, social, and environmental impact. Our application helps feeding the poor and alleviate the government paying to do so. According to statista.com, the cost of SNAP, the supplemental nutrition assistance program funded by the United States government, cost in 2019 over 60 billion dollars and in the year 2020 over 85.6 billion dollars amid increasing poverty and unemployment during the COVID-19 pandemic. Our app helps feed the poor and as a byproduct lower the amount of spending by local and federal government. Those funds can be allocated to other programs. Our application can be extended globally once it shows promise in a local setting. On a global scale, our application solves futuristic problems.

The nature of our application involves a win-win situation where gardeners get their gardens harvested for them and a percentage of the yield goes towards the needy. People realized that these win-win situations exist, and it led towards more of these types of opportunities and fostered the relationship between the fortunate and misfortunate. People became more involved in helping the needy and made them more socially involved in other aspects which led to having a positive effect on society.

The environmental impact involved with our application is that less industrialized farming is needed to prevent hunger. Industrialized farming has in some cases, negative effects on the environment such as the use of fertilizer that have high ammonium emissions. Focusing on utilizing home grown produce is reducing negative effects on society.

3.2 Project Results and Deliverables

The result of the project is a functioning mobile application that serves as a suitable platform between home gardeners and charities. The application can be used on both for iOS and android devices.

1. Project Plan
    a. Structured plan on how we will advance our project.
    b. Possible deadlines
    c. Major milestones
    d. Key risks
2. Requirements Document
    a. Project Scope
    b. Functional/Non-functional requirements
    c. Operating environment
3. Design Document
    a. Contains application design
        i. Sign Up page
        ii. Login page
        iii. Home page
        iv. Donation pages
4. Testing
    a. Applications runs well and has a high availability
    b. Check for bugs

        c. Solve problems

5. User Guides
    a. Informs the user on how to use the application
        i. Registration process
        ii. Donation process
        iii. Receiving process for charities

3.2 Project Report Structure

The following sections of the report include background and related work, project requirements, system design and implementation, tools and standards used to complete the project, testing procedure and conclusion and future work.

Chapter 2 Background and Related Work

Yonatan Greenblum

2.1 Background and Used Technologies

There are many technology stacks that can be used in creating an application. According to Fingent.com and esparkinfo.com, the most used tech stack is the MEAN Stack which involves MongoDB, Express.js, AngularJS and Node.js (Thomas,2020). There are multiple stacks that can be used to create the application. Our preference is using the Flutter framework which allows for making the application available for both Android and iOS devices with minimum downtime. The flutter framework also allows for asynchronous programming in an easy approach. In addition, the application utilizes Firebase Authentication services and Firestore services to handle the backend portion of the application.

Course	Application
CS-146 Data Structures and Algorithms	Map data structure. Usage of mapping routes between screens of the application.
CS-174 Server-side Web Programming	Client side and server-side validation of user input. Authentication and application security.
CS-172 Enterprise software platforms	Usage of 3rd enterprise software – Firebase and Firestore for application backend.

SE-187 Software Quality Engineering	Testing our software in multiple stages using Agile methodology.

Table 2.1.1: Courses used in project

2.2 Literature Search

The literary research for developing the application included scholarly research and investigating other applications and websites that focus on donating fruits and vegetables. The research that we have used is focused on the software development process, tools, and usability guidelines. Regarding the software development process, according to Moravcik, one of the most followed ideas for software design is the usage of Object-relational mapping (ORM) due to its wide field of usability. An example would be a three-tiered structure which includes the Presentation Layer, Business Layer (Business logic) and the Persistence Layer (Database) which our application will be following. (Moravcik, 892).

Using ORM has its benefits and disadvantages. The benefits include support for many database systems, database creating and updates, automatic mapping of relations, support for transactions, strong query mechanism and notifications when a change occurs. The disadvantages are present when we have an issue of large amounts of data, the impossibility of joining non relatable objects and querying calculated attributes which decreases query time performance. (Moravcik, 894).

Usability is a fundamental aspect in creating a mobile application. Our application must follow past research conclusions to achieve efficient usability of the application. According to Hsiao, Lui and Wang, who completed a study which focused on touch screen devices, usability is an especially important factor regarding an older population who might struggle more with using a touchscreen device. Our application will largely be focused on homeowners which tend to be part of the older population. (Hsiao, 2013). According to Hsiao, positioning the buttons on the bottom part of the screen, and increasing font size all helped the older population when using an application. Our team intends to follow these practices when developing our application.

2.3 State-of-the-art Summary

From our research, there are websites that accept fruit and vegetable donations from home gardeners such as ampleharvest.org and shfb.org. Those websites accept donations from home gardeners, but the gardener must drive his produce to the locations and pick his own vegetables from his garden and do not categorize the type of fruit and vegetables that the charity may or may not need to be donated. Our application intends to make this process much easier for gardeners by connecting to them local charity or food pantry through a mobile application. The gardener will be able to choose which fruits and vegetables from his garden he wishes to donate through the application. He will be able to choose what food pantry or charity he wishes to donate to, and the application connects between the home gardener and the local charity. Our application will also be able to determine if the donation by the home gardener might better suit a different local charity then what the home gardener has chosen. We believe that this improved procedure will decrease food waste and make sure each local gardener and local charity fully utilizes their bounty.

Chapter 3 Project Requirements

Xuxiang Huang

3.1 Domain and Business Requirements

Diagram 1: Process summary diagram

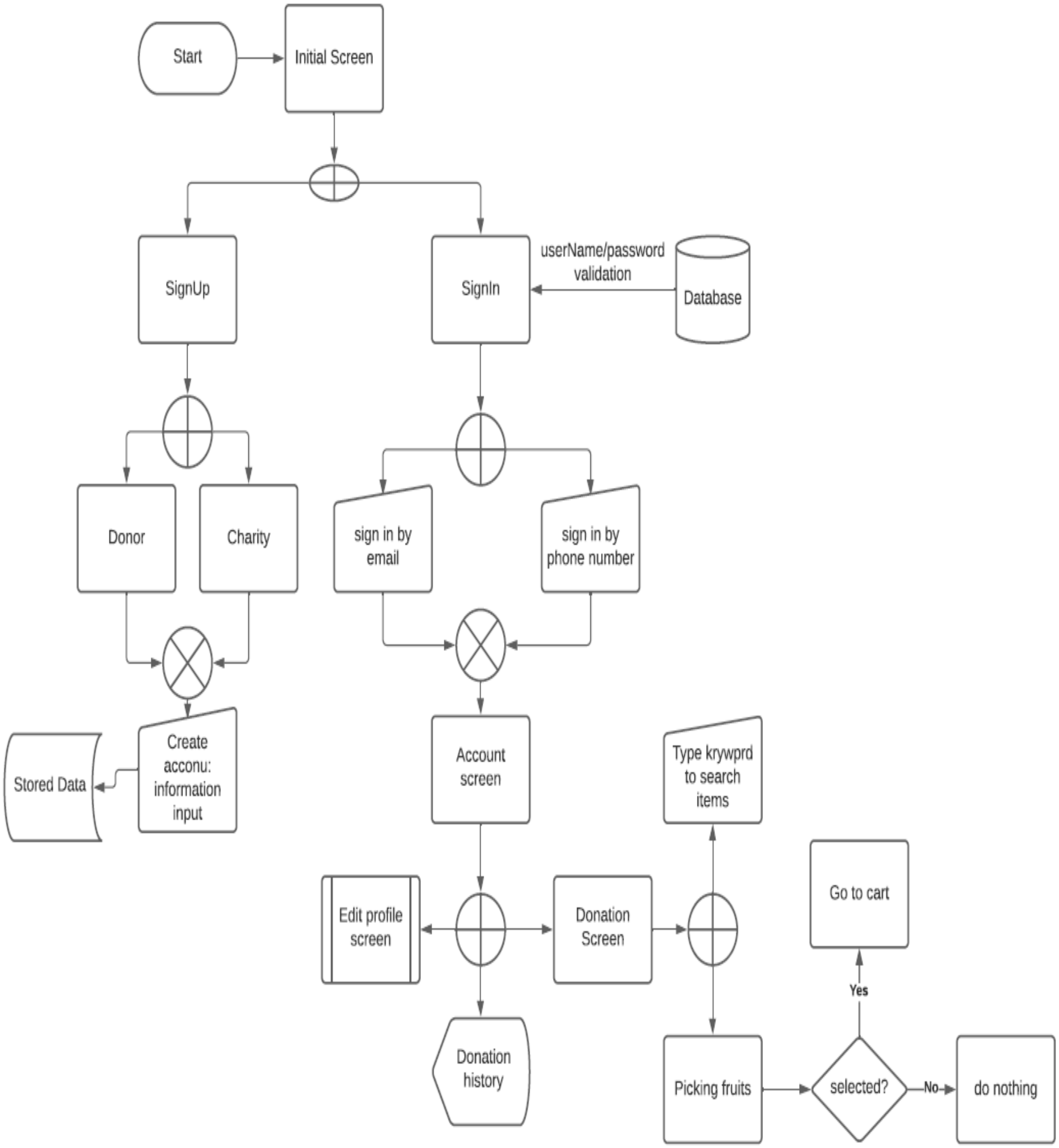


Diagram 2: Edit profile detail diagram

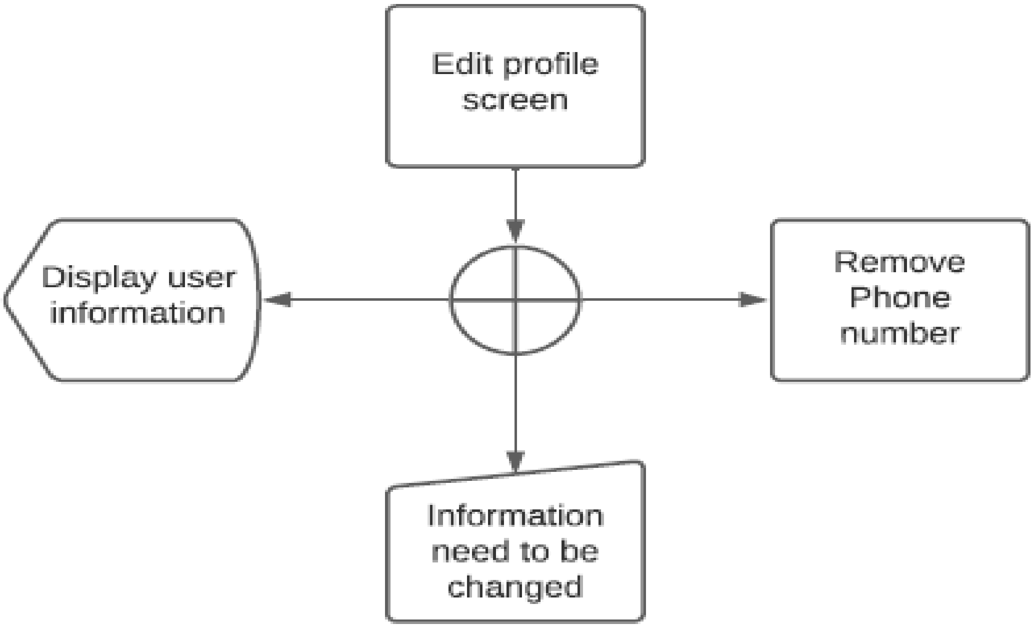


Diagram 3: Domain class diagram

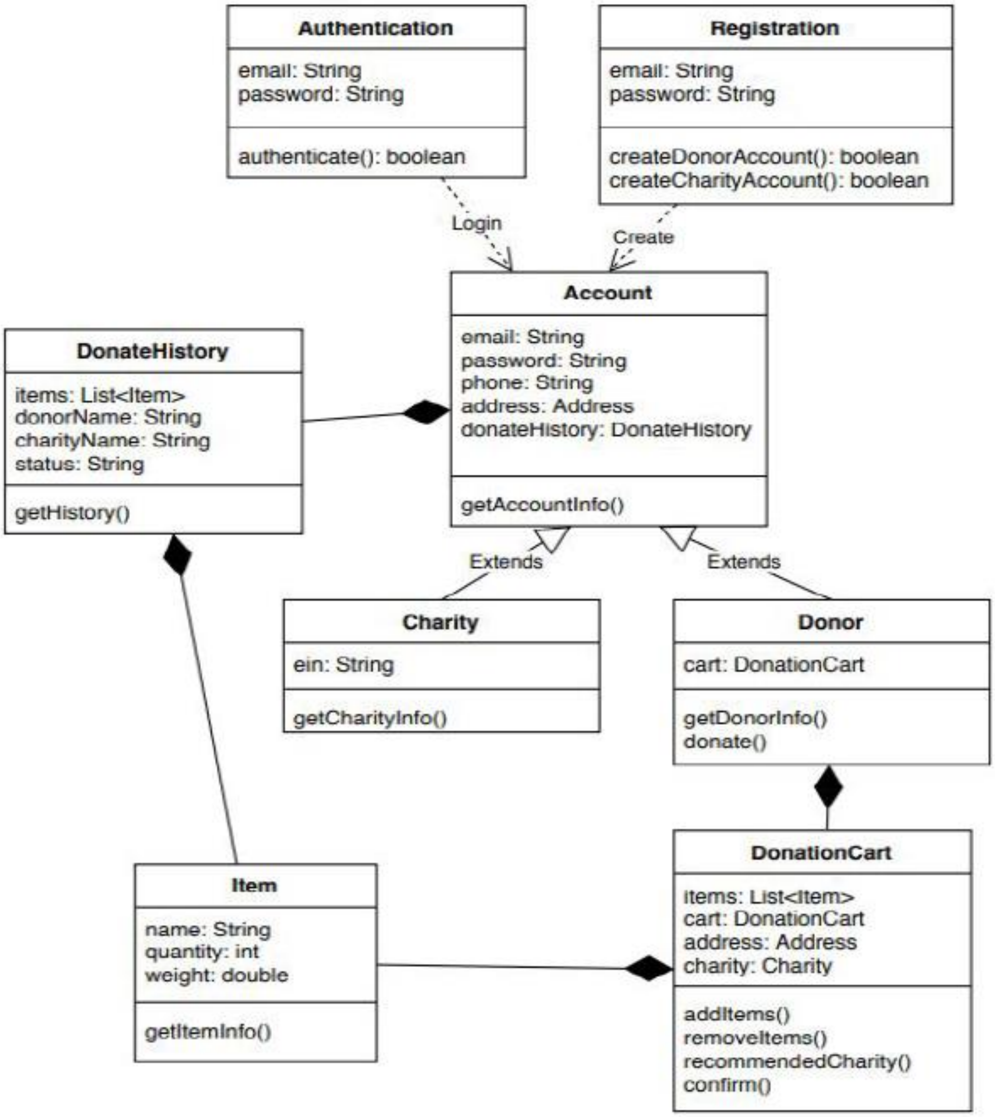


Diagram 4: Sign-up state diagram

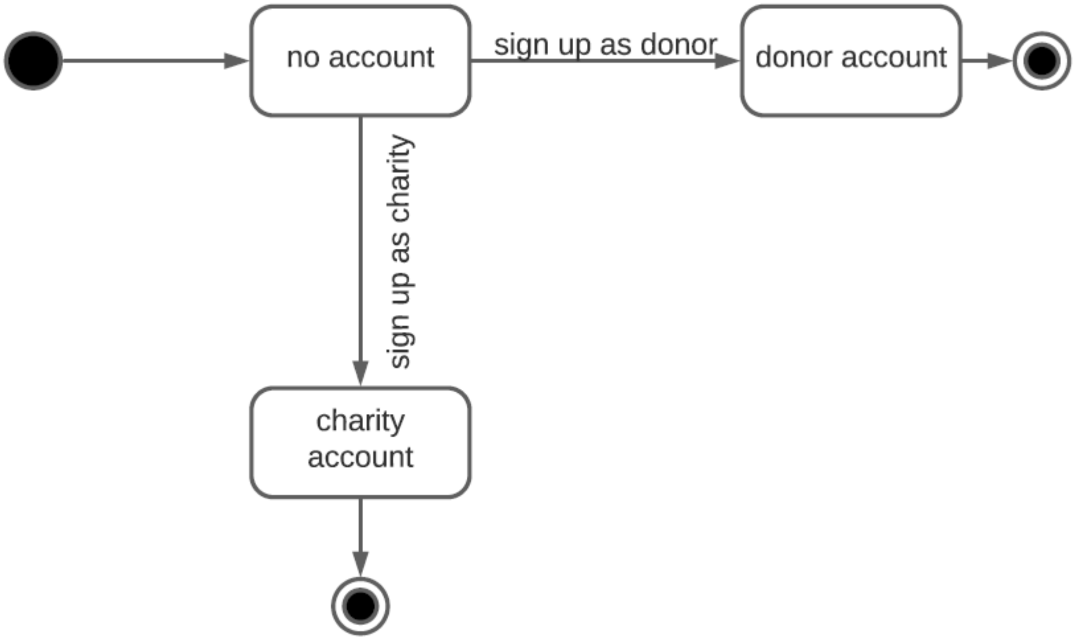


Diagram 5: Edit profile state diagram

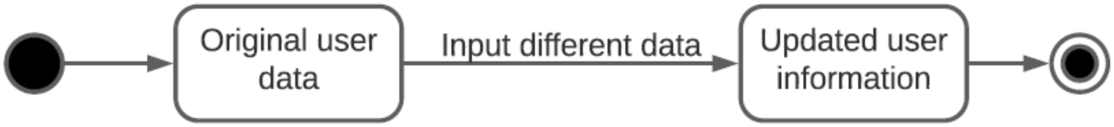


Diagram 6: Picking fruit state diagram

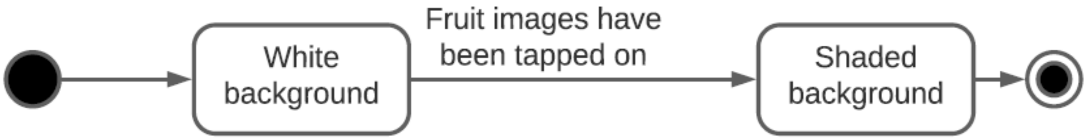
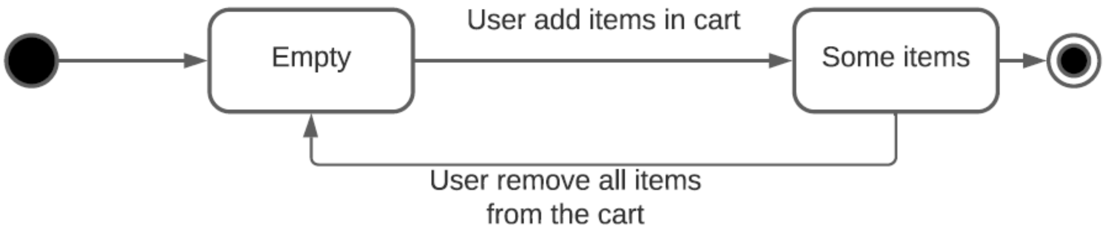


Diagram 7: Donation cart state diagram



3.2 System (or Component) Functional Requirements

Functional Requirement	**Description**
Sign Up	Users shall sign up to the application using their first name, last name, email, and password.
Login/Logout	Users shall login to the application by entering their email and password or via phone number confirmation.
Donor (User who donates fruit and vegetables in the application)	1. The application shall display on the donating process page pictures of fruits and vegetables the user can donate. 2. The application shall display a search bar to search for fruit by name. 3. The application shall have a confirm donation page where the user must enter address and phone number to confirm the donation. 4. The donor shall be presented with three recommendations of charities to choose to donate to, which is based on the charities wish list (made by the charity) and geographical distance. 5. The donor homepage shall include a tracking feature of donations to know if the donation was accepted.

Charity (User who accepts/declines donations from Donors)	1. The application should verify the status of the charity. 2. The application shall allow the charity to select fruits and vegetables to add to their wish list. 3. The application shall allow the charity to remove fruits and vegetables from the wish list. 4. The application shall present the donor’s address information and contact details to the charity once a donation is accepted inside the application and not through email

Table 3.2.1 Functional Requirements

3.3 Non-functional Requirements

Non-functional Requirement	**Description**
Security	1. The application shall not store email/password combinations. 2. The application shall utilize Firebase Authentication services.
Usability	1. A user shall be able to sign up in less than 60 seconds. 2. A donor shall be able to complete his side of the donating process in less than 300 seconds. 3. A charity shall be able to create their wish list in less than 300 seconds.
Reliability	1. The application shall have at most 1 hour of downtime per day. (The application relies heavily on Enterprise software not in our control)

Appearance	1. The application shall have a GUI with non-offensive colors. 2. The GUI will be visible 12-14 inches from the user’s eyes.
Maintainability	1. The application shall be built such that changing certain features such as GUI appearance shall take less than 20 minutes. 2. Adding and removing fruit/vegetable options from the application shall take no more than 5 minutes.
Performance	1. The response time for a successful Sign Up and Login shall be less than 5 seconds. 2. They system shall process three recommended charities to the Donor in less than 30 seconds.

Table 3.3.1 Non-functional Requirements

3.4 Technology and Resource Requirements

The technology and resource requirements used to develop, test, and deploy Fruit Fairy are the following software:

Software:

- IDE:
  - Android Studio 4.1.2/Android Studio 4.2/VS
  - Visual Studio Code 1.53.2
  - Xcode 12.4
- Repository:
  - GitHub

The testing and deployment of the code is executed through a mobile device emulator deployed on PC. The code is deployed and tested on Android and iOS devices.

- Testing:
    - AVD Manager (Android Virtual Device Manager) - For Android devices

The Flutter framework was used to build Fruit Fairy. The compiler for Android is Android's NDK. The compiler for iOS devices is LLVM.

- Databases
    - Firebase Authentication – used for authenticating a user at Login/Sign Up
    - Cloud Firestore – used for holding donation information of each user.

Hardware:

- Testing Android devices is completed with an emulator that is executed on PC environment.
- Testing iOS devices is completed with a physical iOS device.

Chapter 4 System Design

Thuan Chau/Xuxiang Huang

4.1 Architecture Design

The mobile application bridges the gap between local charities and home gardeners by providing an interface for them to seamlessly create interaction. It is made available through fairly new technologies such as Flutter framework and Firebase platform. Flutter is a UI toolkit from Google which will build our mobile application entirely on widgets but still providing the experience of using a native app for both Android and iOS users. Regarding development, the application focused on designing a better user experience without needing to maintain two different codebases by utilizing just one programming language called Dart. With Dart, we used built-in UI and created our own widgets from the Flutter library and at the same time provided the business logic for the application. The application was created using a serverless architecture utilizing Firebase. It is a platform also created by Google that provides backend services for small projects. Cloud Firestore, a storage solution from Firebase, helps us keep data of the user which can also be scaled up with growth from the number of users in the future. Authentication is another service from Firebase that was used to securely authenticate our users through the internet.

The main data flow in our mobile application goes from the donor to the database then to the charity and vice versa. The donor has an option to donate where they can choose which local charity to donate, the fruit or vegetable and donating amount. A record is created in the database with “Pending” status and the chosen charity is notified about the request where they can either accept or deny. When accepted, the record changes its status to “In-progress” and the charity is presented with the donors registered address and collects the fruit or vegetable according to the record. After the collecting is done, the charity marks the record as “Completed”, the app updates the corresponding record in the database and notifies the donor about the status.

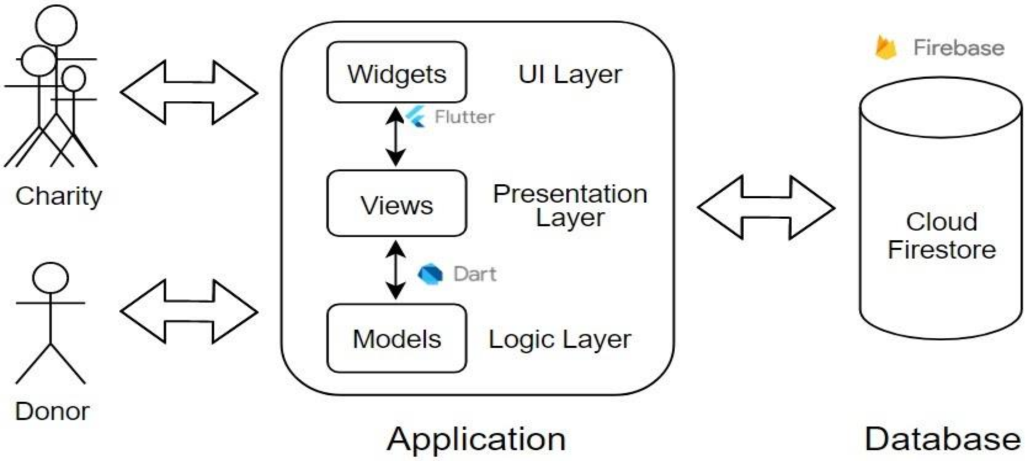
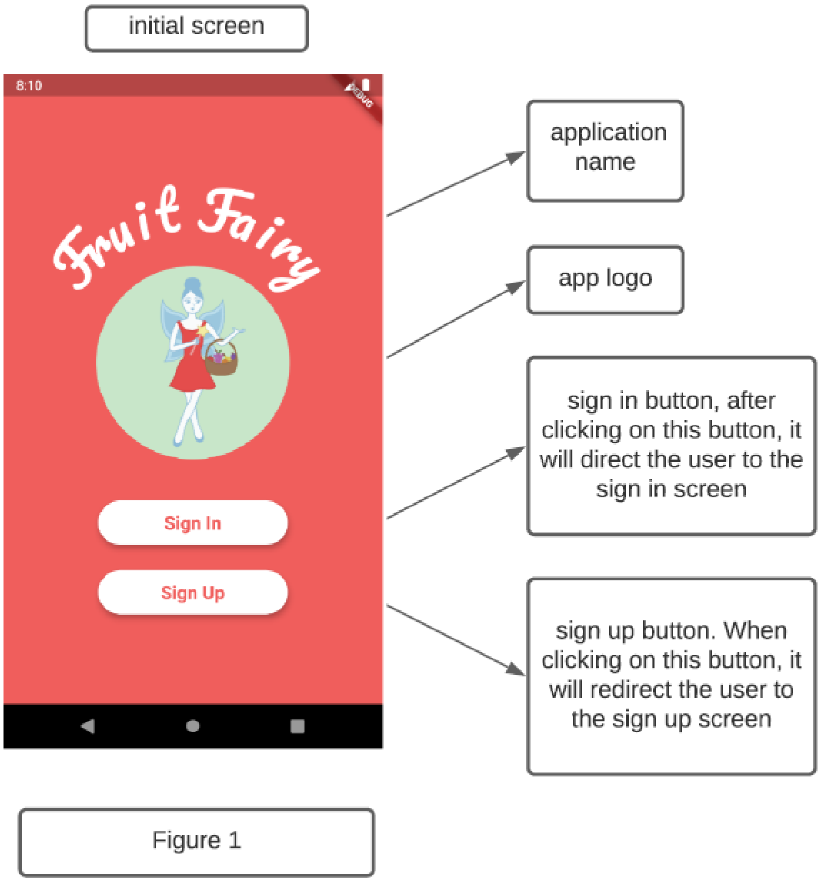


Figure 4.1.1 Architecture design diagram

4.2 Interface and Component Design


initial screen

application name

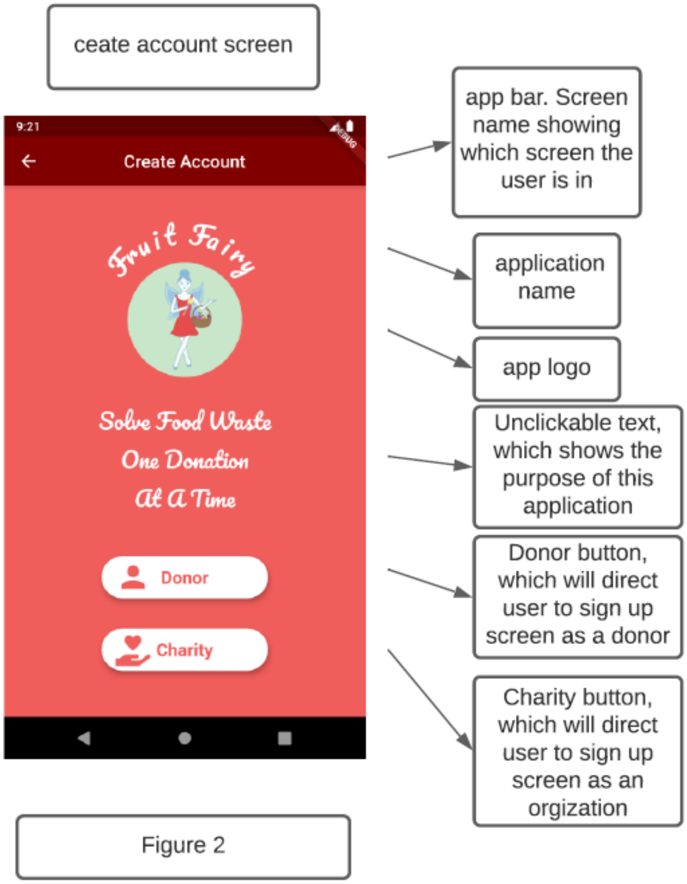
app logo

sign in button, after clicking on this button, it will direct the user to the sign in screen

sign up button. When clicking on this button, it will redirect the user to the sign up screen

Figure 1

Figure 4.2.1 – Initial Screen



ceate account screen

app bar. Screen name showing which screen the user is in

application name

app logo

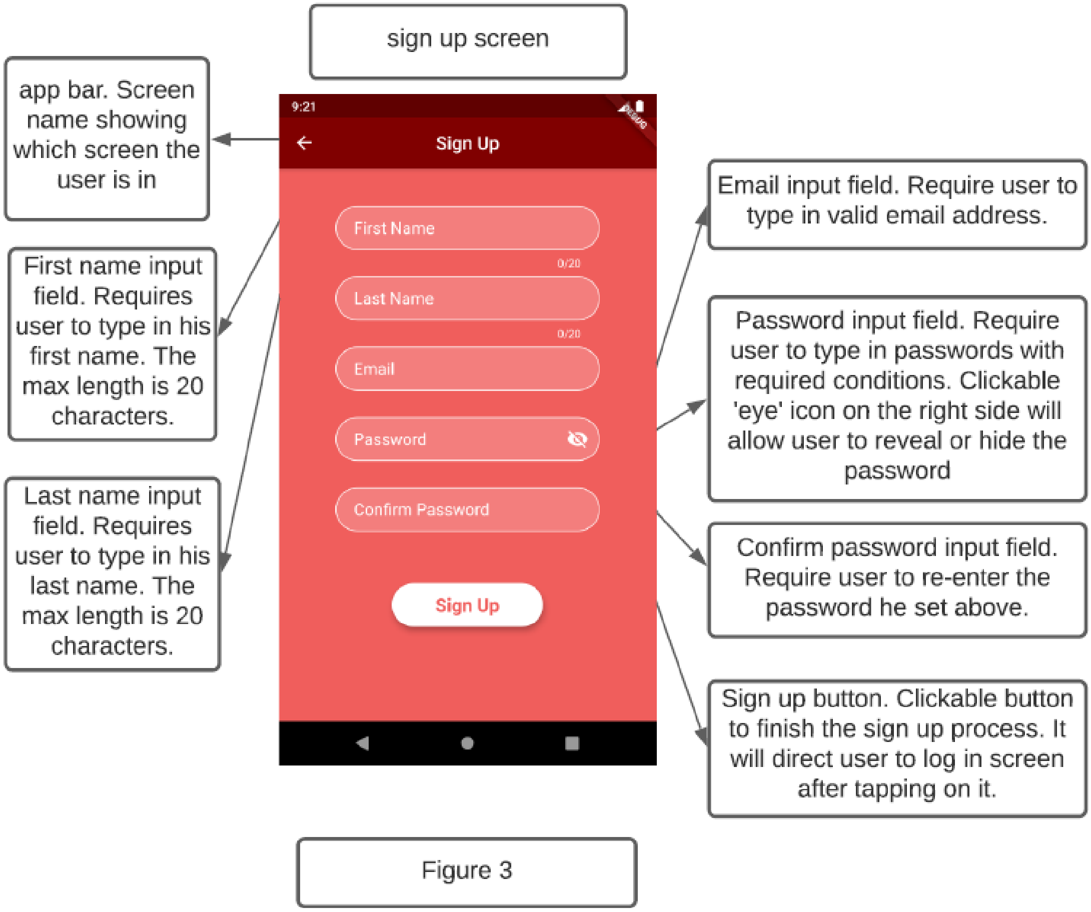
Unclickable text, which shows the purpose of this application

Donor button, which will direct user to sign up screen as a donor

Charity button, which will direct user to sign up screen as an orgization

Figure 2

Figure 4.2.2 – Create Account Screen



sign up screen

app bar. Screen name showing which screen the user is in

First name input field. Requires user to type in his first name. The max length is 20 characters.

Last name input field. Requires user to type in his last name. The max length is 20 characters.

Email input field. Require user to type in valid email address.

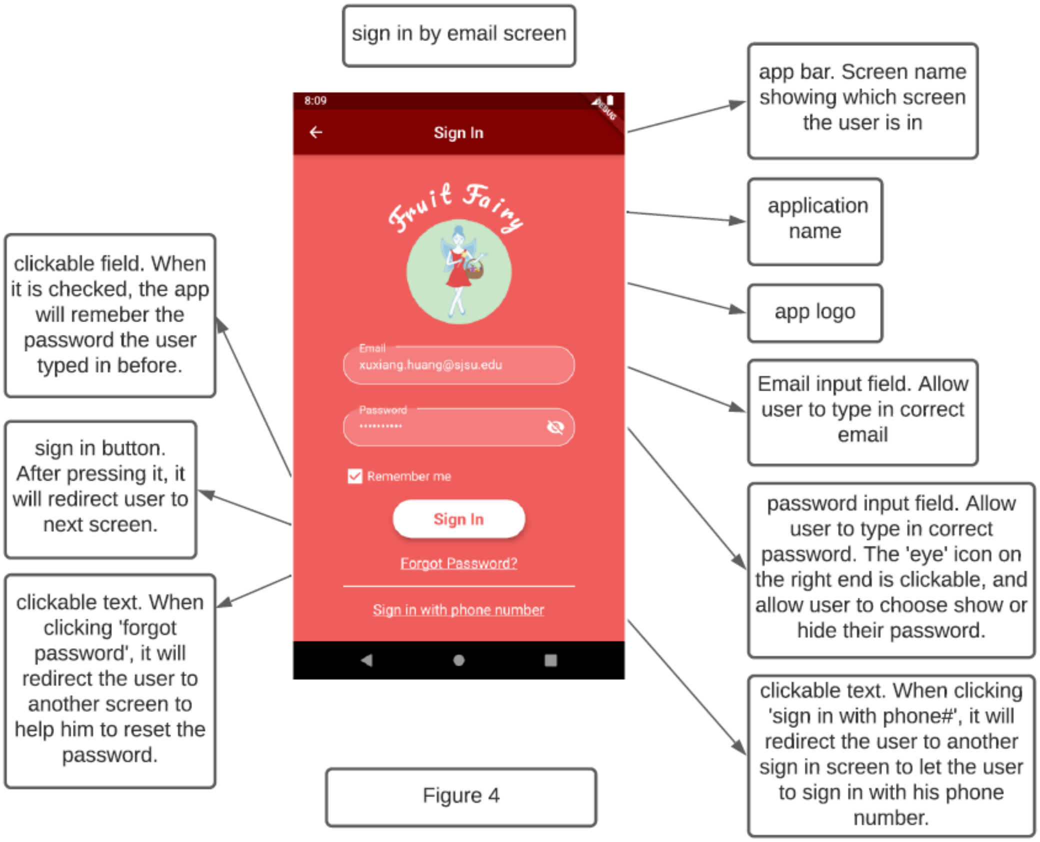
Password input field. Require user to type in passwords with required conditions. Clickable 'eye' icon on the right side will allow user to reveal or hide the password

Confirm password input field. Require user to re-enter the password he set above.

Sign up button. Clickable button to finish the sign up process. It will direct user to log in screen after tapping on it.

Figure 3

Figure 4.2.3 – Sign Up Screen



sign in by email screen

clickable field. When it is checked, the app will remeber the password the user typed in before.

sign in button. After pressing it, it will redirect user to next screen.

clickable text. When clicking 'forgot password', it will redirect the user to another screen to help him to reset the password.

app bar. Screen name showing which screen the user is in

application name

app logo

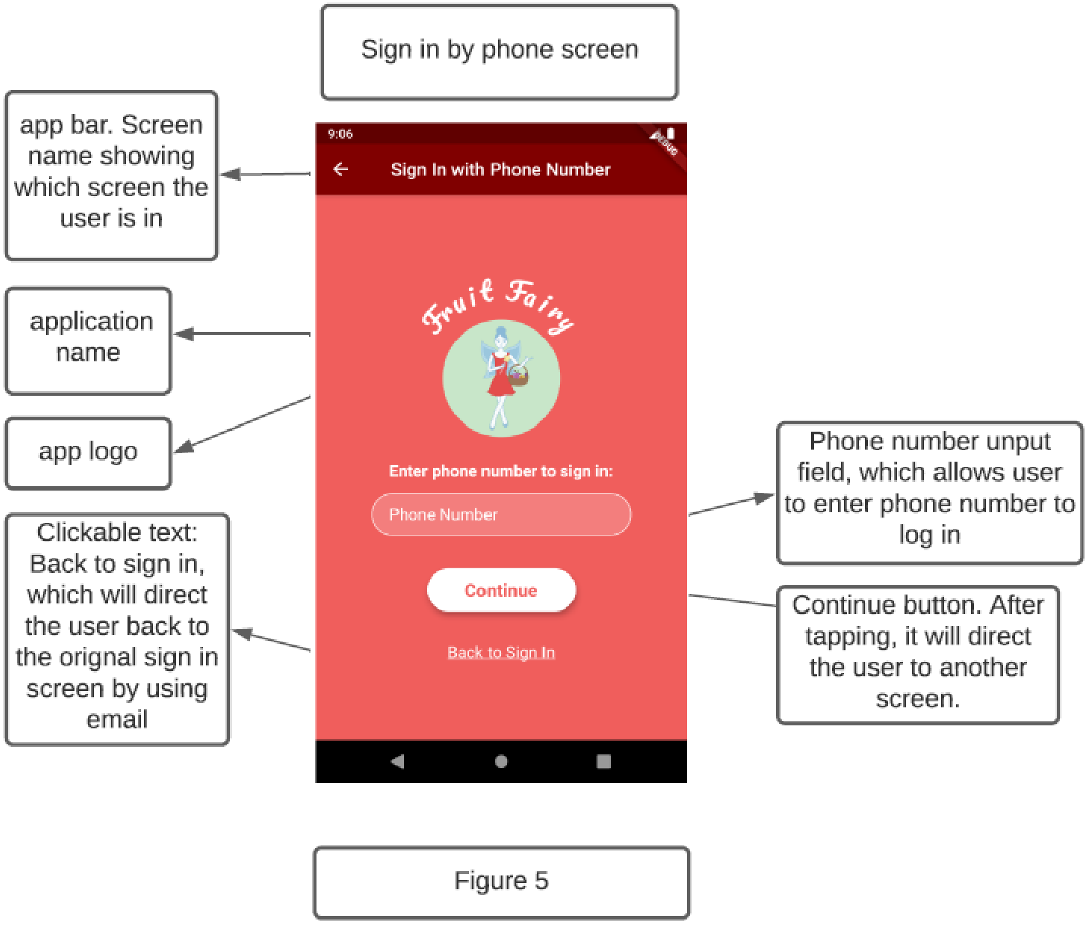
Email input field. Allow user to type in correct email

password input field. Allow user to type in correct password. The 'eye' icon on the right end is clickable, and allow user to choose show or hide their password.

clickable text. When clicking 'sign in with phone#', it will redirect the user to another sign in screen to let the user to sign in with his phone number.

Figure 4

Figure 4.2.4 – Sign in by Email Screen



Sign in by phone screen

app bar. Screen name showing which screen the user is in

application name

app logo

Clickable text: Back to sign in, which will direct the user back to the orignal sign in screen by using email

Phone number unput field, which allows user to enter phone number to log in

Continue button. After tapping, it will direct the user to another screen.

Figure 5

Figure 4.2.5 – Sign in by Phone Screen

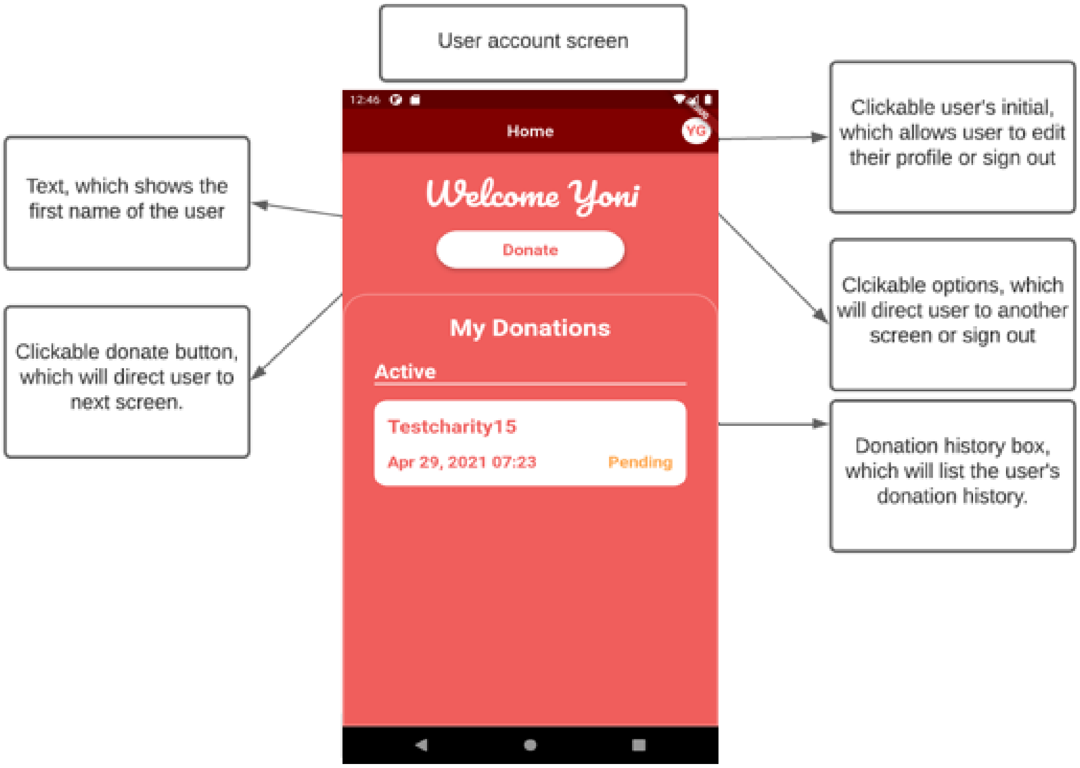

User account screen
12:46
Home
YG
Welcome Yoni
Donate
My Donations
Active
Testcharity15
Apr 29, 2021 07:23
Pending
Text, which shows the first name of the user
Clickable donate button, which will direct user to next screen.
Clickable user's initial, which allows user to edit their profile or sign out
Clcikable options, which will direct user to another screen or sign out
Donation history box, which will list the user's donation history.

Figure 6

Figure 4.2.6 – Home Screen

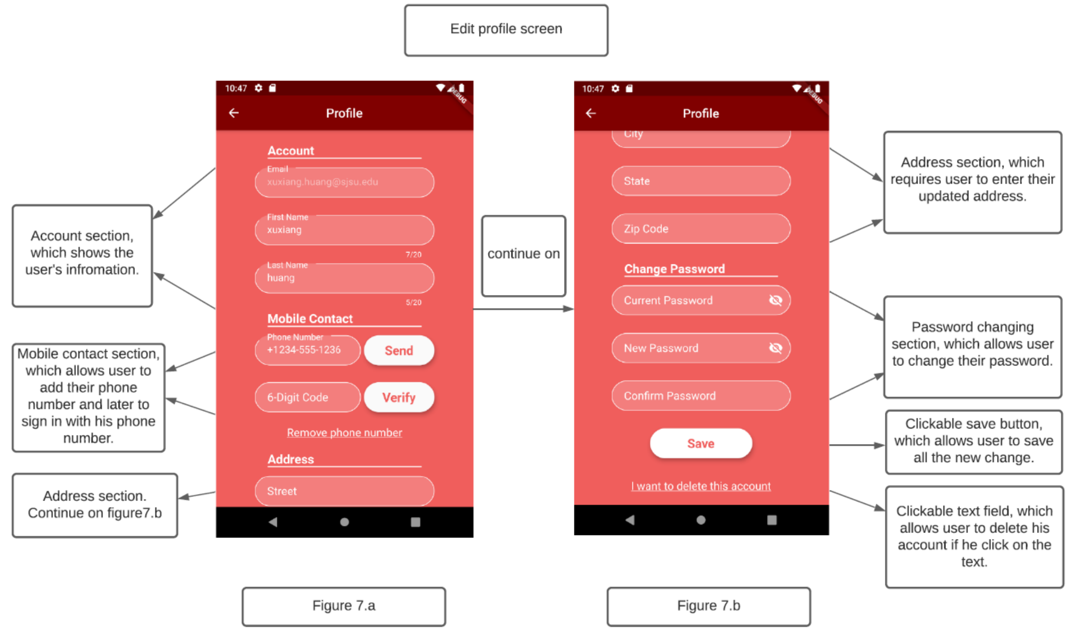

Edit profile screen
Profile
Account
Email
xuxiang.huang@sjsu.edu
First Name
xuxiang
7/20
Last Name
huang
5/20
Mobile Contact
Phone Number
+1234-555-1236
Send
6-Digit Code
Verify
Remove phone number
Address
Street
continue on
Profile
City
State
Zip Code
Change Password
Current Password
New Password
Confirm Password
Save
I want to delete this account
Account section, which shows the user's infromation.
Mobile contact section, which allows user to add their phone number and later to sign in with his phone number.
Address section. Continue on figure7.b
Address section, which requires user to enter their updated address.
Password changing section, which allows user to change their password.
Clickable save button, which allows user to save all the new change.
Clickable text field, which allows user to delete his account if he click on the text.
Figure 7.a
Figure 7.b

Figure 4.2.7 - Edit Profile Screen

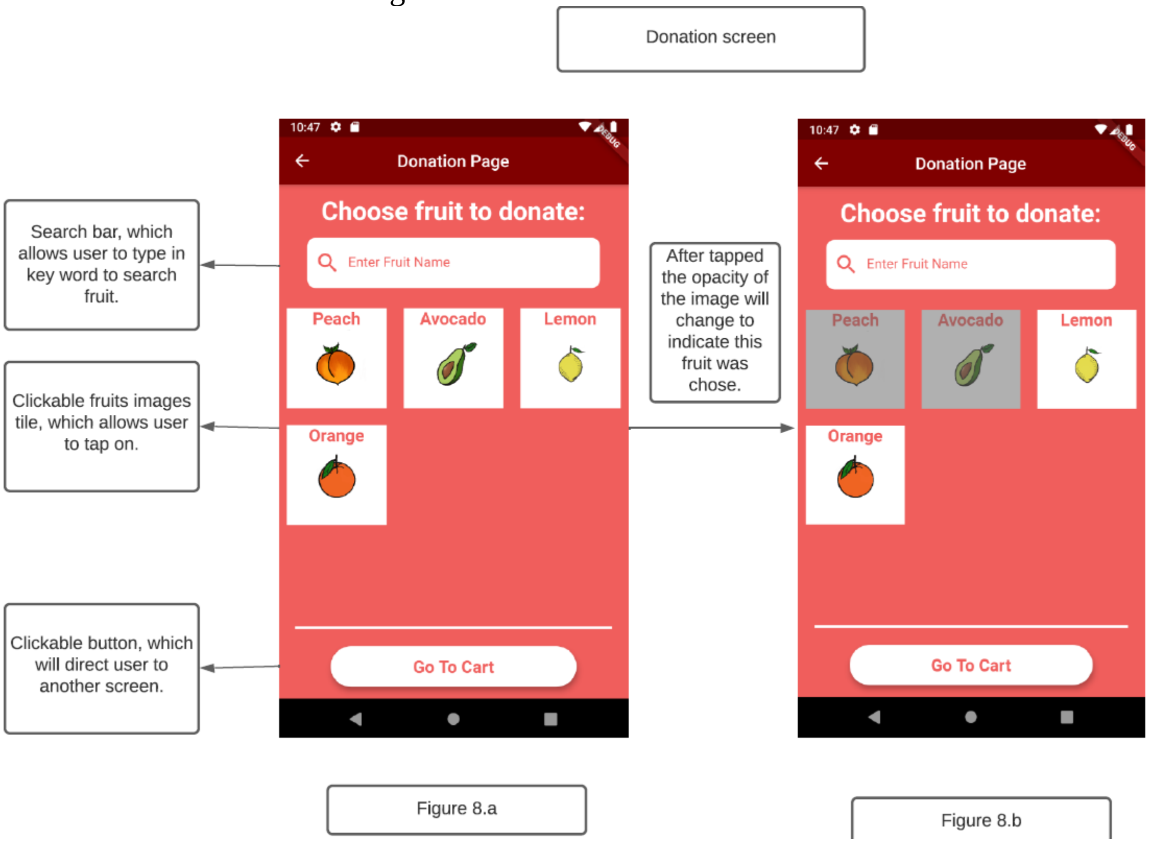

Donation screen
Donation Page
Choose fruit to donate:
Enter Fruit Name
Peach
Avocado
Lemon
Orange
Go To Cart
Donation Page
Choose fruit to donate:
Enter Fruit Name
Peach
Avocado
Lemon
Orange
Go To Cart
Search bar, which allows user to type in key word to search fruit.
Clickable fruits images tile, which allows user to tap on.
Clickable button, which will direct user to another screen.
After tapped the opacity of the image will change to indicate this fruit was chose.
Figure 8.a
Figure 8.b

Figure 4.2.8 – Donation Screen

4.3 Structure and Logic Design

Fruit Fairy is built upon FlutterFire stack, where Flutter is responsible for the UI of the app and Firebase handles authentication, moves data to and from the database, as well as storing it. Fruit Fairy overall structure design can be viewed through figure 4.3.1. Because the logic layout is clearly separated from the implemented stack, we grouped our file structure into two categories (Figure 4.3.2). One group contains only UI components that are placed into two folders, screens, and widget. Screens includes all interfaces that are displayed onto the user's devices to show their information and at the same time, allows them to interact with the app through buttons, dropdowns, and text fields. All screens are built from both default widgets provided by Flutter framework as well as our custom widgets to reduce repetitive code. Another group is responsible for handling data through Firebase framework (services) and holds the data in a more manageable format (models). This layout helped us divide the work for each member.

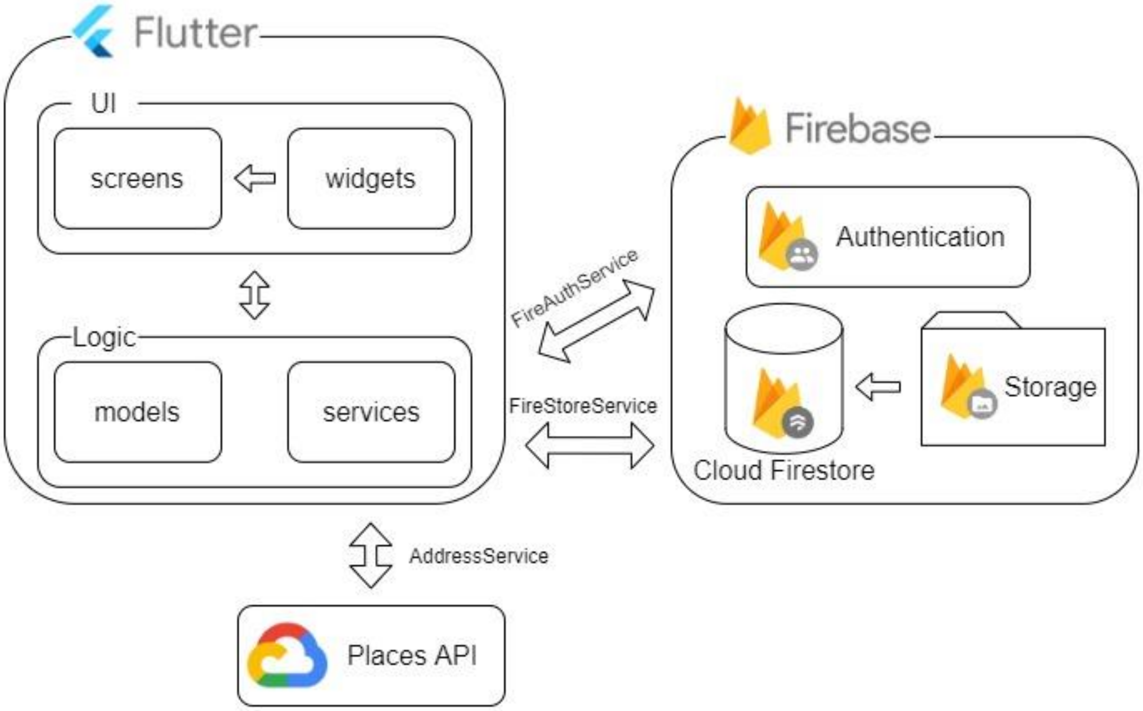
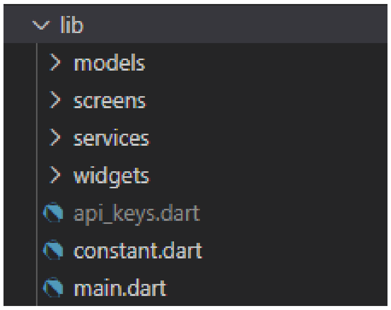
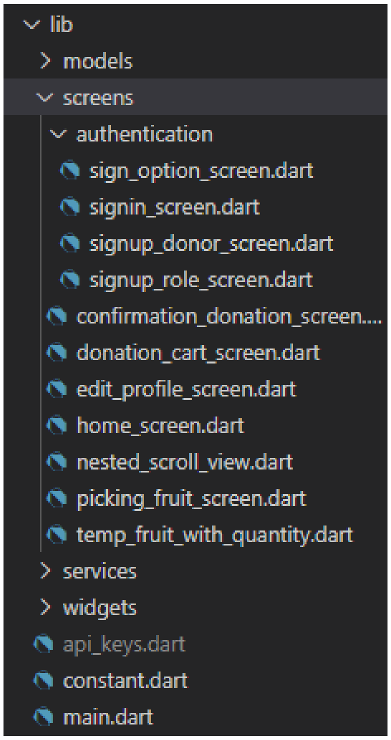


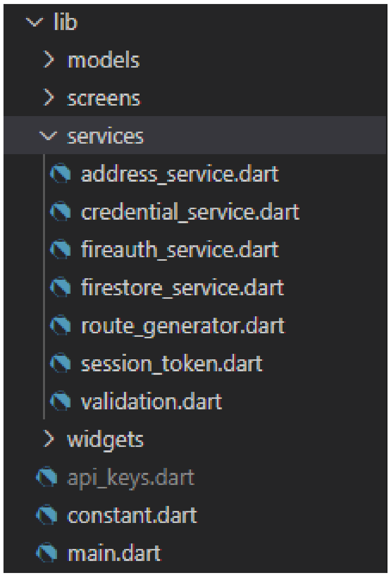
Figure 4.3.1: Overall structure design



```
lib
  > models
  > screens
  > services
  > widgets
  api_keys.dart
  constant.dart
  main.dart
```



```
lib
  > models
  ∨ screens
    ∨ authentication
        sign_option_screen.dart
        signin_screen.dart
        signup_donor_screen.dart
        signup_role_screen.dart
      confirmation_donation_screen....
      donation_cart_screen.dart
      edit_profile_screen.dart
      home_screen.dart
      nested_scroll_view.dart
      picking_fruit_screen.dart
      temp_fruit_with_quantity.dart
  > services
  > widgets
  api_keys.dart
  constant.dart
  main.dart
```



```
lib
  > models
  > screens
  ∨ services
      address_service.dart
      credential_service.dart
      fireauth_service.dart
      firestore_service.dart
      route_generator.dart
      session_token.dart
      validation.dart
  > widgets
  api_keys.dart
  constant.dart
  main.dart
```

Figure 4.3.2: File structure

Everything in Flutter is a widget, that makes our application relies heavily on both inheritance properties between widgets and Decorator Pattern Design. Because the finish product will look like a top-down tree (Figure 4.3.3), we place classes and models on the root to be able to access their properties and methods throughout the entire tree when we need a specific service from them. FireAuthService and FireStoreService are two wrapper classes that implement methods from Firebase Authentication class and Firebase FireStore class. The former class keeps the user authentication status through an instance inside it. The class

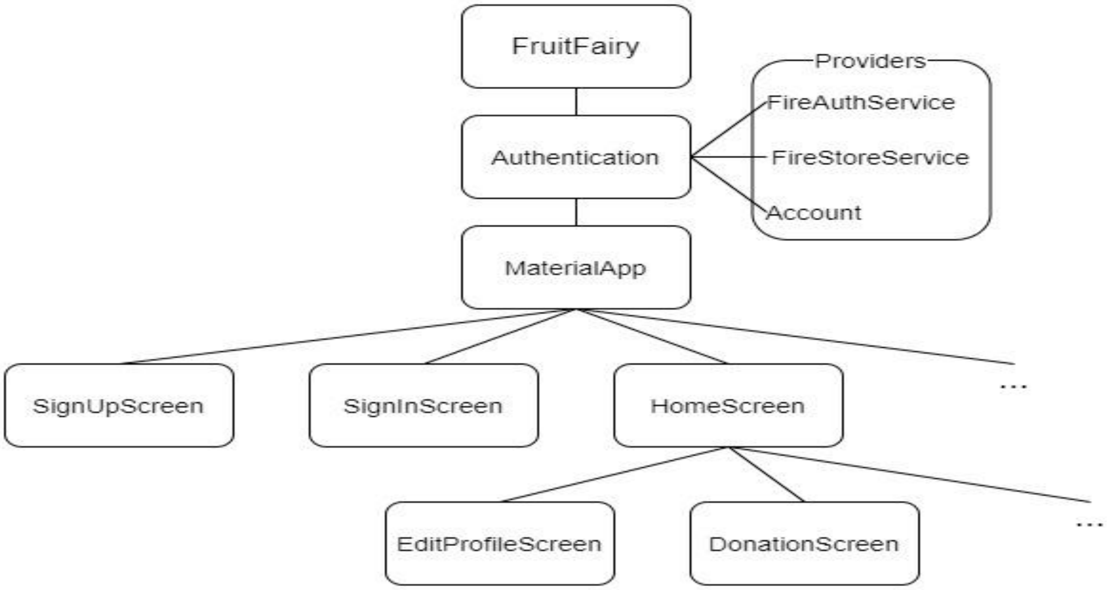
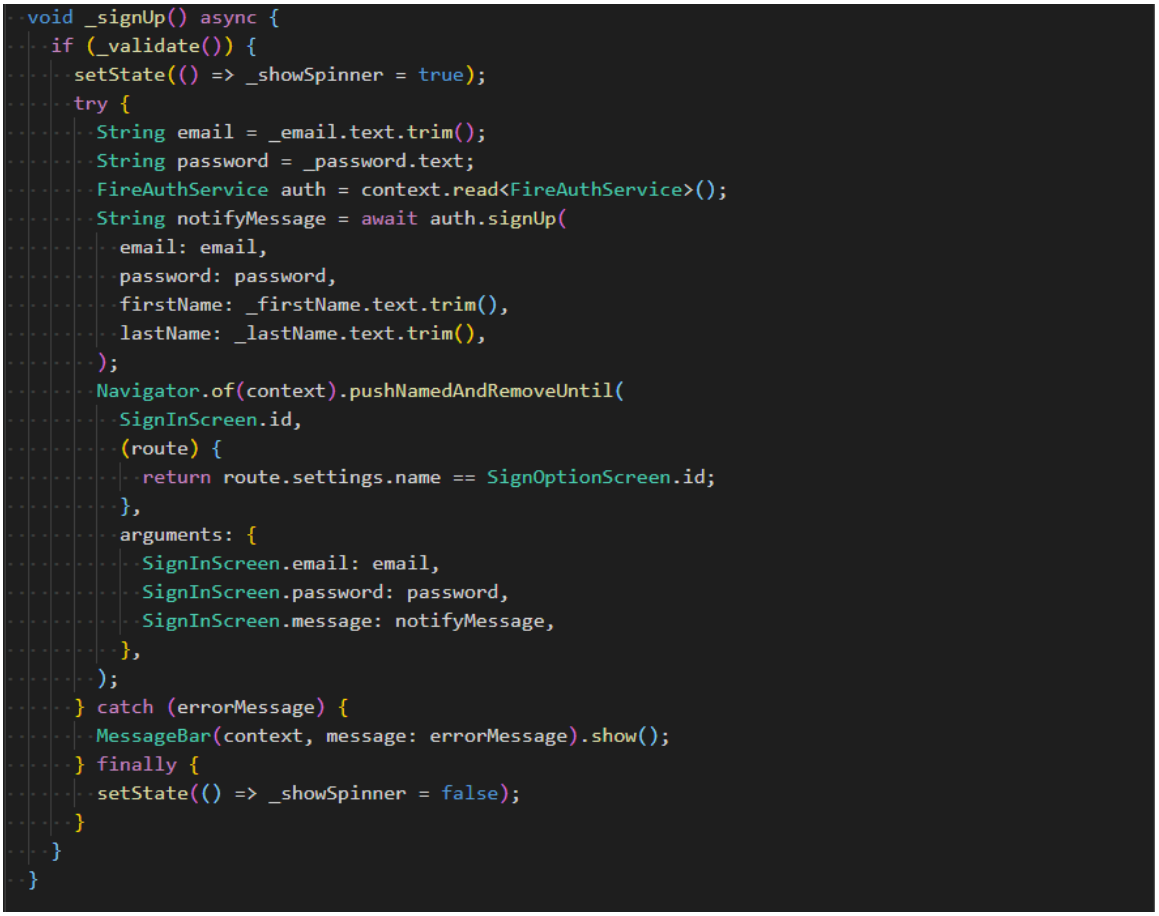


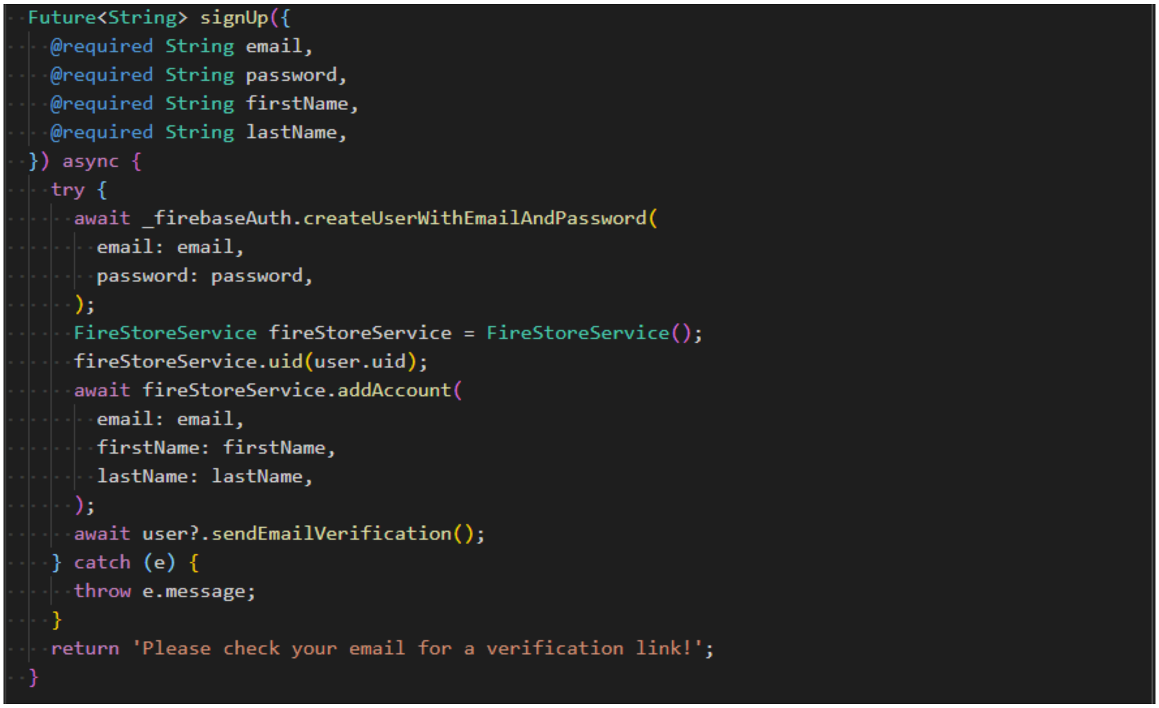
Figure 4.3.3: Fruit Fairy root widget tree

provides a sign-up method for when the user wants to create an account for the app as seen from figure 4.3.4 and figure 4.3.5.



```
void _signUp() async {
  if (_validate()) {
    setState(() => _showSpinner = true);
    try {
      String email = _email.text.trim();
      String password = _password.text;
      FireAuthService auth = context.read<FireAuthService>();
      String notifyMessage = await auth.signUp(
        email: email,
        password: password,
        firstName: _firstName.text.trim(),
        lastName: _lastName.text.trim(),
      );
      Navigator.of(context).pushNamedAndRemoveUntil(
        SignInScreen.id,
        (route) {
          return route.settings.name == SignOptionScreen.id;
        },
        arguments: {
          SignInScreen.email: email,
          SignInScreen.password: password,
          SignInScreen.message: notifyMessage,
        },
      );
    } catch (errorMessage) {
      MessageBar(context, message: errorMessage).show();
    } finally {
      setState(() => _showSpinner = false);
    }
  }
}
```

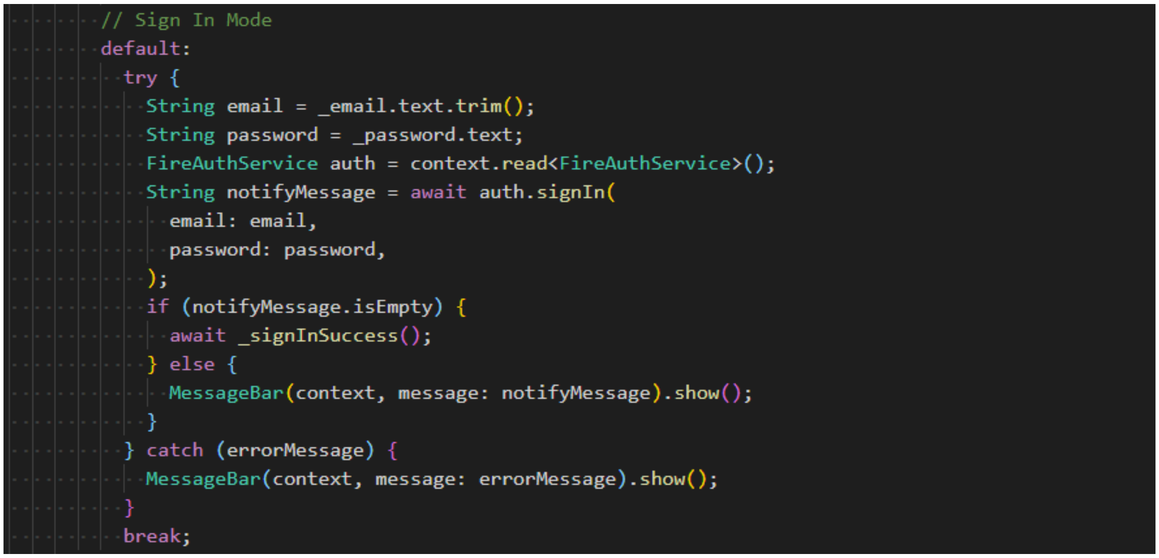
Figure 4.3.4: SignUp method called from a sign-up donor screen



```
Future<String> signUp({
  @required String email,
  @required String password,
  @required String firstName,
  @required String lastName,
}) async {
  try {
    await _firebaseAuth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    FireStoreService fireStoreService = FireStoreService();
    fireStoreService.uid(user.uid);
    await fireStoreService.addAccount(
      email: email,
      firstName: firstName,
      lastName: lastName,
    );
    await user?.sendEmailVerification();
  } catch (e) {
    throw e.message;
  }
  return 'Please check your email for a verification link!';
}
```

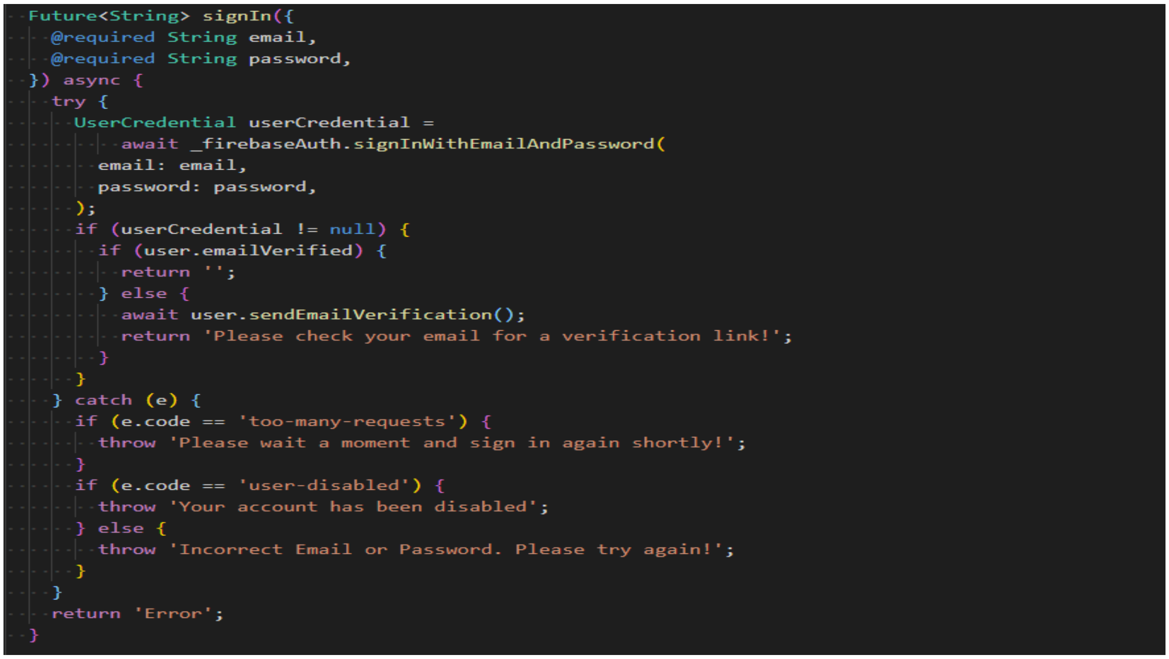
Figure 4.3.5: SignUp method definition inside FireAuthService class

FireAuthService can also perform sign in with email address and password as the user need as well as re-send email verification to registered user (Figure 4.3.6 and 4.3.7).



```
// Sign In Mode
default:
  try {
    String email = _email.text.trim();
    String password = _password.text;
    FireAuthService auth = context.read<FireAuthService>();
    String notifyMessage = await auth.signIn(
      email: email,
      password: password,
    );
    if (notifyMessage.isEmpty) {
      await _signInSuccess();
    } else {
      MessageBar(context, message: notifyMessage).show();
    }
  } catch (errorMessage) {
    MessageBar(context, message: errorMessage).show();
  }
  break;
```

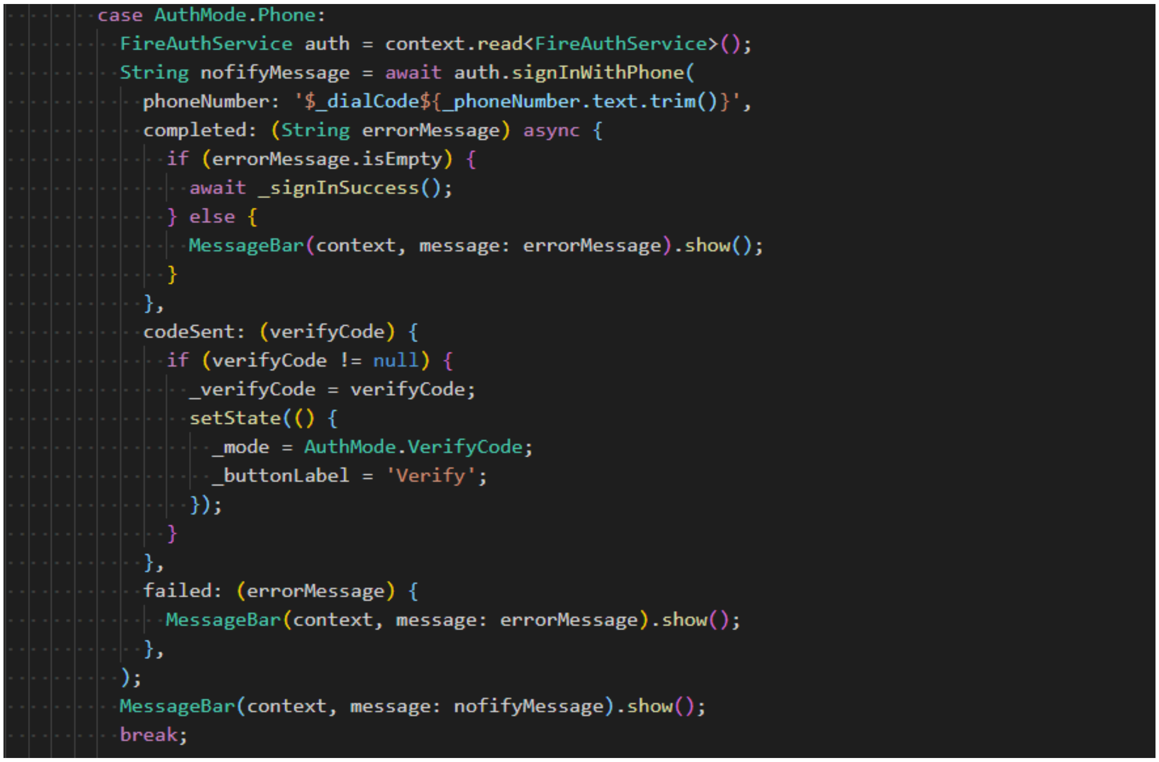
Figure 4.3.6: SignIn method called from sign in screen



```
Future<String> signIn({
  @required String email,
  @required String password,
}) async {
  try {
    UserCredential userCredential =
        await _firebaseAuth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    if (userCredential != null) {
      if (user.emailVerified) {
        return '';
      } else {
        await user.sendEmailVerification();
        return 'Please check your email for a verification link!';
      }
    }
  } catch (e) {
    if (e.code == 'too-many-requests') {
      throw 'Please wait a moment and sign in again shortly!';
    }
    if (e.code == 'user-disabled') {
      throw 'Your account has been disabled';
    } else {
      throw 'Incorrect Email or Password. Please try again!';
    }
  }
  return 'Error';
}
```

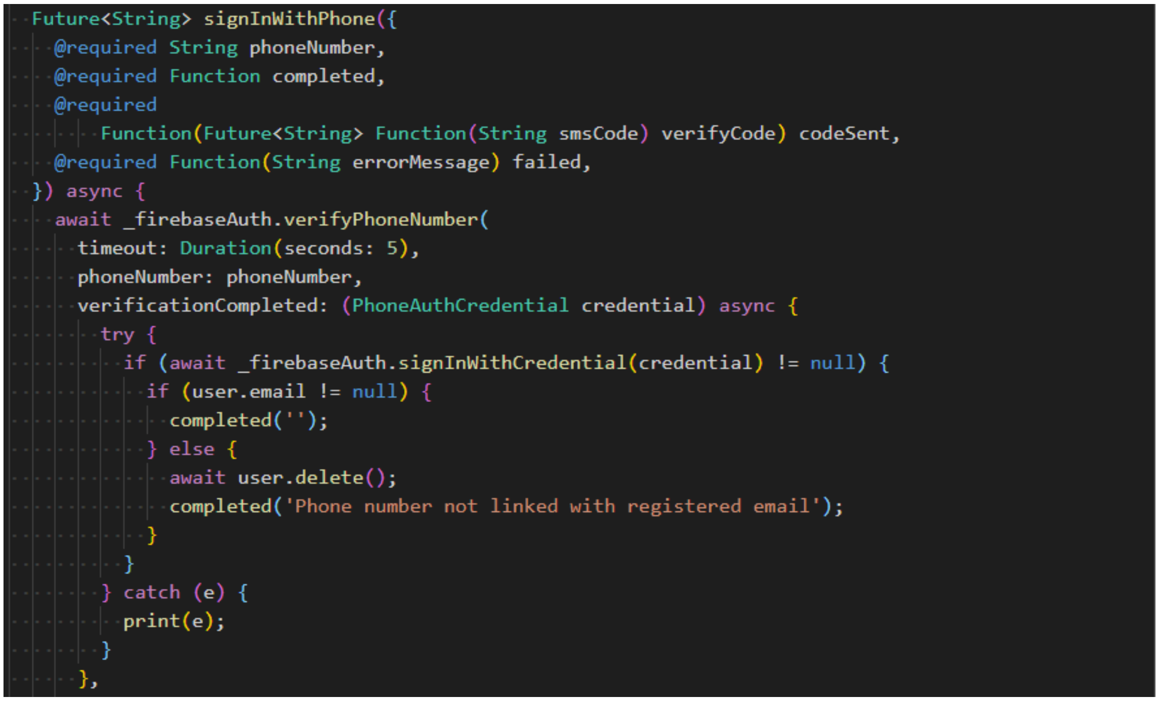
Figure 4.3.7: SignIn method definition inside FireAuthService class

Validating user’s email early upon sign up will reduce the possibility of fraud later as we need to route their donation to certain charities in the area. Fruit Fairy also provides sign in with phone number as an alternative way to authenticate user, who does not feel comfortable with typing password in public area. As a constraint, the user must already have an account and a registered phone number linked with our app through Profile section. When signing in, user will request a verification id and an SMS code through a phone number (Figure 4.3.8 and 4.3.9). After receiving the code, the user is required to enter it within a short amount of time to sign in. Figure 4.3.8, figure 4.3.10 and figure 4.3.11 illustrate how backend and frontend code are kept separated by passing and returning callback functions between different state of UI. When successfully signed in, the user unique identification number is retrieved from Firebase and assigned to the current FireStore instance to get additional data we stored to the user (Figure 4.3.12). Lastly, FireAuthService can sign a user out securely through a simple dropdown from touching the initial icon from home screen (Figure 4.3.13).



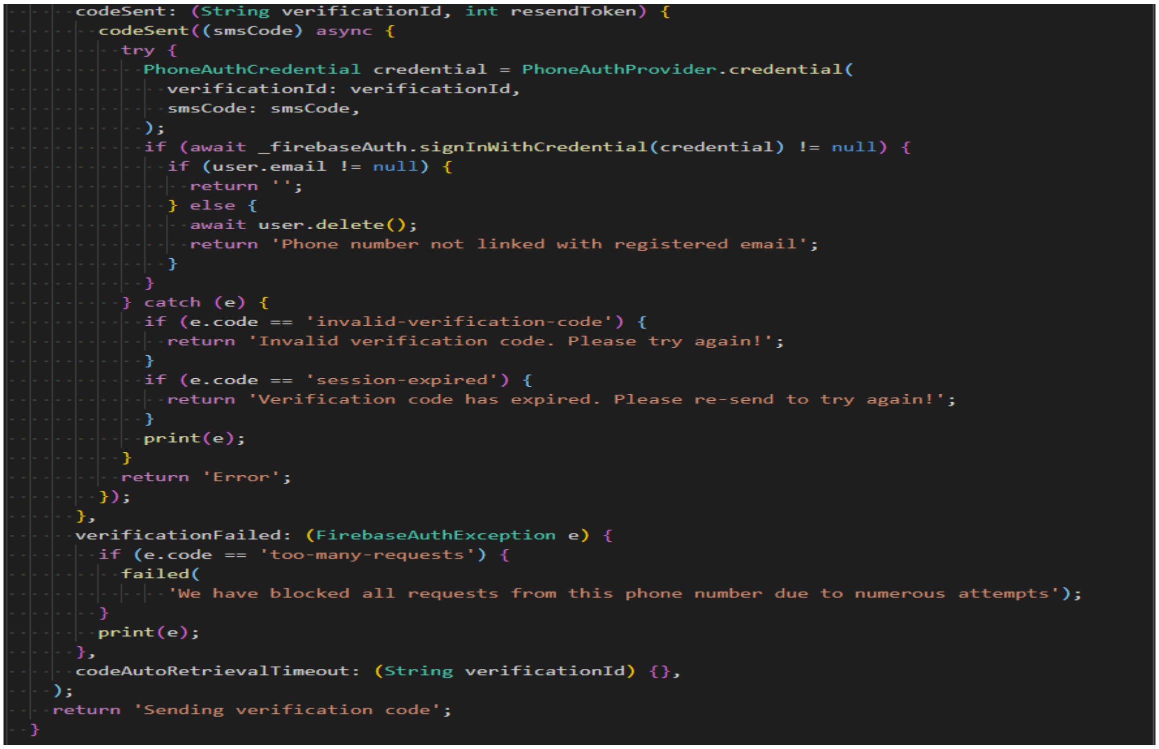
```
        case AuthMode.Phone:
          FireAuthService auth = context.read<FireAuthService>();
          String nofifyMessage = await auth.signInWithPhone(
            phoneNumber: '$_dialCode${_phoneNumber.text.trim()}',
            completed: (String errorMessage) async {
              if (errorMessage.isEmpty) {
                await _signInSuccess();
              } else {
                MessageBar(context, message: errorMessage).show();
              }
            },
            codeSent: (verifyCode) {
              if (verifyCode != null) {
                _verifyCode = verifyCode;
                setState(() {
                  _mode = AuthMode.VerifyCode;
                  _buttonLabel = 'Verify';
                });
              }
            },
            failed: (errorMessage) {
              MessageBar(context, message: errorMessage).show();
            },
          );
          MessageBar(context, message: nofifyMessage).show();
          break;
```

Figure 4.3.8: SignInWithPhone method called from sign in screen.



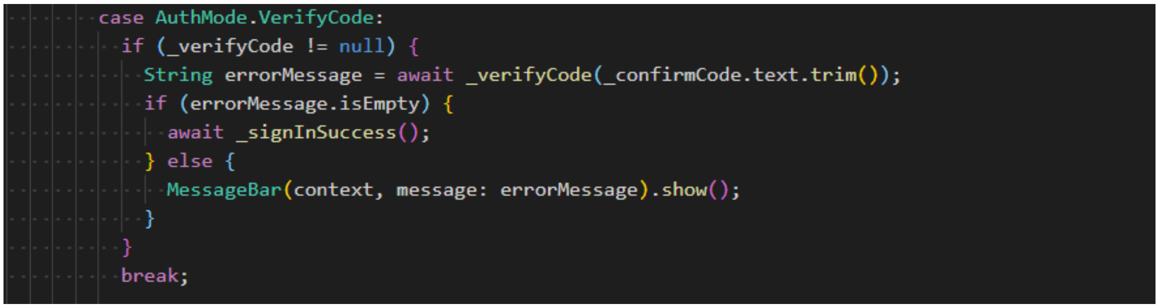
```
Future<String> signInWithPhone({
  @required String phoneNumber,
  @required Function completed,
  @required
      Function(Future<String> Function(String smsCode) verifyCode) codeSent,
  @required Function(String errorMessage) failed,
}) async {
  await _firebaseAuth.verifyPhoneNumber(
    timeout: Duration(seconds: 5),
    phoneNumber: phoneNumber,
    verificationCompleted: (PhoneAuthCredential credential) async {
      try {
        if (await _firebaseAuth.signInWithCredential(credential) != null) {
          if (user.email != null) {
            completed('');
          } else {
            await user.delete();
            completed('Phone number not linked with registered email');
          }
        }
      } catch (e) {
        print(e);
      }
    },
```

Figure 4.3.9: SignInWithPhone method definition inside FireAuthService class (Part 1)



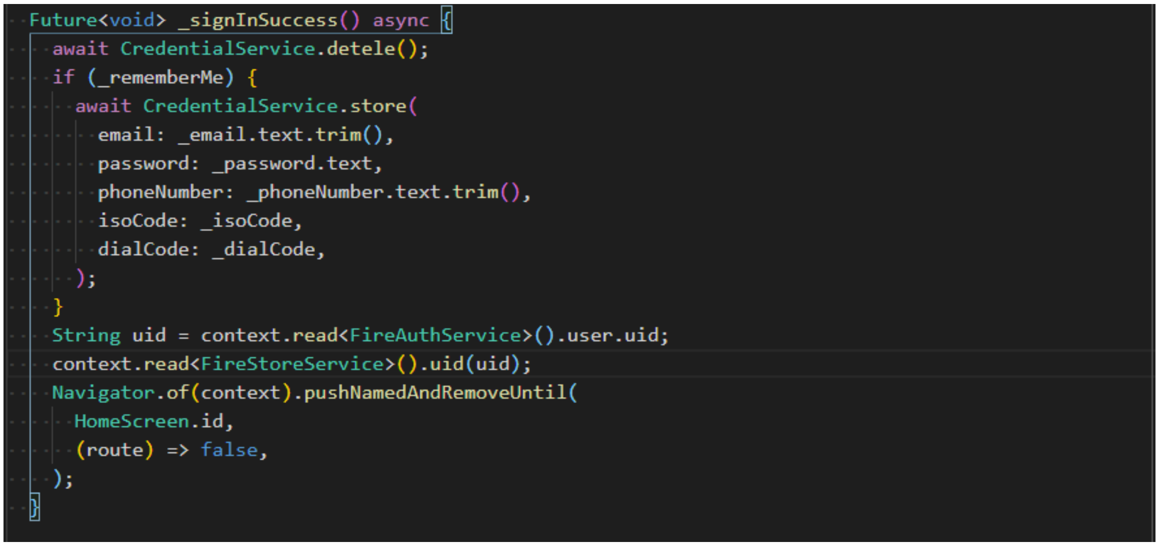
```
    codeSent: (String verificationId, int resendToken) {
      codeSent((smsCode) async {
        try {
          PhoneAuthCredential credential = PhoneAuthProvider.credential(
            verificationId: verificationId,
            smsCode: smsCode,
          );
          if (await _firebaseAuth.signInWithCredential(credential) != null) {
            if (user.email != null) {
              return '';
            } else {
              await user.delete();
              return 'Phone number not linked with registered email';
            }
          }
        } catch (e) {
          if (e.code == 'invalid-verification-code') {
            return 'Invalid verification code. Please try again!';
          }
          if (e.code == 'session-expired') {
            return 'Verification code has expired. Please re-send to try again!';
          }
          print(e);
        }
        return 'Error';
      });
    },
    verificationFailed: (FirebaseAuthException e) {
      if (e.code == 'too-many-requests') {
        failed(
            'We have blocked all requests from this phone number due to numerous attempts');
      }
      print(e);
    },
    codeAutoRetrievalTimeout: (String verificationId) {},
  );
  return 'Sending verification code';
}
```

Figure 4.3.10: SignInWithPhone method definition inside FireAuthService class (Part 2)



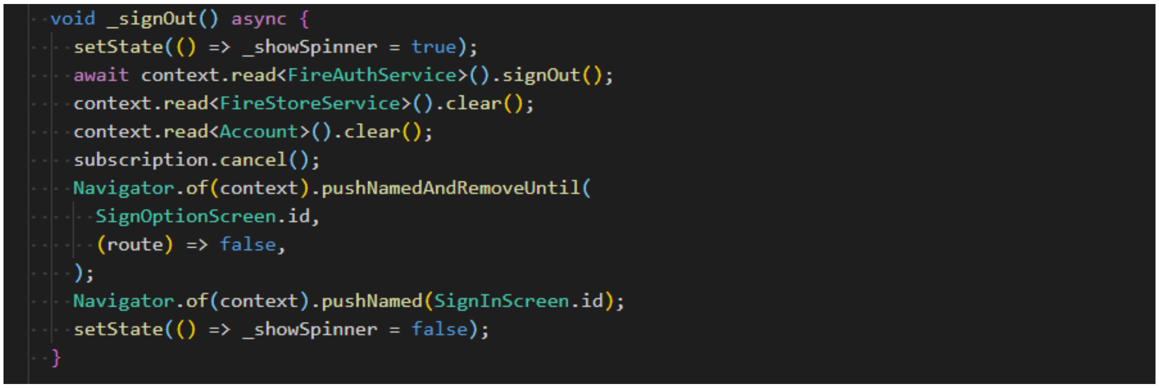
```
        case AuthMode.VerifyCode:
          if (_verifyCode != null) {
            String errorMessage = await _verifyCode(_confirmCode.text.trim());
            if (errorMessage.isEmpty) {
              await _signInSuccess();
            } else {
              MessageBar(context, message: errorMessage).show();
            }
          }
          break;
```

Figure 4.3.11: VerifyCode method is passed as a callback to receive update from user input to continue with authentication process.



```
Future<void> _signInSuccess() async {
  await CredentialService.detele();
  if (_rememberMe) {
    await CredentialService.store(
      email: _email.text.trim(),
      password: _password.text,
      phoneNumber: _phoneNumber.text.trim(),
      isoCode: _isoCode,
      dialCode: _dialCode,
    );
  }
  String uid = context.read<FireAuthService>().user.uid;
  context.read<FireStoreService>().uid(uid);
  Navigator.of(context).pushNamedAndRemoveUntil(
    HomeScreen.id,
    (route) => false,
  );
}
```

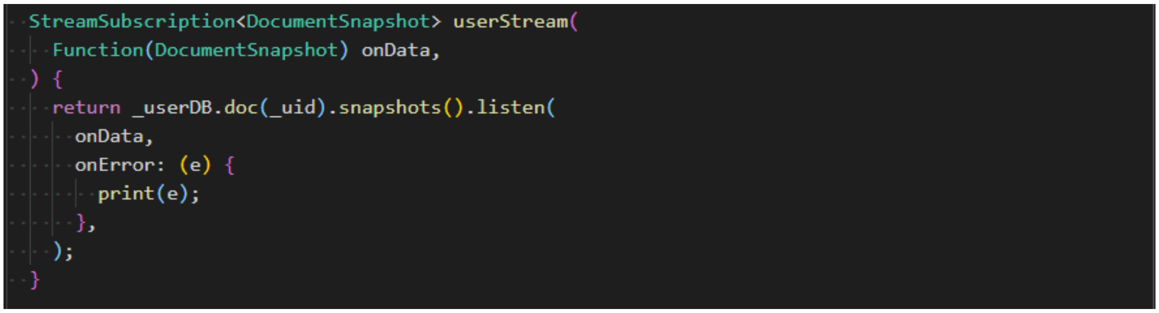
Figure 4.3.12: Retrieve user uid and transition to home screen



```
void _signOut() async {
  setState(() => _showSpinner = true);
  await context.read<FireAuthService>().signOut();
  context.read<FireStoreService>().clear();
  context.read<Account>().clear();
  subscription.cancel();
  Navigator.of(context).pushNamedAndRemoveUntil(
    SignOptionScreen.id,
    (route) => false,
  );
  Navigator.of(context).pushNamed(SignInScreen.id);
  setState(() => _showSpinner = false);
}
```

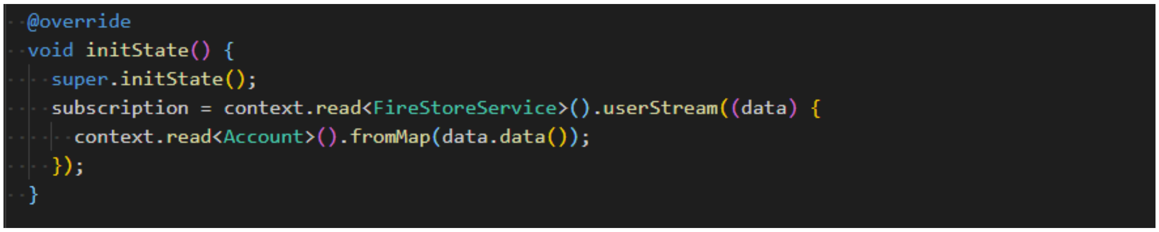
Figure 4.3.13: Sign out method call from home screen

An advantage of using a database like Cloud FireStore from Firebase is the flexibility of storing data in a non-tabular format of NoSQL database comparing to traditional relational database. Not only this reduced the amount of time in planning for us, FireStore also provided a real-time database to actively update user with changes from their account and this can be implemented easily as seen through figure 4.3.14 and 4.3.15. Upon signing in, the user device subscribes to a stream of data and listens for information changes through that account. Through this, the user can see what they update right away as well as receiving new statuses of their donations when a charity responds to the donation to without reloading the application.



```
StreamSubscription<DocumentSnapshot> userStream(
  Function(DocumentSnapshot) onData,
) {
  return _userDB.doc(_uid).snapshots().listen(
    onData,
    onError: (e) {
      print(e);
    },
  );
}
```

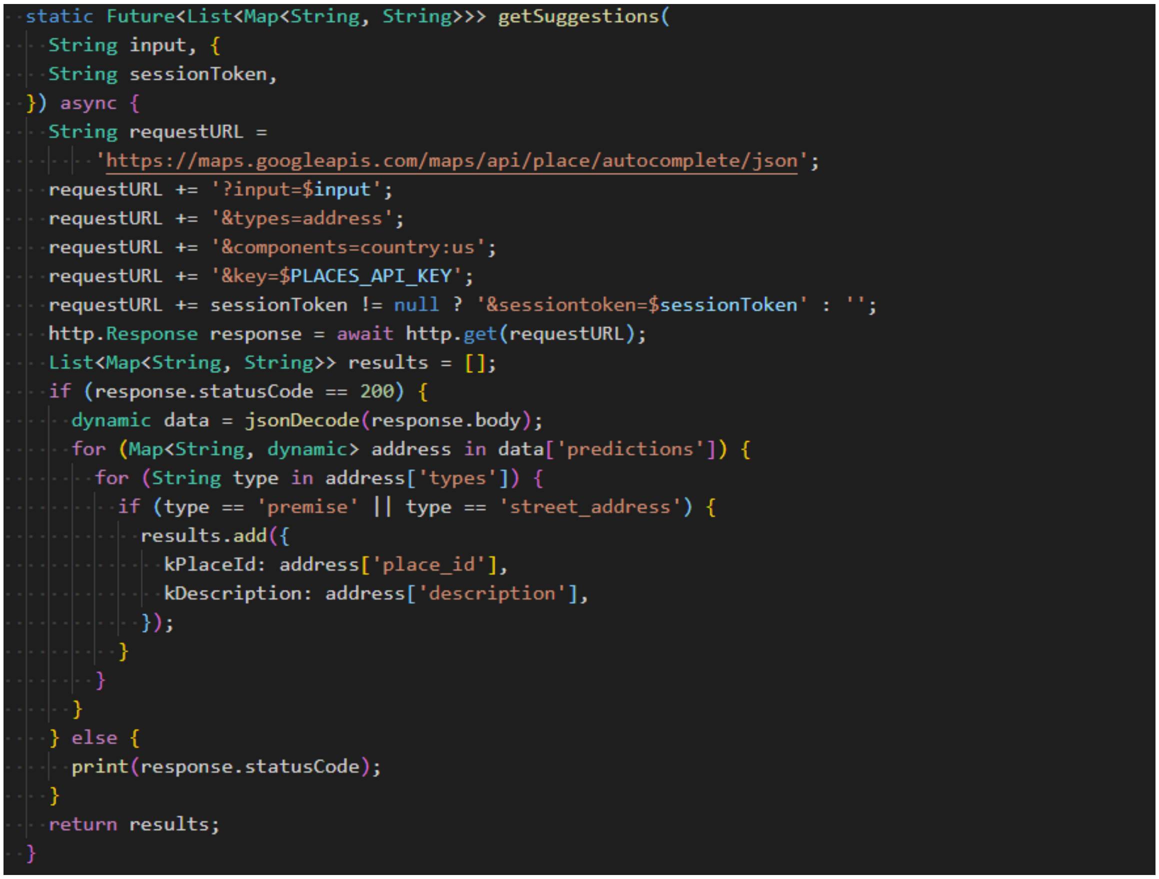
Figure 4.3.14: A method from FireStoreService class that let the caller subscribe and retrieve real-time data from a user.



```
@override
void initState() {
  super.initState();
  subscription = context.read<FireStoreService>().userStream((data) {
    context.read<Account>().fromMap(data.data());
  });
}
```

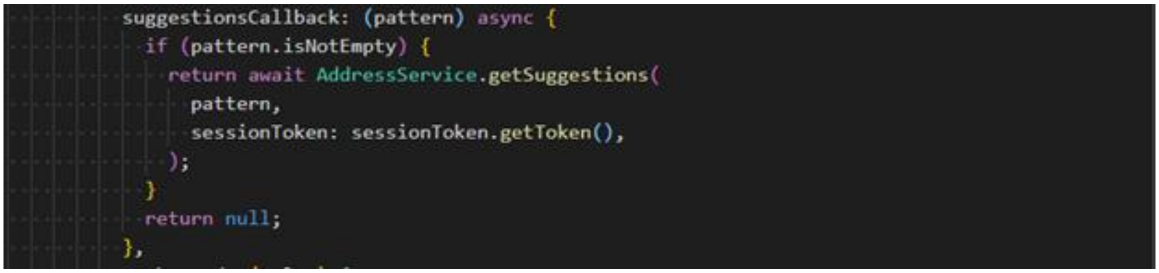
Figure 4.3.15: Subscribe to a stream upon sign in on home screen

Fruit Fairy is responsible for routing donations from registered users to the correct charity in the area while balancing the number of fruits or vegetables that each charity will receive; therefore, it is important for us to collect addresses from both ends to perform accurate calculation. On the first donation, the donor is required to provide their home address and to reduce the user's error from manually typing, we implemented Places APIs from Google Cloud platform to retrieve a complete valid address based on the first fill characters that the user entered. The process required two steps. First, we use the pattern from the user's input and make an API call to get recommended addresses, then display them as a dropdown list to the user as in figure 4.3.16 and figure 4.3.17. When the user selects a provided option, the app makes a second call to retrieve full information about the address (Figure 4.3.18 and 4.3.19). Because Google APIs will charge for every call to their service and recommend bundle multiple calls into one session to reduce the cost, we create a helper class, SessionToken, to generate a new session token and reuse it within a short amount of time then void it when the token becomes invalid (Figure 4.20).



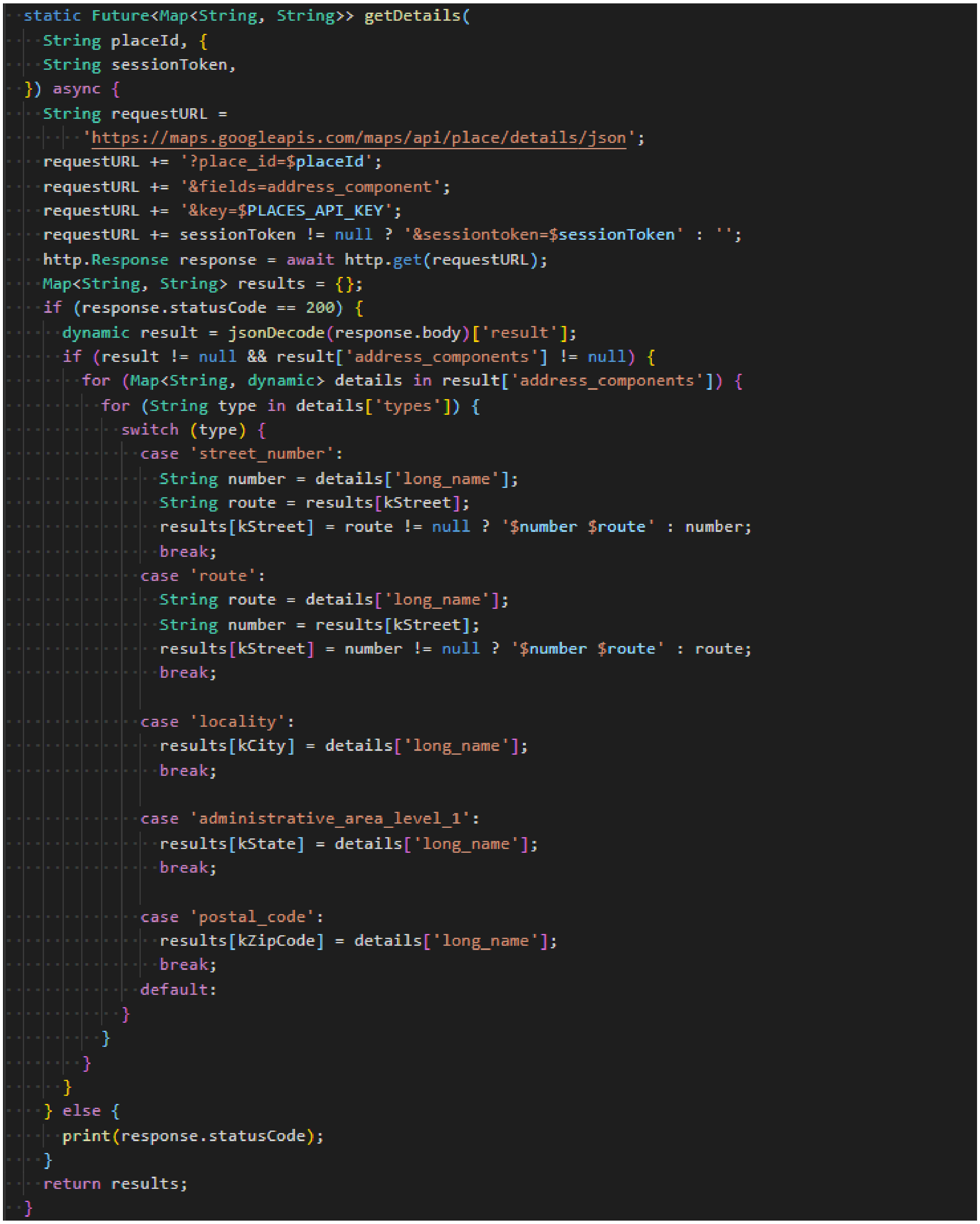
```
static Future<List<Map<String, String>>> getSuggestions(
  String input, {
  String sessionToken,
}) async {
  String requestURL =
      'https://maps.googleapis.com/maps/api/place/autocomplete/json';
  requestURL += '?input=$input';
  requestURL += '&types=address';
  requestURL += '&components=country:us';
  requestURL += '&key=$PLACES_API_KEY';
  requestURL += sessionToken != null ? '&sessiontoken=$sessionToken' : '';
  http.Response response = await http.get(requestURL);
  List<Map<String, String>> results = [];
  if (response.statusCode == 200) {
    dynamic data = jsonDecode(response.body);
    for (Map<String, dynamic> address in data['predictions']) {
      for (String type in address['types']) {
        if (type == 'premise' || type == 'street_address') {
          results.add({
            kPlaceId: address['place_id'],
            kDescription: address['description'],
          });
        }
      }
    }
  } else {
    print(response.statusCode);
  }
  return results;
}
```

Figure 4.3.16: Address suggestion method definition inside AddressService class



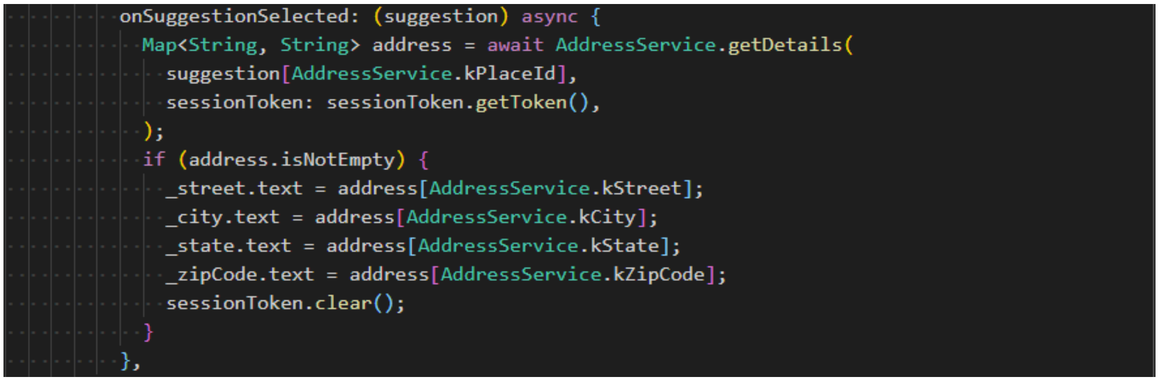
```
suggestionsCallback: (pattern) async {
  if (pattern.isNotEmpty) {
    return await AddressService.getSuggestions(
      pattern,
      sessionToken: sessionToken.getToken(),
    );
  }
  return null;
},
```

Figure 4.3.17: Take user input from a screen and perform API call for address suggestion



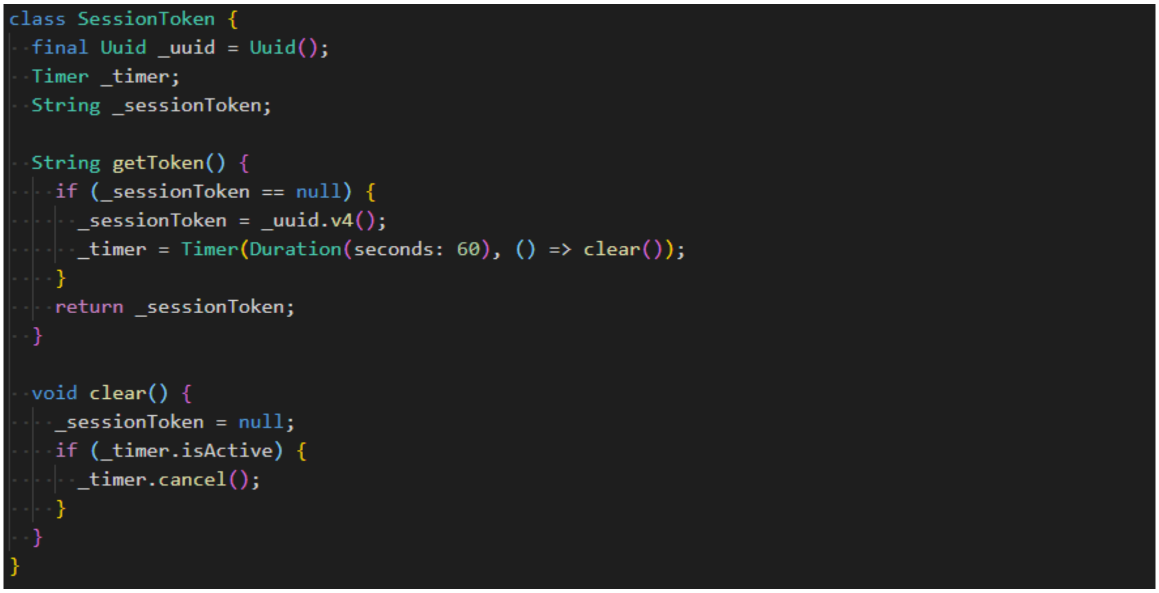
```
static Future<Map<String, String>> getDetails(
  String placeId, {
  String sessionToken,
}) async {
  String requestURL =
      'https://maps.googleapis.com/maps/api/place/details/json';
  requestURL += '?place_id=$placeId';
  requestURL += '&fields=address_component';
  requestURL += '&key=$PLACES_API_KEY';
  requestURL += sessionToken != null ? '&sessiontoken=$sessionToken' : '';
  http.Response response = await http.get(requestURL);
  Map<String, String> results = {};
  if (response.statusCode == 200) {
    dynamic result = jsonDecode(response.body)['result'];
    if (result != null && result['address_components'] != null) {
      for (Map<String, dynamic> details in result['address_components']) {
        for (String type in details['types']) {
          switch (type) {
            case 'street_number':
              String number = details['long_name'];
              String route = results[kStreet];
              results[kStreet] = route != null ? '$number $route' : number;
              break;
            case 'route':
              String route = details['long_name'];
              String number = results[kStreet];
              results[kStreet] = number != null ? '$number $route' : route;
              break;

            case 'locality':
              results[kCity] = details['long_name'];
              break;

            case 'administrative_area_level_1':
              results[kState] = details['long_name'];
              break;

            case 'postal_code':
              results[kZipCode] = details['long_name'];
              break;
            default:
          }
        }
      }
    }
  } else {
    print(response.statusCode);
  }
  return results;
}
```

Figure 4.3.18: Address details method definition inside AddressService class



```
          onSuggestionSelected: (suggestion) async {
            Map<String, String> address = await AddressService.getDetails(
              suggestion[AddressService.kPlaceId],
              sessionToken: sessionToken.getToken(),
            );
            if (address.isNotEmpty) {
              _street.text = address[AddressService.kStreet];
              _city.text = address[AddressService.kCity];
              _state.text = address[AddressService.kState];
              _zipCode.text = address[AddressService.kZipCode];
              sessionToken.clear();
            }
          },
```

Figure 4.3.19: Perform API call to retrieve more details about an address upon user selection from a screen



```
class SessionToken {
  final Uuid _uuid = Uuid();
  Timer _timer;
  String _sessionToken;

  String getToken() {
    if (_sessionToken == null) {
      _sessionToken = _uuid.v4();
      _timer = Timer(Duration(seconds: 60), () => clear());
    }
    return _sessionToken;
  }

  void clear() {
    _sessionToken = null;
    if (_timer.isActive) {
      _timer.cancel();
    }
  }
}
```

Figure 4.3.20: Helper class that generates and manages session token

4.4 Design Constraints, Problems, Trade-offs, and Solutions

4.4.1 Design Constraints and Challenges

The application involved multiple aspects such as creating a user-friendly interface, storing user information, and storing the business data of our users to ensure the applications functionality.

One of the main concerns of the application was to make it user friendly for seniors. Seniors are a population that will probably have excess fruits and vegetables in their garden that would be willing to donate and are a good candidate for using the application. The current application is only implemented on mobile devices. The mobile application was designed clearly enough for seniors to use it efficiently on a smaller screen.

An additional design constraint was using 3rd party software in creating our application. This will not allow us to fully customize our application and will need to follow predetermined settings such as using Cloud Firestore

4.4.2 Design Solutions and Trade-offs

To solve the issues above, we decided on choosing the Flutter framework and the Cloud Firestore database. The Flutter framework was used to create a large UI design that will make the functionality of the application clear. Another solution was to add a user manual that helps explain the process of donating. The trade-off in this case was that more work will be needed on the development side to learn the framework which we believed will guarantee high usage of our application.

The trade-off to using Cloud Firestore is immense because it has a high compatibility with the Flutter framework and handles the storing our data. Even though we were not able to fully customize Cloud Firestore, it was able to handle most of our needs.

Chapter 5 System Implementation

Thuan Chau

5.1 Implementation Overview

Fruit Fairy targets only mobile users and with the Flutter framework, we were able to cover both Android and iOS users, which are the two largest domains on mobile now. Because we relied on the structure design from Flutter where Dart is used; therefore, it is also our main programming language for this entire project. To use the application, the user is required to have a mobile phone that operates on Android 6.0 or iOS 8 and above. Fruit Fairy should be available to install on both Google Play Store and Apple App Store if everything goes according to our plan. On the backend side, our project relied on the Firebase framework, where predefined packages can be used to authenticate users as well as store and load their data from both the database and storage to their devices. Fruit Fairy also uses Google Maps API to help us distribute donations evenly to every registered charity in the area.

5.2 Implementation of Developed Solutions

The application used the MVC(model-view-controller) as the design pattern. Model represents the data of every individual user and their donations on the app. The view displays the data under the UI design in the most understandable way, which reduces learning time and helps the user make quicker decisions. The controller enables the user to interact with the application including moving data from one screen to another as well as update it correctly with the model and the database. The application is designed fully with user authentication. After signing up, the user receives an email with a verification link to finish the signup process. When this user login the application, it will keep this user's sign in state till he decides to logout the app. The application provides multiple ways to authenticate the user such as the combination of email and password or phone number with a verification code sent through SMS message. Both login methods are handled correctly with multiple testing cycles throughout the development.

5.3 Implementation Problems, Challenges, and Lesson Learned

When implementing the project, we faced more problems than expected, and fortunately, we tackled each problem one by one throughout the development time. For example, we planned to do the alternate verification method by using the phone number when the user wants to add his phone number in the profile and use it to login his account later. We also had problems doing unit testing to fulfil the system requirement of safety and accessibility. Furthermore, the application considers the data flow among all the different screens, and the flow of authentication. Our team also figured out how to utilize packages from Firebase on our backend while spending the same amount of time to use packages from Flutter. Despite all the difficulties we faced, we found the solutions to solve all these problems and complete the implementation. We read through the flutter development documentation to find all the useful information we needed. Additionally, the development team completed lots of research and watched multiple tutorial videos to find the most effective way to implement the code correctly and functions for our project. Lastly, we implemented predefined packages from Flutter to design a cleaner application UI. We learned how to do user authentication with email login and phone number sign-in as well as transferring real-time data from Firestore to the user's devices.

Chapter 6 Tools and Standards Yonatan Greenblum/George Michael Cuevas

6.1. Tools Used

Multiple tools were used in developing Fruit Fairy and are divided up into separate categories. The reason for choosing most of the tools mentioned below are because of positive experience from previous work.

Code development:

a. Android Studio/VS code – IDE's (Integrated Development Environments) that are familiar with the teams past coding experience. The selection of Android Studio for most of the team is because the online flutter course that the team learned was taught using Android Studio. These IDEs were used throughout the entire creation of Fruit Fairy.

b. GitHub – Online platform to host the project codebase. GitHub was used because the team members have previous experience using it. GitHub was used because it allows for multiple members to code different parts of the project independently. Each member can copy code from other members seamlessly into their own codebase through GitHub.

c. GitLive – Allowed live code editing between members. This method was used when members worked on the code together. The selection of GitLive was because it works for both the IDEs our team members used.

UI creation:

a. AdobeXD - UI/UX Design Tool. AdobeXD is fast and powerful for designing mockups and prototypes. It was used to create all the mockups for the application.

b. Procreate - Digital Graphics Illustration Tool. Procreate is a great tool for drawing and creating graphic images on iPad. It was used to design the fruit illustrations for our application. It was decided to use Procreate to design the images ourselves because we did not have to worry about copyrighting images.

c. Google Drive – Allows to share large and small files between members. The site mockup pages that were created were uploaded to the teams shared google drive for future reference when coding the user interface. The reason for choosing Google Drive is that all members already have Google Drive available.

Communication:

a. Slack – Online messaging tool between members. Slack allows for communication in separate channels. A private channel was created for developing the code. A separate channel used for communicating with the project advisor and for private chat between members. Slack was used throughout the entire project. The reason for choosing Slack was because of past positive experience of using it in a team-based project.

b. Zoom – Voice and video conference tool. The reason for choosing Zoom is because all members have previous use of Zoom and it is the main video tool used by SJSU. Zoom was used throughout the entirety of the project. The share screen feature was used as well for creating mockup pages for the site, discussing code development, and code documentation.

6.2. Standards

Fruit Fairy, being a mobile application, has multiple standards involving UI and security. The UI standards used in the application are based off the human interface guidelines constructed by Apple.

UI:

The UI (User Interface) was created with a focus on the users of the application. The application follows the concepts of clarity, deference, and depth. Clarity meaning that the font sizes of the text and style of the text is readable throughout the entire application and elevating on screen buttons to guide the user to the next action, screen clutter is non-existent. Deference – Movement between screens is crisp and the content normally fills the entire screen. Minimizing the number of screens needed. Depth – facilitate understanding of the user. User actions change the appearance of the screen and his options.

The design principles implemented in the application are aesthetic integrity, consistency, direct manipulation, feedback, and user control.

Aesthetic Integrity – Fruit Fairy is a procedural application so navigating through the application is a step-by-step process with predictable features and buttons.

Consistency - Fruit Fairy screen themes are identical with the app bar being a dark red color and the main screen being a light shade of pink. The color of the buttons and the text throughout the application are the same and all the text is bolded. Usage of known icons such as the "user profile" icon to access profile options and the question mark icon for help.

Direct Manipulation – User actions lead to the next screen. Alerts appear for the user if the user clicks on the question mark icon. Rotating the device changes the screen to fit accordingly.

Feedback – Clickable items are highlighted/make a sound for the user. Hitting the next button on the screens makes the default sound for hitting a button. Selecting fruit in the screens highlights them as marked.

User-control – The user is in control of the application and not vice-versa. At the same time, if a user is attempting an action that is not useful or undermining the application's purpose, the application will move back on itself to a previous screen. For example, if a user attempts to donate zero fruits or vegetables the application will redirect the user to the selecting fruit screen.

Security:

Fruit Fairy is a mobile app that involves user login from possible donors and charities. Security features on the user side exist by allowing the user to hide their password input while creating a new account. When creating a new user, the account will only be verified and accessible after email verification by the user. A user is also able to login to their account using their phone number which was verified in a previous step.

The user credentials for signing-up and logging in are not stored server side but are saved locally on the user's device.

Chapter 7 Testing and Experiment Yonatan Greenblum/Xuxiang Huang/Thuan Chau

7.1 Testing and Experiment Scope

Testing the functionality of Fruit Fairy was conducted at the creation of each major function in the development of the application. The major functions include the following: Donor Sign Up, Charity Sign Up, Login (using email/password, phone), Edit Profile, Donate, Wish List, and Logout.

The testing process included Black Box testing – inputting values and matching them with expected outputs, White Box testing – code review and ensuring correct transitioning between states, including branch testing. The testing also included performance testing of the application related to how long does the user need to wait to sign up/login and how long does it take for the application to download fruit images from the database. Another test process conducted was user scenario testing which encompasses testing multiple areas of the application in one sequence.

7.2 Testing and Experiment Approach

Performance testing –

a. Login time for a user:

   i. First time logging in: 856 milliseconds.

   ii. Login time: 621 milliseconds.

b. Loading fruit images:

i. First time loading images: 2217 milliseconds. (9 total fruit images)

ii. Loading images: Between 1484 – 1170 milliseconds.

c. Charity Suggestion Algorithm:

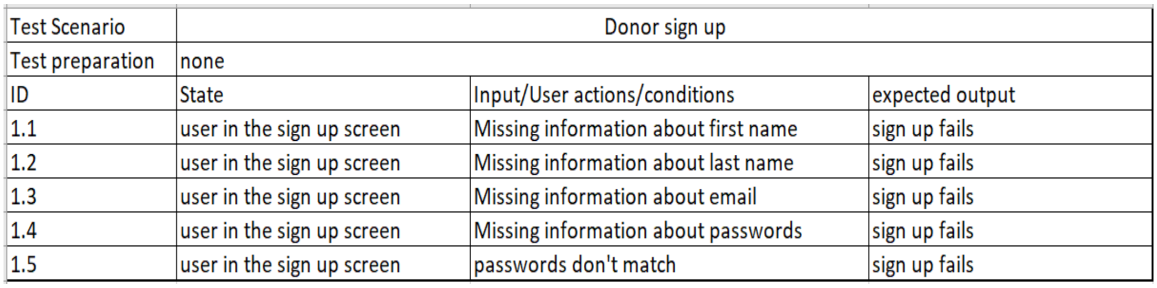
i. 5 Charities in the system: On average 1185 milliseconds. (The longest amount of time was 1248 milliseconds).

ii. 10 Charities in the system: On average 1484 milliseconds. (The longest amount of time was 1612 milliseconds).

iii. 15 Charities in the system: On average 2258 milliseconds. (The longest amount of time was 2510 milliseconds).

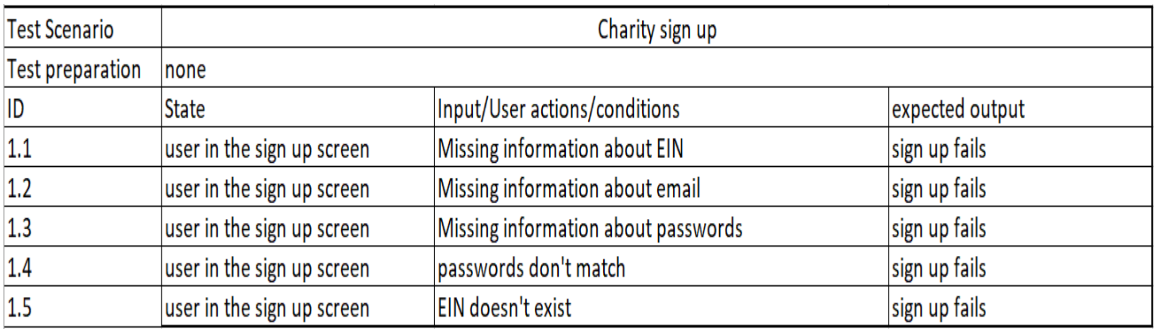
iv. 20 Charities in the system: On average 2786 milliseconds. (The longest amount of time was 3152 milliseconds).

User scenario testing –



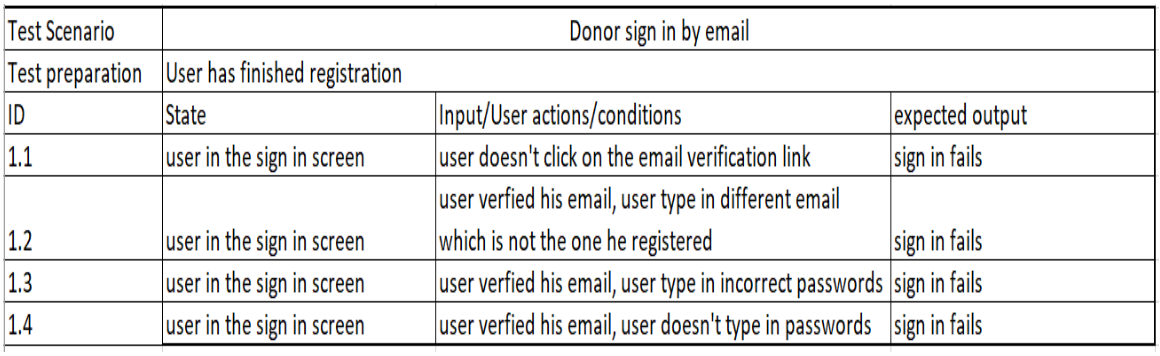
Test Scenario	Donor sign up		
Test preparation	none		
ID	State	Input/User actions/conditions	expected output
1.1	user in the sign up screen	Missing information about first name	sign up fails
1.2	user in the sign up screen	Missing information about last name	sign up fails
1.3	user in the sign up screen	Missing information about email	sign up fails
1.4	user in the sign up screen	Missing information about passwords	sign up fails
1.5	user in the sign up screen	passwords don't match	sign up fails

Table 7.2.1 Donor Sign Up Scenario Testing



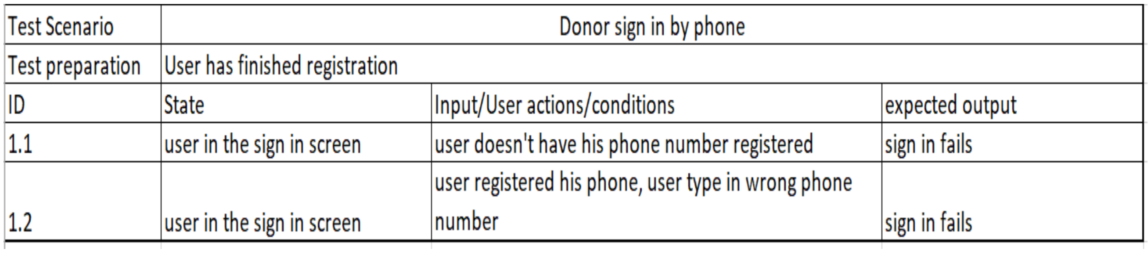
Test Scenario	Charity sign up		
Test preparation	none		
ID	State	Input/User actions/conditions	expected output
1.1	user in the sign up screen	Missing information about EIN	sign up fails
1.2	user in the sign up screen	Missing information about email	sign up fails
1.3	user in the sign up screen	Missing information about passwords	sign up fails
1.4	user in the sign up screen	passwords don't match	sign up fails
1.5	user in the sign up screen	EIN doesn't exist	sign up fails

Table 7.2.2 Charity Sign Up Scenario Testing



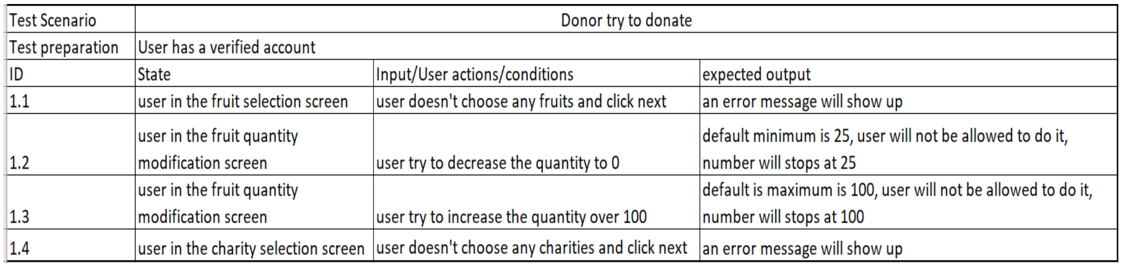
Test Scenario	Donor sign in by email		
Test preparation	User has finished registration		
ID	State	Input/User actions/conditions	expected output
1.1	user in the sign in screen	user doesn't click on the email verification link	sign in fails
1.2	user in the sign in screen	user verfied his email, user type in different email which is not the one he registered	sign in fails
1.3	user in the sign in screen	user verfied his email, user type in incorrect passwords	sign in fails
1.4	user in the sign in screen	user verfied his email, user doesn't type in passwords	sign in fails

Table 7.2.3 Donor Sign in by Email Scenario Testing

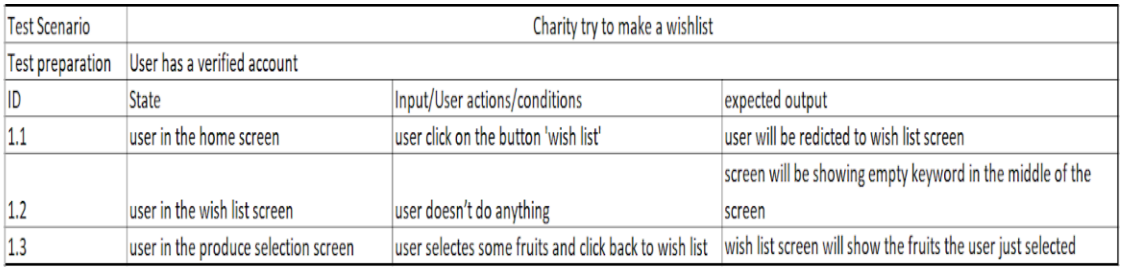


Test Scenario	Donor sign in by phone		
Test preparation	User has finished registration		
ID	State	Input/User actions/conditions	expected output
1.1	user in the sign in screen	user doesn't have his phone number registered	sign in fails
1.2	user in the sign in screen	user registered his phone, user type in wrong phone number	sign in fails

Table 7.2.4 Donor Sign in by Phone Scenario Testing



Test Scenario	Donor try to donate		
Test preparation	User has a verified account		
ID	State	Input/User actions/conditions	expected output
1.1	user in the fruit selection screen	user doesn't choose any fruits and click next	an error message will show up
1.2	user in the fruit quantity modification screen	user try to decrease the quantity to 0	default minimum is 25, user will not be allowed to do it, number will stops at 25
1.3	user in the fruit quantity modification screen	user try to increase the quantity over 100	default is maximum is 100, user will not be allowed to do it, number will stops at 100
1.4	user in the charity selection screen	user doesn't choose any charities and click next	an error message will show up

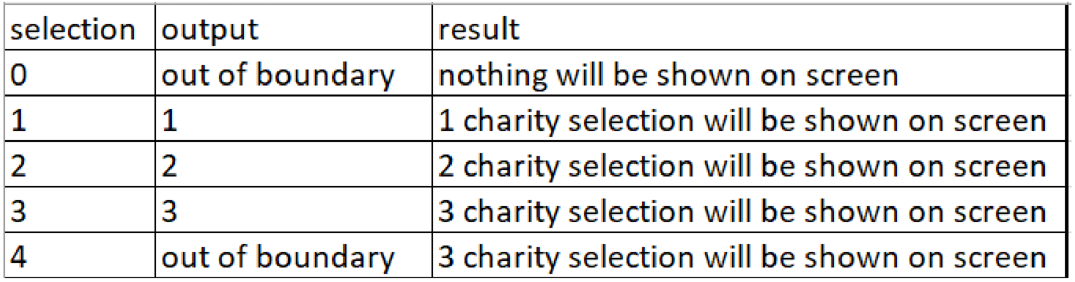


Test Scenario	Charity try to make a wishlist		
Test preparation	User has a verified account		
ID	State	Input/User actions/conditions	expected output
1.1	user in the home screen	user click on the button 'wish list'	user will be redicted to wish list screen
1.2	user in the wish list screen	user doesn't do anything	screen will be showing empty keyword in the middle of the screen
1.3	user in the produce selection screen	user selectes some fruits and click back to wish list	wish list screen will show the fruits the user just selected

Table 7.2.5 Charity Wishlist Scenario Testing

Black Box Testing -

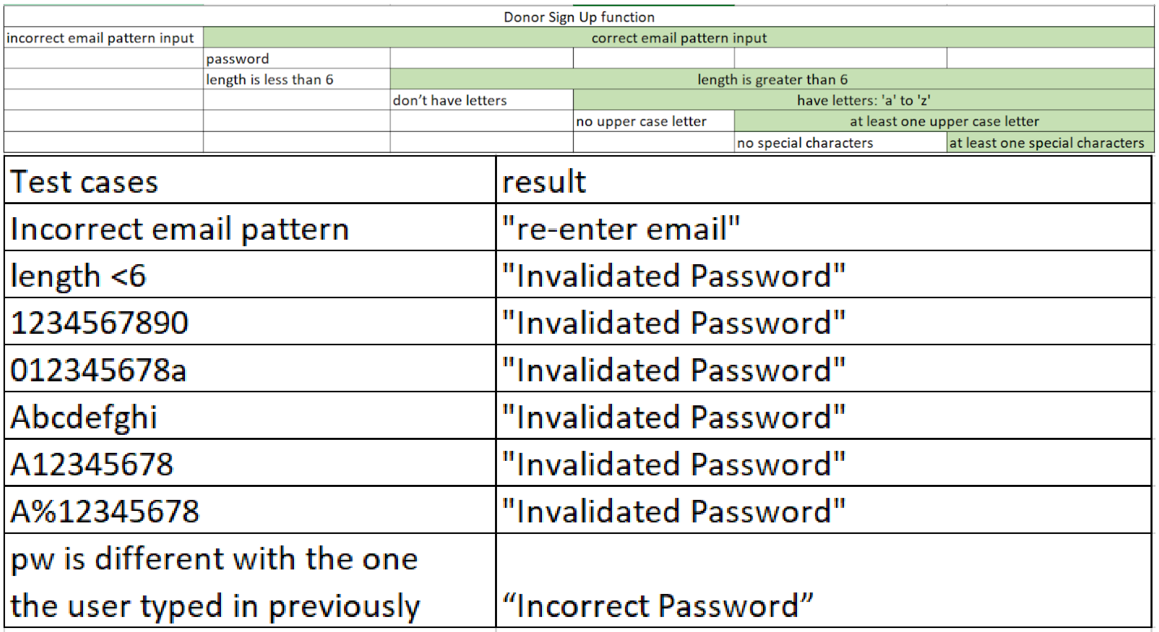
a. Boundary value testing
Charity Selection Choosing : [1,3]
test values: 0, 1, 2, 3, 4



selection	output	result
0	out of boundary	nothing will be shown on screen
1	1	1 charity selection will be shown on screen
2	2	2 charity selection will be shown on screen
3	3	3 charity selection will be shown on screen
4	out of boundary	3 charity selection will be shown on screen

Table 7.2.6 – Charity Selection Boundary Value Testing

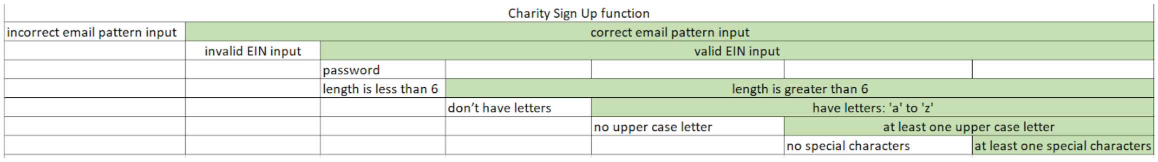
b. Equivalence Partitioning Testing



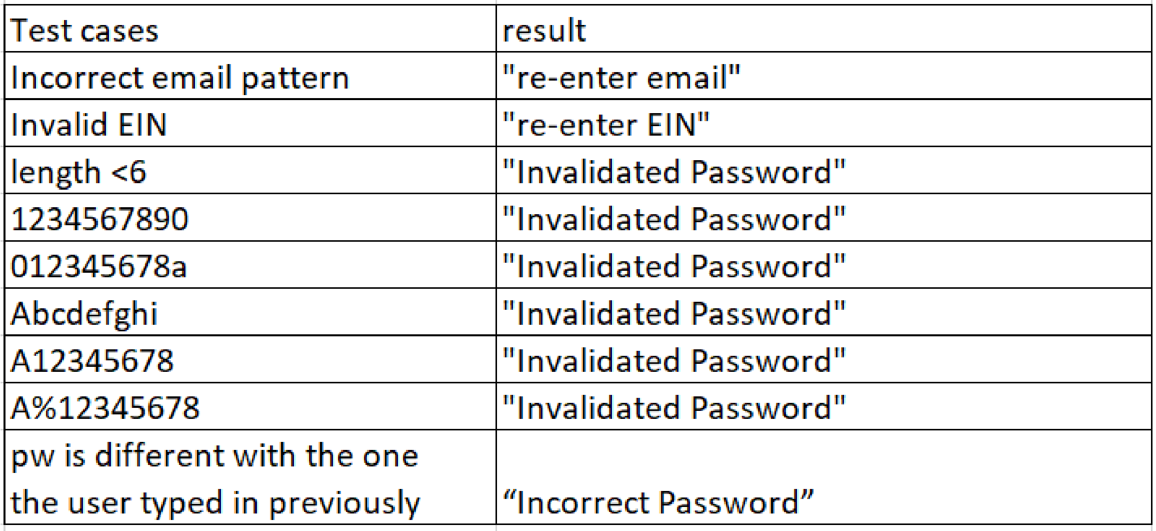
Donor Sign Up function					
incorrect email pattern input	correct email pattern input				
	password				
	length is less than 6	length is greater than 6			
		don't have letters	have letters: 'a' to 'z'		
			no upper case letter	at least one upper case letter	
				no special characters	at least one special characters

Test cases	result
Incorrect email pattern	"re-enter email"
length <6	"Invalidated Password"
1234567890	"Invalidated Password"
012345678a	"Invalidated Password"
Abcdefghi	"Invalidated Password"
A12345678	"Invalidated Password"
A%12345678	"Invalidated Password"
pw is different with the one the user typed in previously	"Incorrect Password"

Table 7.2.7 – Donor Sign Up Equivalence Partitioning Testing



Charity Sign Up function						
incorrect email pattern input	correct email pattern input					
	invalid EIN input	valid EIN input				
		password				
		length is less than 6	length is greater than 6			
			don’t have letters	have letters: 'a' to 'z'		
				no upper case letter	at least one upper case letter	
					no special characters	at least one special characters



Test cases	result
Incorrect email pattern	"re-enter email"
Invalid EIN	"re-enter EIN"
length <6	"Invalidated Password"
1234567890	"Invalidated Password"
012345678a	"Invalidated Password"
Abcdefghi	"Invalidated Password"
A12345678	"Invalidated Password"
A%12345678	"Invalidated Password"
pw is different with the one the user typed in previously	“Incorrect Password”

Table 7.2.8 – Charity Sign Up Equivalence Partitioning Testing

White Box Testing -

a. State Testing –

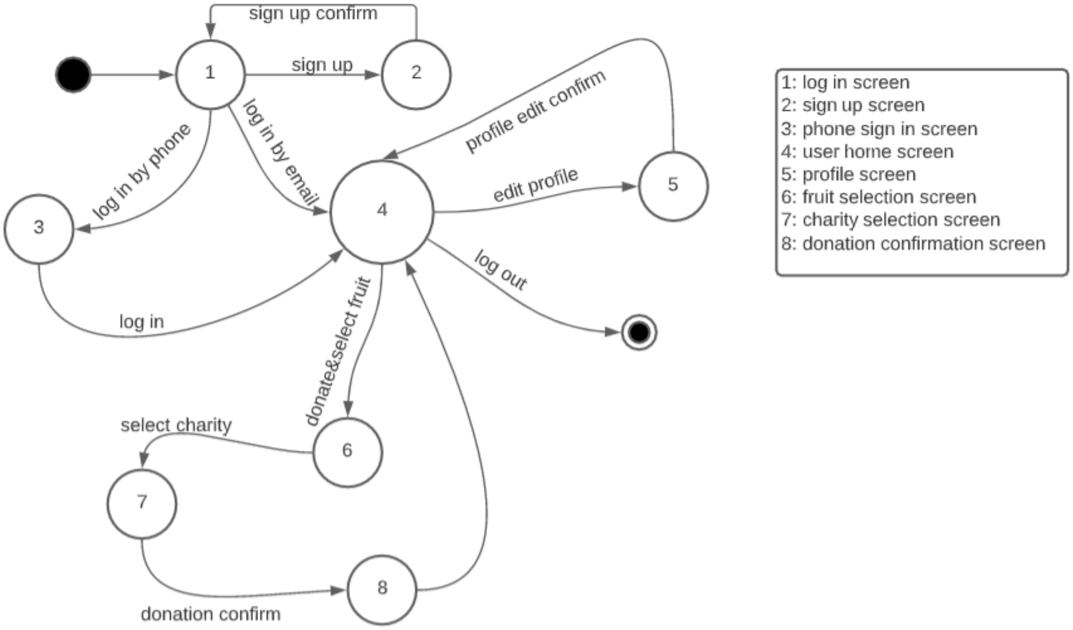


Figure 7.2.1 Complete Donation State Testing

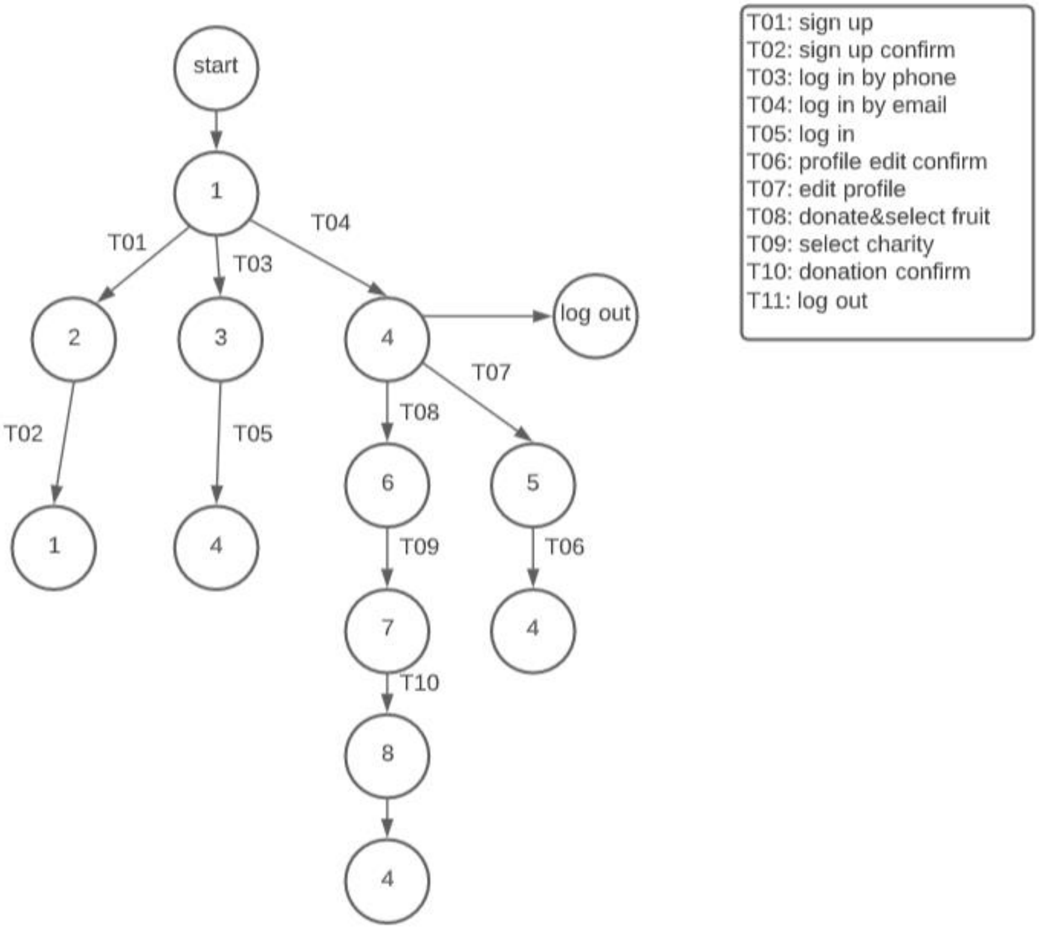
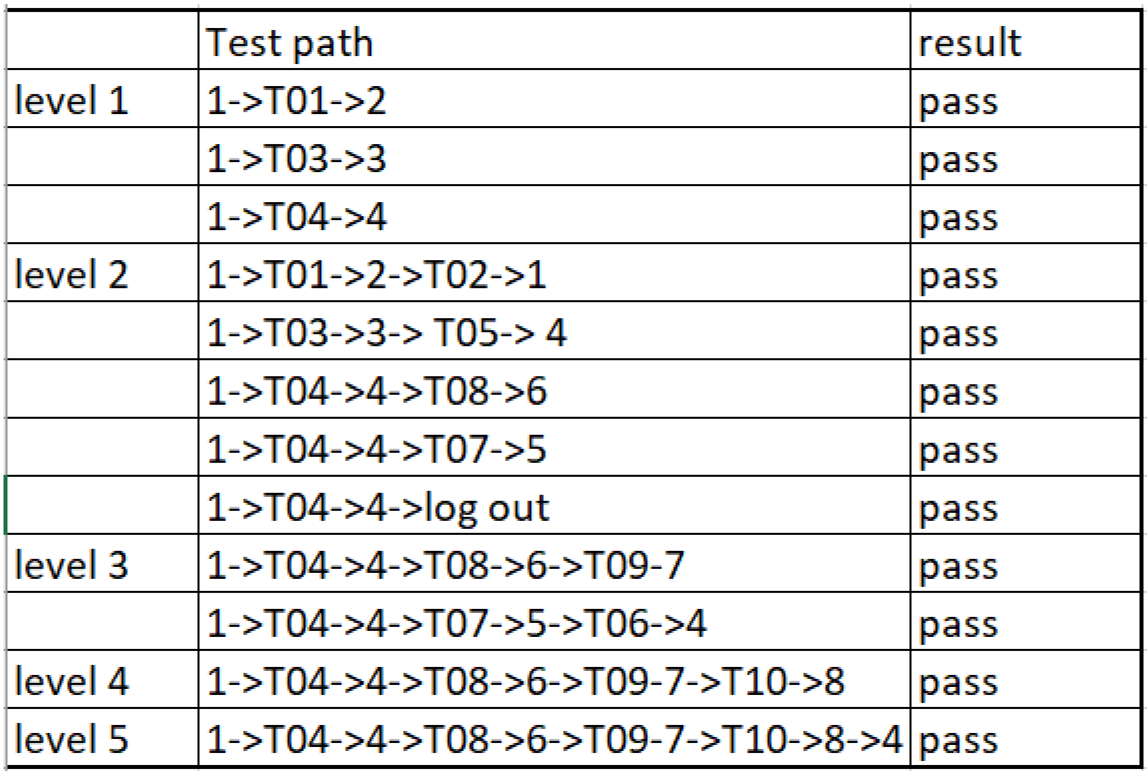


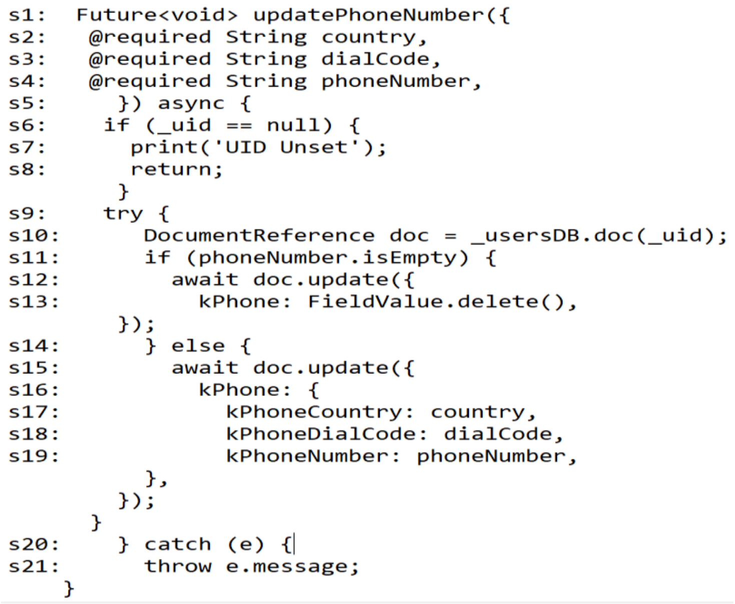
Figure 7.2.2 Path Testing



	Test path	result
level 1	1->T01->2	pass
	1->T03->3	pass
	1->T04->4	pass
level 2	1->T01->2->T02->1	pass
	1->T03->3-> T05-> 4	pass
	1->T04->4->T08->6	pass
	1->T04->4->T07->5	pass
	1->T04->4->log out	pass
level 3	1->T04->4->T08->6->T09-7	pass
	1->T04->4->T07->5->T06->4	pass
level 4	1->T04->4->T08->6->T09-7->T10->8	pass
level 5	1->T04->4->T08->6->T09-7->T10->8->4	pass

Table 7.2.9 – Complete path testing

b. Branch Testing - function of ‘update phone number’



```
s1:   Future<void> updatePhoneNumber({
s2:    @required String country,
s3:    @required String dialCode,
s4:    @required String phoneNumber,
s5:      }) async {
s6:     if (_uid == null) {
s7:       print('UID Unset');
s8:       return;
         }
s9:     try {
s10:       DocumentReference doc = _usersDB.doc(_uid);
s11:       if (phoneNumber.isEmpty) {
s12:         await doc.update({
s13:           kPhone: FieldValue.delete(),
         });
s14:       } else {
s15:         await doc.update({
s16:           kPhone: {
s17:             kPhoneCountry: country,
s18:             kPhoneDialCode: dialCode,
s19:             kPhoneNumber: phoneNumber,
           },
         });
       }
s20:     } catch (e) {
s21:       throw e.message;
     }
```

Figure 7.2.3 Update phone number function

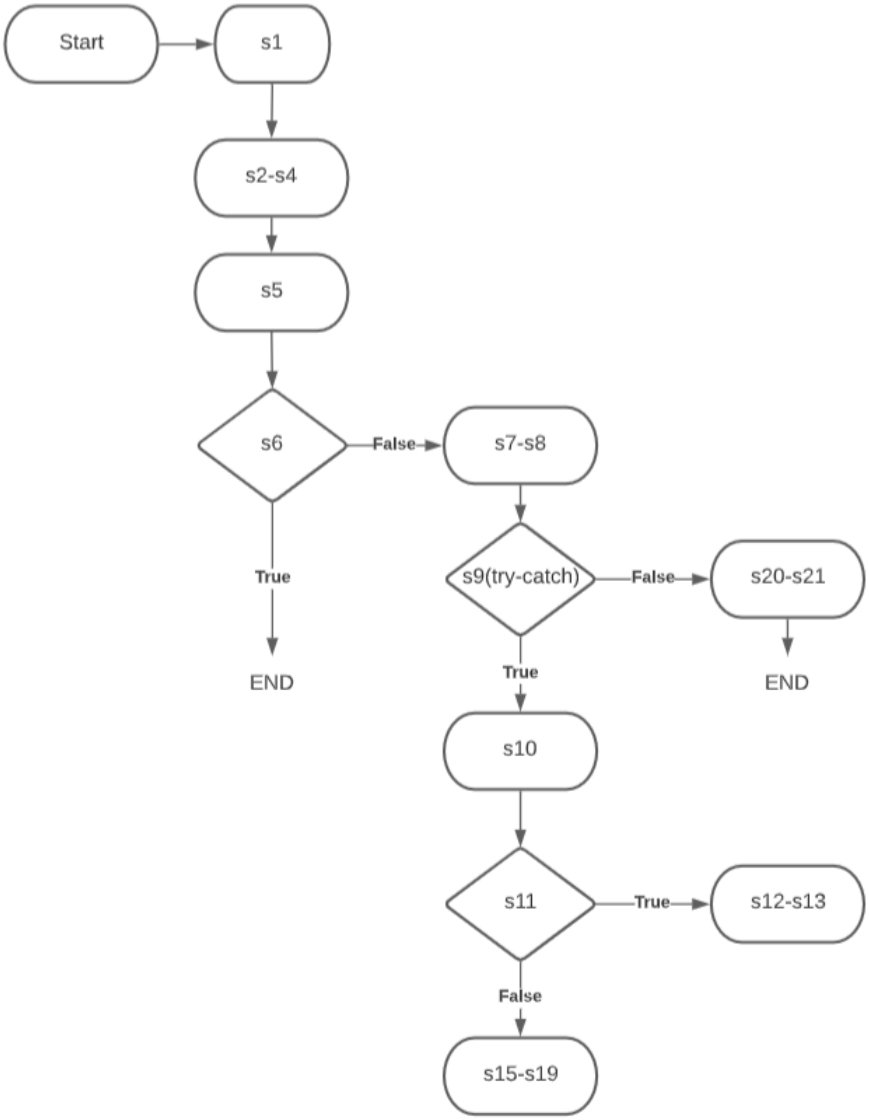
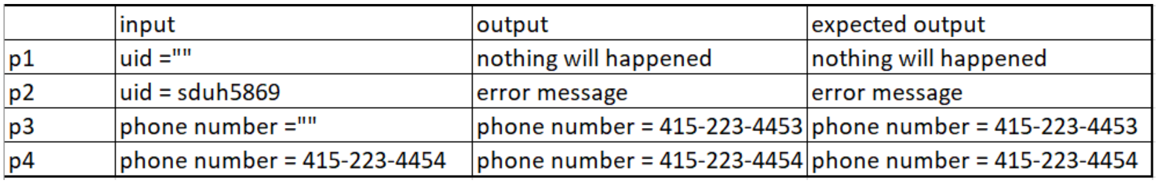


Figure 7.2.4 – Update phone number branch testing diagram

Path:

1. start -> s1-> s2-s4-> s5-> s6-> end
2. start-> s1-> s2-s4-> s5-> s6 -> s7-s8-> s9-> s20-s21
3. start-> s1-> s2-s4-> s5-> s6 -> s7-s8-> s9 -> s10->s11-> s12-s13
4. start-> s1-> s2-s4-> s5-> s6 -> s7-s8-> s9 -> s10->s11-> s15-s19



	input	output	expected output
p1	uid =""	nothing will happened	nothing will happened
p2	uid = sduh5869	error message	error message
p3	phone number =""	phone number = 415-223-4453	phone number = 415-223-4453
p4	phone number = 415-223-4454	phone number = 415-223-4454	phone number = 415-223-4454

Table 7.2.10 – Phone number update branch testing

7.3 Testing and Experiment Results and Analysis

The results from the testing phase for the application are satisfactory. The application worked as intended and performs each task in a minimum amount of time between one to three seconds. The coverage of the test cases is not complete, and more testing needs to be completed especially testing related to load and stress levels and scalability purposes of database usage. The large portion of testing was executed during the development of the application. The result of each separate test is shown in section 7.2 above.

Another type of testing that will need to be performed in the future is user experience testing. A qualitative approach will be used to continue improvement of the application.

Chapter 8 Conclusion and Future Work

Food waste is a major concerning issue around the world. Our application (Fruit Fairy) intends to combat food waste by making donating fruits and vegetables accessible to every home gardener. Creating Fruit Fairy was a lengthy process that has taught us multiple skills such as teamwork, UI development approach, usage of multiple API's, user authentication, backend, testing procedures, and the flutter framework. The creation of the application Fruit Fairy was a success, and all key features work as intended.

For future work, the project team intends to approach local charities and possible donors to ask them to sign up to the application and begin the process of donations. From an initial survey among people the project team personally know, donors are very willing to sign up to Fruit Fairy. In addition, more testing needs to be performed such as qualitative research by asking users how they felt using the application and what changes they recommend if any.

References

1. Apple. (2021). from https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

2. Hsiao CY., Liu YJ., Wang MJ.J. (2013) Usability Evaluation of the Touch Screen User Interface Design. In: Yamamoto S. (eds) Human Interface and the Management of Information. Information and Interaction Design. HIMI 2013. Lecture Notes in Computer Science, vol 8016. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39209-2_6

3. Moravcik, Oliver & Petrik, Daniel & Skripcak, T. & Schreiber, Peter. (2012). Elements of the Modern Application Software Development. International Journal of Computer Theory and Engineering. 891-896. 10.7763/IJCTE.2012.V4.601.

4. Thomas, A. (2020, April 17). Top 6 Tech Stacks That Reign Software Development in 2020: Fingent Blog. Retrieved September 13, 2020, from https://www.fingent.com/blog/top-6-tech-stacks-that-reign-software-development-in-2020/