AN ABSTRACT OF THE MASTER'S PROJECT REPORT OF

Amulya Gangam for the degree of Master of Science in Computer Science presented on December 7, 2022.

Title: Share What You Can

Abstract approved: _____

Dr. Will Braynen

Food is our primary source of nourishment. However, with the growing population, there has been a substantial increase in food waste by both restaurants and individuals. *Share What You Can* is an Android application designed to tackle this problem. The application operates as a two-sided marketplace to which *donors* upload images of and details about surplus food they want to share with *receivers*. *Donors* advertise that they have food by uploading images of the food they want to donate with an additional description that includes the food details. *Receivers* go through the *donors'* posts and, if they like some food item from the listings, then they can request it. Moreover, food expiration dates are rarely considered in most food-sharing applications. To avoid the possibility of donating expired food, *Share What You Can* allows *donors* to scan expiration dates, ensuring only fresh food is shared.

Share What You Can

by

Amulya Gangam

A MASTER'S PROJECT REPORT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 7, 2022
Commencement June 2023

Master of Science master's project report of Amulya Gangam presented on December 7, 2022.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my master's project report will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my master's project report to any reader upon request.

Amulya Gangam, Author

ACKNOWLEDGEMENTS

I want to convey my heartfelt gratitude to Will Braynen, my advisor, for making my project possible. He is the best mentor I could ever get. His unwavering assistance has enabled me to grasp a number of industrial practices. He never hesitated to correct me; and his iterative approach to examining my work was highly constructive. Morgan Lutz, my industrial mentor, deserves special recognition. Her excellent feedback helped shape my project and introduced me to latest Android concepts. I want to appreciate my committee members, Dr. Bella Bose and Dr. Mike Bailey for making time in their busy schedules. They never failed to respond to my emails, and during my defense, I received fantastic remarks and recommendations. I want to convey my deepest gratitude to my parents for their constant emotional support. They always encouraged me to strive more, which was a massive help for me throughout my academic career. They believed in me and motivated me even when I doubted myself. I will always be grateful to my parents and will do my best to make them proud. Finally, I thank God for showering me with unconditional love and being my most incredible supporter throughout my life. He empowered me with his blessings.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

Chapter 1: Introduction

Pope Francis said, "Throwing away food is like stealing from the table of those who are poor and hungry" (Francis, 2013). The Android application *Share What You Can* facilitates the donation of surplus food by individuals or food pantries by connecting *donors* with *recipients*. Its main objective is to reduce food wastage.

1.1 Background

The problem of food wastage dates back to the 1800s when industrialization made food transport from farms and factories easier for consumer; because farms and factories were able to produce more food than was required, people began to dispose of food (Chen & Chen, 2019; Nunley, 2013). Estimates of food waste range from 1.3 billion tons globally "per year" (Schanes, Dobernig, & Gözet, 2018), which is an annual 2600 billion pounds globally, to 133 billion pounds in 2010 for the United States alone (Buzby, Farah-Wells, & Hyman, 2014). Approximately 30 to 40 percent of the American food supply is wasted (Hall, Guo, Dore, & Chow, 2009). Only 3 percent of food waste is "diverted to emergency food programs", while the other 72 percent is disposed of in landfills instead of getting composted (Griffin, Sobal, & Lyson, 2022).

My best guess is that this happens because people often overestimate how much

food they need. They over-prepare for gatherings, parties, and functions and end up throwing the extra food away. Because most food banks do not accept prepared food, restaurants trash out leftovers. Grocery stores discard food products that go unsold because of overproduction and over-ordering of food.

Many governments and organizations have introduced awareness programs and laws to decrease these losses. In the following year, the USDA and the EPA together coined the term "The U.S. Food Loss and Waste 2030 Champions", awarding the title to "businesses and organizations that have made a public commitment to reduce food loss and waste in their own operations in the United States by 50 percent by the year 2030" (*U.S. Food Loss and Waste 2030 Champions*, 2017). Despite the promising momentum and recognizable progress, much more work remains.

In an effort to help minimize food waste, I have developed a mobile application called *Share What You Can* for the Android platform. Using this application, individuals and organizations can donate edible prepared foods, including packaged foods and ingredients. Those in need of food can also use the application to browse the available food and request the food of their choice. The *Share What You Can* provides direct interaction between the users of the application, so that the food*donor* will know to whom their food is donated to. Furthermore, this application uses OCR technology to scan the expiration dates of the food. So that only fresh food can be donated. The food post will be automatically removed if the food expires after the post is left unnoticed for a long time, thereby ensuring only fresh food is donated.

1.2 Existing Solutions

Currently, there are no applications that can be used by both food pantries and individuals. Furthermore, only a few applications check the food's expiration date, meaning that some people may be donating usable food. The following are some food donation applications available in the Google Play store.

Share The Meal Charity Donate(Amarri, Corbi, Randall Smith, & Wu, 2015) is a 2020 Nobel Peace Prize-winning Android application that allows users to donate money to help the underprivileged buy food. Its primary premise is that small donations can have a tremendous impact when given in large numbers; thus it collects thousands of small contributions that add up to significant results in the lives of people who struggle to afford food. The application is limited, though, in that it can only meet its goal if people continue to contribute money.

Careit Food Donation(Schill, 2022) is an Android application that connects businesses and institutions (*donors*) that have surplus food to non-profit food recovery agencies that need that food (*receivers*). Non-profits look for nearby food of interest and either pick it up from the organization or arrange for a drop-off. One of the limitations of the application is that only organizations (as opposed to individuals) can donate or receive food.

Meal connect,(Claire, Byrdak, Fitzgerald, Hall, & Bernard, 2017) is an Android and web application that facilitates the donation of surplus food by connecting *donors* with volunteers who can distribute food. When uploading the details of the available food, the *donor* also provides their Zip code and preferred pickup

time so that nearby volunteers pick up the items and redistribute them.

This application's main drawback is that it doesn't allow *donors* to directly connect with and donate food to *receivers*.

1.3 Proposed Solution

The solution I propose is an Android application that aims to minimize food wastage by providing a platform to donate and receive food. This simple and user-friendly application is compatible with all Android devices and serves as an interface between the *donor* (food pantries and individuals) and the *receiver* (general public). Further, *Share What You Can* would not limit food distribution to low-income groups. Anyone in need of food would be eligible to receive it.

Organizations and individuals willing to contribute surplus food could register for the application, while individuals seeking food could locate nearby food donations and contact the respective food *donor* to request a specific food item. *Donors* would then review and accept or deny those food requests based on the item's availability.

Chapter 2: Product requirements

2.1 General public as food donors, not just food pantries

Most food-sharing applications in the Google Play store and App stores connect commercial entities, such as grocery stores, restaurants, and catered event organizers, with non-profit organizations (food banks) needing food. Not many applications include all types of users.

Share What You Can enables both food pantries and the general public to contribute food by facilitating the direct connection between food *donors* and *recipients* without the use of intermediaries. As a result, *donors* know where exactly their food is going rather than blindly depending on a distributor.

2.2 System Description

The project is divided into three modules: *donor*, *receiver*, and the Login module, where Login is the common module for both *donor* and *receiver*.

Login Module:

The application authenticates each user by using a device ID, a unique identifier assigned to each device, such as a smartphone, iPad, tablet, or laptop.

Donor Module:

In this module, *donor* can donate surplus food to the public. To accomplish

this, they advertise the donation by including a name, image, description for the food item, contact details, type of food, and expiration date. The system prohibits the user from uploading expired food items. The post will be automatically removed if the food is unnoticed for a long time and expires after it is posted.

The modules can be further classified as:

1. Sign in
2. Select user type
3. Home page
4. Add food details
5. View requests
6. Approve requests
7. View donation history
8. Logout

Receiver Module:

In this module, the *receiver* can view posts that include the details of food items uploaded by the *donor*. The module sorts the posts in descending order so that the most recently uploaded posts appear on the top. After viewing the food donation posts *receiver* can request a specific food item and contact the *donor* for further details.

I further classified this module as follows:

1. Sign in
2. Send Request
3. Connect with *donor*

2.3 Screen Flow chart

The flow chart below is intended to illustrate the way that the application is used, in addition to providing the functional descriptions of the individual users of the application.

Step 1: To begin, the user must launch the *Share What You Can* application.

Step 2: Following this, the application will prompt the user to identify their role as either a *donor* or a *recipient*.

After the user type is selected, the order of application screens differs based on the following cases.

Step 3: Case 1: *First-time user of the application*

- To use the application for the first time, users must register with their device ID.
- If the user is logged in as a *donor*, then they must select their *donor* user-type (pantry or individual). If logged in as a *receiver*, the current step is skipped.

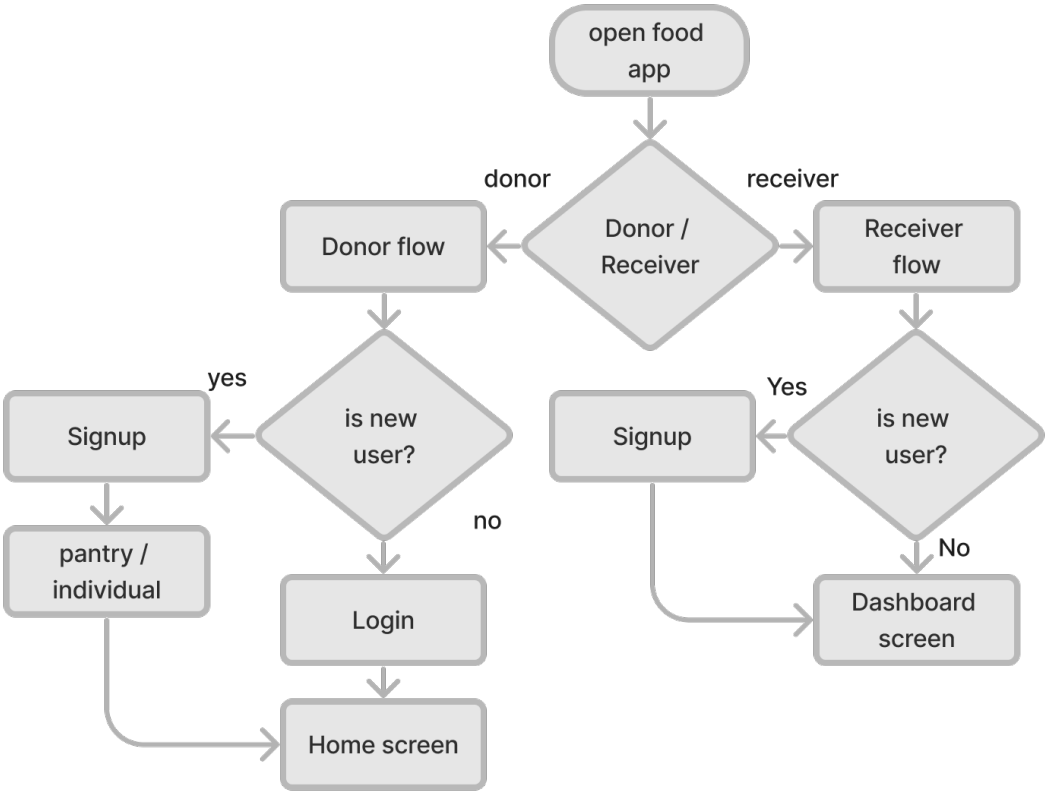


Figure 2.1: Authentication

- After the user makes their selection, the application will direct the *donor* to the home screen and *receiver* to the dashboard screen where the food

donation posts are listed.

Case 2: *Not a first-time user*

- The user can directly login to the application.
- After signing-in *donor* gets navigated to the home screen and the *receiver* to the dashboard screen of the application where the food donation posts are displayed.

Step 4: The *donor* can perform the following operations on the home screen (Figure 2.2):

(a) Add food details

(b) View food requests received from the *receiver*.

(c) Logout

Step 5: Add food details (Figure 2.2):

On the home screen, the *donor* can upload the details of food such as

1. Food name and description: In order to aid the *receiver* in identifying the food, the *donor* must add a short name and description of the food.

2. Food Type: The *donor* must choose a category for the donated food item. The following are the allowed food types:

(a) Cooked Food

(b) Packaged Food

(c) Groceries

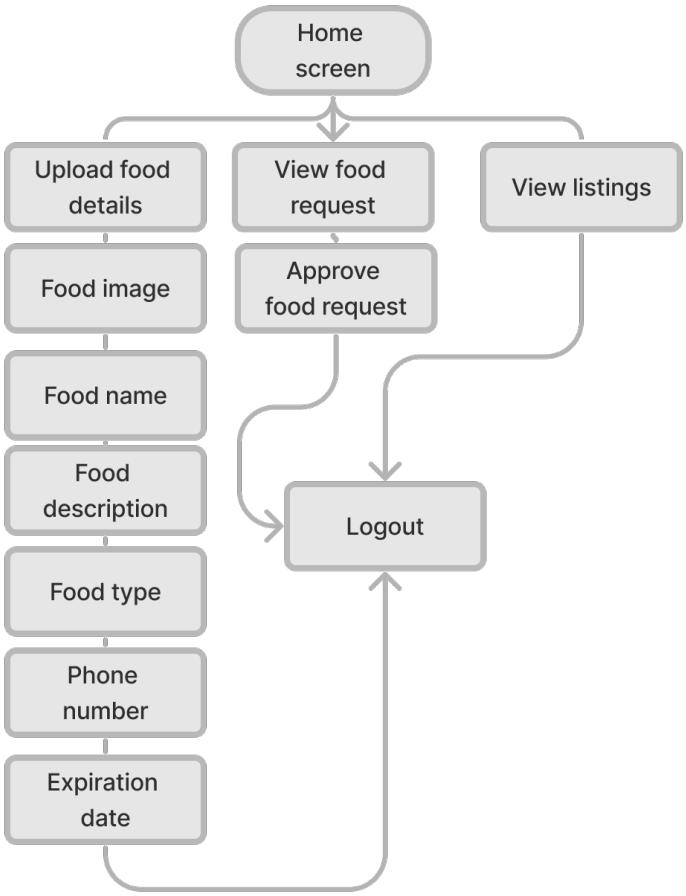


Figure 2.2: donor home screen flow

(d) Ingredients

4. Expiration date: The final step before uploading the food details is to provide its expiration date. The *donor* can use this application to either scan the expiration date or enter the date manually. The manual entry is provided considering the case when the date on the product is not clear(Figure 2.2).

Query:

"*Current date less than the expiration date*"

There is a possibility that a food *donor* may accidentally donate expired food. As a precautionary measure, *Share What You Can* uses an optical character recognition (OCR) feature to scan the food's expiration date and identify expired food. To make the food available for donation, the current date must be less than the expiration date. If the above condition fails, the system does not allow the user to donate food.

Step 6: View food donation posts

The application immediately updates the *receiver* dashboard with the *donor's* food donation posts (Figure 2.3).

Step 7: Request for the food

The *receiver* can check for the food of their choice in the list of food donations and requests a specific food item (Figure 2.3).

Step 8: View food Requests from *receiver*

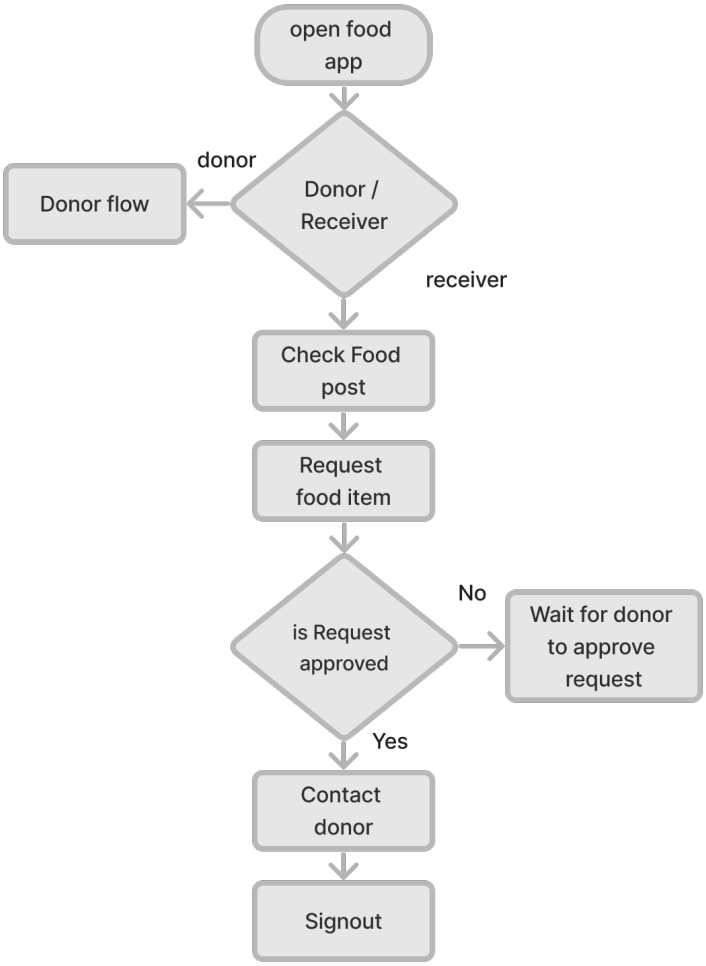


Figure 2.3: Receiver flow

The food *donor* views the food requests and decides whether or not to approve them. When a request is approved, the application assigns the food item to the *receiver*, and after this, the *donor* can donate the approved food item by confirming the popup that says "*Are you sure you want to donate the food?*." When the food is donated, the application automatically removes its associated post from *receiver* dashboard (Figure 2.3).

Step 9: If the *receiver* food request is approved by the *donor* both the users can further communicate regarding the status of food, the mode of food transport (pickup / drop off), and the time and date that the food can be provided. If the *receiver* food request is not approved by the *donor*, it can be because of the delayed response or the food might have already been assigned to another person. In those cases, the *receiver* can wait for the request to be approved or request another food item (Figure 2.3).

Step 10: The user can choose to logout to visit the select type of user (*donor* / *receiver*) screen.

2.4 Use-case diagram

In this chapter, I discussed various use cases of the *Share What You Can* application. This use-case diagram demonstrates the interaction of the system and entities outside the system, known as *actors*. *Actors* can be humans or other external hardware or other systems.

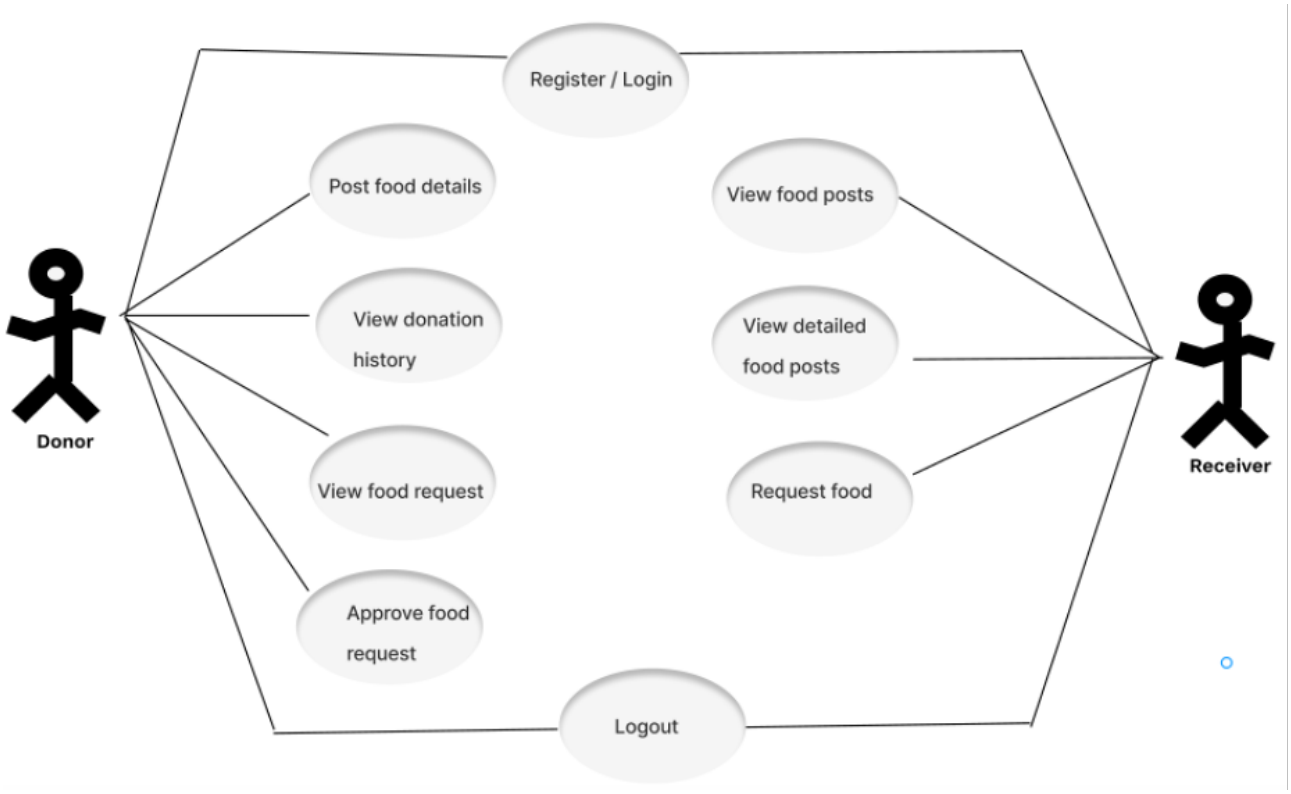


Figure 2.4: Use case Diagram

As shown in the use-case diagram above, the user (*donor* or *receiver*) registers or signs in to the application. While the *donor* posts the food details, the *receiver* checks for the nearby food and chooses based on his/her personal choice. On clicking the individual food item, the *receiver* can view detailed food posts from which food requests can be sent. After receiving a request, the food *donor* views it and decides to approve or disapprove it.

Actors: *donors* and *receivers* are the *actors* of the system.

Use cases:

Name	Register
Description	The user registers with device ID.
Actor(s)	*donor* and *receiver*
Flow	The following use case starts when the user still needs to register with the application. a) The system prompts the user to register with the device ID. b) The user clicks on the sign-in button. c) The system stores the device ID and creates an account.
Success Case	The system authenticates the user with their device ID and redirects them to the home page.
Error Case	When the user attempts to register without an internet connection, the system displays an error message.

Table 2.1: Register use case

Name	Post food details
Description	The user adds food details.
Actor(s)	*donor*
Flow	This use case begins when the *donor* logins to the application. a) The system prompts the user to select the donor type (pantry or individual). b) The user navigates to the home screen to select the donor type. c) The user selects the upload food button. d) The system prompts the user to fill in the food details, including food name, description, cooked hours, and expiration date. e) The system verifies the information and uploads the food details.
Success Case	The user navigates to the home screen after entering the appropriate food details.
Error Case	The system prompts the user to upload food details in the following scenarios: Case 1: Invalid data The user enters an expiration date that is less than or equal to the current date. Case 2: Missing Fields a) The user attempts to upload food without entering details in all of the fields. The system displays an error message prompting the user to enter information in all the fields.

Table 2.2: Post Food details

Name	View Food requests
Description	The user views the list of food requests sent by the *receiver*.
Actor(s)	*receiver*
Flow	The following use case starts when the user is on the application's home screen. a) The user chooses the food request option from the drawer. b) The user views the list of food requests sent by the *receiver*.

Table 2.3: View Food Request use case

Name	Approve Food requests
Description	The user approves the food request.
Actor(s)	*donor*
Flow	The following use case starts when the user is on the application's home screen. a) The system prompts the user to approve the food request. b) The user approves one of the multiple food requests. c) The system prompts the user to confirm the donation of the food. d) After the confirmation of the food donation, the system removes the post from the *receiver* dashboard.
Result	The user successfully approved the food request

Table 2.4: Approve Food Request use case

Name	View Food Posts
Description	The user views the list of uploaded food items.
Actor(s)	*receiver*
Flow	The following use case starts when the user is logged into the application. a) The user views the list of food items uploaded by the *donor*.

Table 2.5: View Food posts Use-case

Name	View detailed food posts
Description	The user views the detailed view of the food item.
Actor(s)	*receiver*
Flow	The following use case starts when the user is logged into the application and clicks on a specific food post. a) The user views the detailed view of the food item, and the system displays the additional details about the food and the *donor* contact information to the user.

Table 2.6: View detailed Food posts Use case

Name	Request food
Description	The user views the list of food requests sent by the *receiver*.
Actor(s)	*receiver*
Flow	The following use case starts when the user is on the application's home screen. a) The user chooses the food request option from the drawer. b) The user views the list of food requests sent by the *receiver*.

Table 2.7: Request Food

Chapter 3: Implementation

3.1 Technologies and technical terms used

1. Firebase: Firebase is a serverless platform used for developing high-quality mobile and web applications for both Android and iOS. Because it is cloud-based, developers can use Firebase to sync their cloud data across different devices or share it with multiple users.

2. Kotlin: Kotlin is the cross-platform and general-purpose programming language built by JetBrains. It is a Java-compatible language that may operate on the Java Virtual Machine (JVM). Google has officially recognized Kotlin as an Android programming language, and currently, most Android companies are migrating from Java to Kotlin.

3. Sealed Classes:

   A sealed class is a superclass listed with data states, objects, functions, and classes. To maintain type safety, a sealed class follows a restricted class hierarchy meaning that its instances can only have a limited set of types but cannot have any other type. Furthermore, this type can be determined when the class is compiled. For instance, third-party clients won't be able to extend others' sealed classes in their code. Sealed classes are an extension of an enum class. Each subtype in the sealed class can have different properties.

4. COIL:

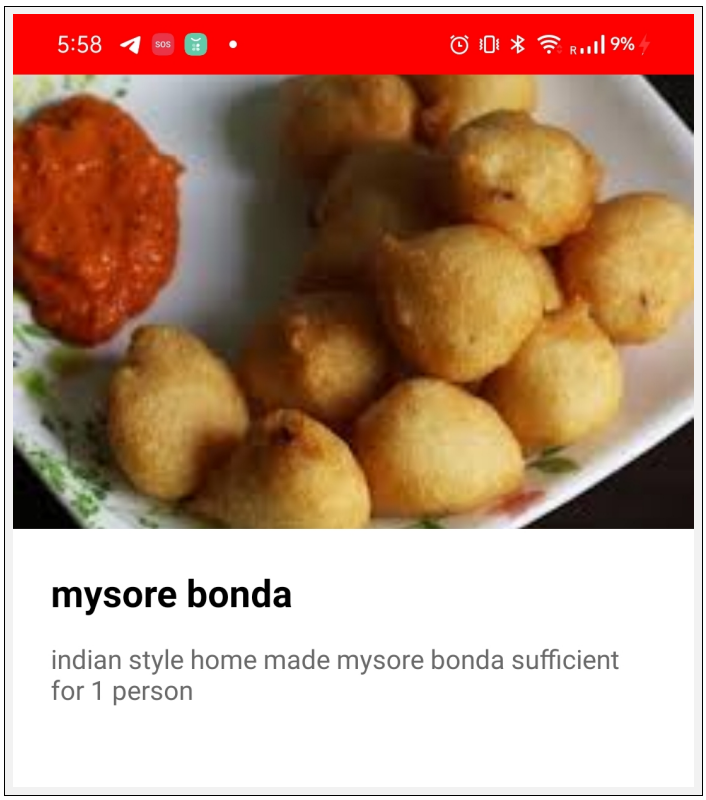


Figure 3.1: Implemented using COIL

Coroutine Image Loader (COIL), the image loading library I used in this application, is used to load images from the backend. Its extensive set of features simplified and eased the process of loading images. Key features of COIL that any developer will find useful include the following:

1. The images can be loaded by simply calling the function load.

2. The developer can apply various transformations, such as blur (applies

blur), circle crop (crops and centers the image into a circle), and rounded corners (rounds the corners of the image).

5. LiveData(*LiveData Overview*, 2022): LiveData is an observable data holder class with a UI (views) that observes the LiveData object, which holds some of the data that the UI wants to show on the screen. When the LiveData changes, the UI updates itself with new data, making it easier to maintain what's going on screen by staying in sync with the data (Figure 3.2).

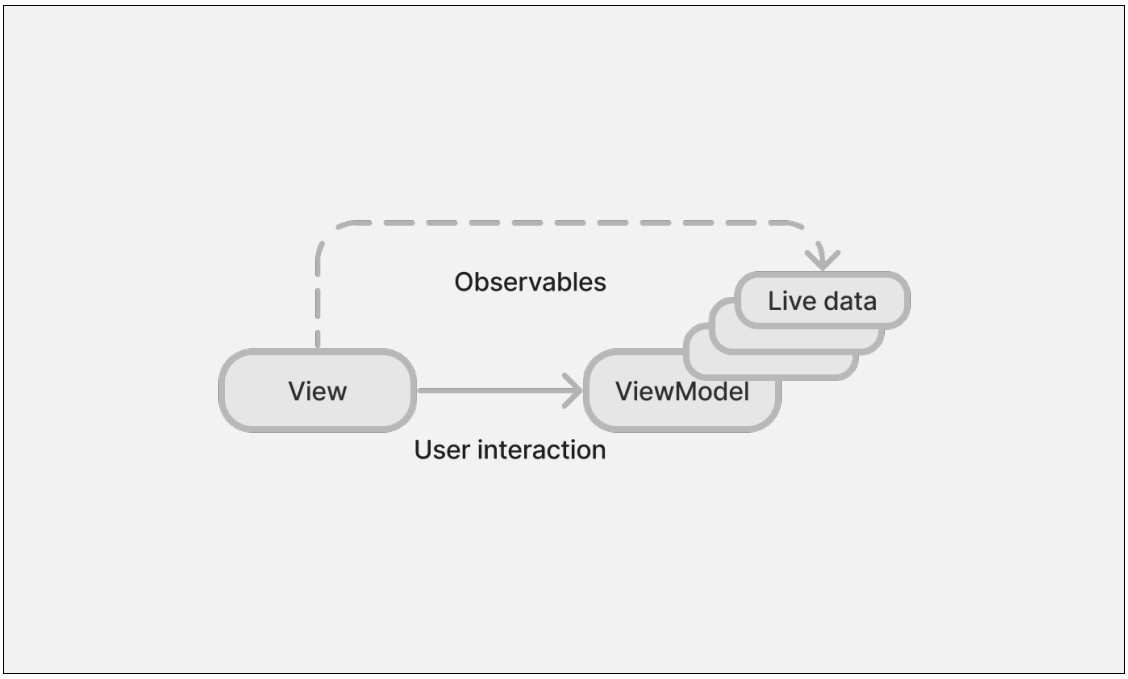


Figure 3.2: Updating UI from View Models

6. Activity

Activity is responsible for handling the user interactions with the application.

Essentially, it creates a window for holding the UI components.

7. Fragment

   Usually, fragments are referred to as sub-activities. The system breaks a single activity into fragments, allowing the developer to work on portions of the UI in smaller pieces.

   Fragment

8. MVVM:

   MVVM is the abbreviation for Model-View-View Model. The following three components are critical for achieving the separation of concerns.

   - Model: A model acts as a holder for the application data; it cannot directly communicate with the view.
   - View: Because views handle the immediate interaction with the user, they contain all of the code related to the UI. They do not, however, handle any business logic. Views also have UI elements that trigger events.
   - View Model: The fragments and activities can be broken down into views and view models. The view model handles the entire business logic, and its main responsibility is to read data from the model and expose it to the view. In short, it acts like glue between the UI and business logic. The view model tells the view what data to display and makes the appropriate data observable so that when some data in

the view model changes, all of the views observing the view model are notified about the change (Figure 3.2).

9. Data Binding: Data Binding, as the name suggests, is a technique that links the UI directly with the data source, ensuring that the UI stays up-to-date. In Android, developers perform data binding using the support library provided by Android Jet Pack. This technique is mainly used for MVVM architecture and helps eliminate traditional, tedious ways of using "FindViewById" calls, thereby increasing application performance.

```
_binding = FragmentRequestBinding.inflate(inflater, container, false)
binding.title.text = "Hello"
```

Just by using the binding variable ID and the properties provided to the reference in the XML, the system can modify the data. The developer must enable the view binding before they can use it.

```
 \ViewBinding = True"
```

Completing this step generates a binding class for all layout XML files. For redirecting to the XML file, viewBinding creates a separate binding object for each layout.

3.2 Architecture

3.2.1 Tech stack

I used the following tech stack diagram, as shown in Figure 3.3.

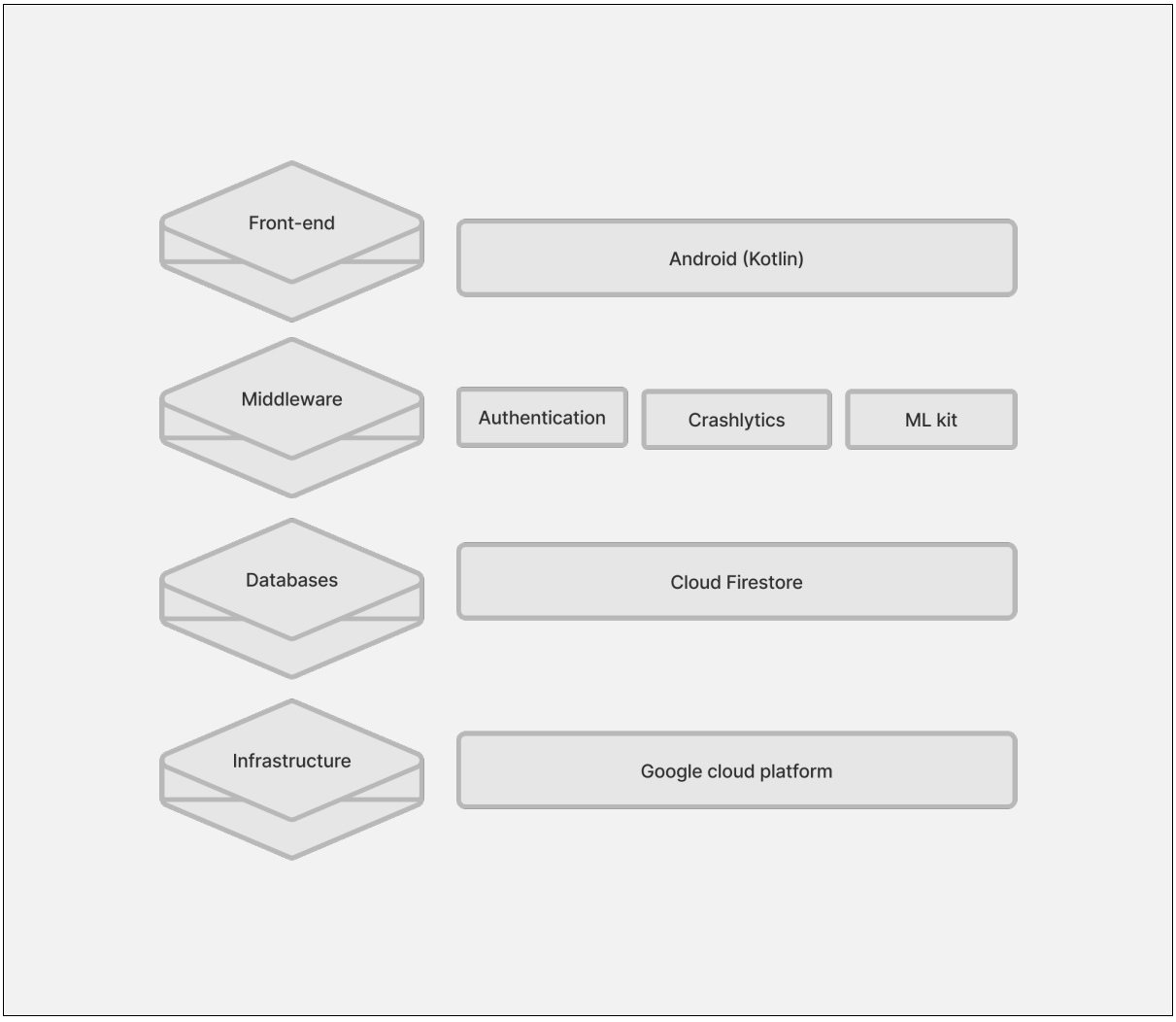


Figure 3.3: Tech Stack Diagram

3.2.1.1 Front-end development

I developed *Share What You Can* with the assistance of various technologies and tools. Android Studio is the Integrated Development Environment (IDE) I used to develop the application's UI, communicate with Firebase, and test the application.

3.2.1.2 Why Android?

I chose Android over iOS for the following reasons:

1. To practice my expertise in Android, I developed an application for Android users.

2. The majority of global market users use Android instead of iOS.

Android / iOS	Global Market Share	United States Market Share
Android	72.2	26.99
iOS	40.54	59.17

Table 3.1: Global vs United States market share in percentages

   Even though the United States has the highest number of iPhone users, with a market share of approximately 50 percent (Iaeme, 2013), Android dominates iOS in the Global Smartphone market, as shown in Table 3.1.

3. The application can be made easily available to low-income groups.

For frontend development, including developing the user interface, I used Android Studio IDE, Kotlin programming language, and Android XML

I have designed the UI for the Android application using Layouts and Widgets; In Layouts, child views can be positioned while in widgets objects such as buttons and text boxes can be positioned.

3.2.1.3 Choice of Programming language

Java	Kotlin

class PostModel {
    private String desc;
    private String title;
 	
    public String getTitle() {
        return title;
    }
 	
    public void setTitle(
    String title) {
        this.title = title;
    }
 	
    public String getDesc() {
        return desc;
    }
 	
    public void setDesc(
    String desc) {
        this.desc = desc;
    }
}
	data class PostModel(
	        val desc: String,
	        val title: String,
	    )

Table 3.2: Java and Kotlin comparison

Kotlin is an official Android programming language that supports a wide range of features.

Following is the list of the Kotlin features that were the most useful for developing *Share What You Can*.

- Null safety: Rather than allowing null safety errors to crash an application during run time, Kotlin detects them at compile time and suggests possible fixes.
- Extend functions from other classes: The developer can easily extend functions from other classes by calling their names.
- Smart Casts: In Java, the developer must manually identify variable types and typecast accordingly. By automatically typecasting, Kotlin simplifies the work.
- Data classes: Kotlin provides the automatic generation of getter-setter methods.

As shown in Table 3.2, in Kotlin there is no need to write the getter-setter methods manually because they are auto-generated.

3.2.1.4 Infrastructure

Google Cloud Platform is a cloud computing platform that provides many cloud computing services, some of which are exposed to Firebase for client-side developers. I used the following Google services in the development of *Share What You Can*:

1. Google Cloud storage Firestore
2. Firebase Crashyltics
3. Firebase Authentication
4. Firebase ML

3.2.1.5 Why Firebase?

Primarily, I chose Firebase because its ready-made features simplify the developer's job. The following are Firebase features I used for building *Share What You Can* that demonstrate the application's appropriateness for Firebase's platform services.

1. Cloud Firestore
2. ML Kit
3. Authentication
4. Crashlytics

3.2.1.6 Backend development

Cloud Firestore eliminates the need to maintain large servers, as well as networking, security, and scalability issues associated with a large user base. It automatically fetches data from the database as they happen or can request and fetch data manually. Additionally, its cloud database can store data online and sync it across devices or share it among multiple users. I used Firebase Firestore to store the user details, food details, and data related to food requests.

1. Real-time query results: The UI is up to date with the latest data changes.

2. High Performance: It can store large documents.

3. Offline Query Support: Cloud Firestone caches queried documents locally.

3.2.1.7 Middleware

1. ML Kit: The developer can integrate the trained machine learning model into an Android mobile application with a few lines of code. This feature requires no prior knowledge of neural networks or models.

   I integrated the Firebase ML Kit SDK to use the Optical Character Recognition feature. With the help of OCR, users can scan the expiration date on the product label.

2. Authentication: Firebase makes authentication easy for end users and developers. It facilitates the use of different modes of authentication, such as email/password, phone number, or sign-in as anonymous users. *Share What You Can* authenticates its users by using the email and password mode.

3. Crashlytics: I used the Crashlytics feature of Firebase exclusively to generate a detailed report on application crashes.

3.2.2 Client Architecture

I implemented the system using the Model-View-View Model (MVVM) Framework (Figure 3.4). My primary rationale for selecting MVVM is that it provides a clean architecture and good code maintainability. It also deconstructs the code-heavy activity or fragment into numerous single-purpose classes, breaking each fragment or Activity into a view (UI) and a view model (data). Because *Share What You Can* is a very small application, separating presentation logic from business logic was easy.

As shown in Figure 3.5, I implemented the project *Share What You Can* using views, view models, and models.

The view is the UI layer of the architecture, where all the presentation logic is handled here. It displays data on the screen and also allows user interactions. When the user starts interacting with the application, there can be a change in data. To reflect those changes, UI should be updated. So, view retrieves the UI state, i.e., application data from the view model, by observing the data exposed by view models using observables. Finally, views convert the observed changes into a form that UI can present and display.

View Model is the Data layer of the architecture, all the business logic related to the UI elements displayed in the view are handled here. If there is any interaction with the UI, then the UI notifies that event to the View Model. View Model handles these user interactions; the model fetches the data from Firebase Firestore and gives it back to the View Model. View Model is a data (state) holder that

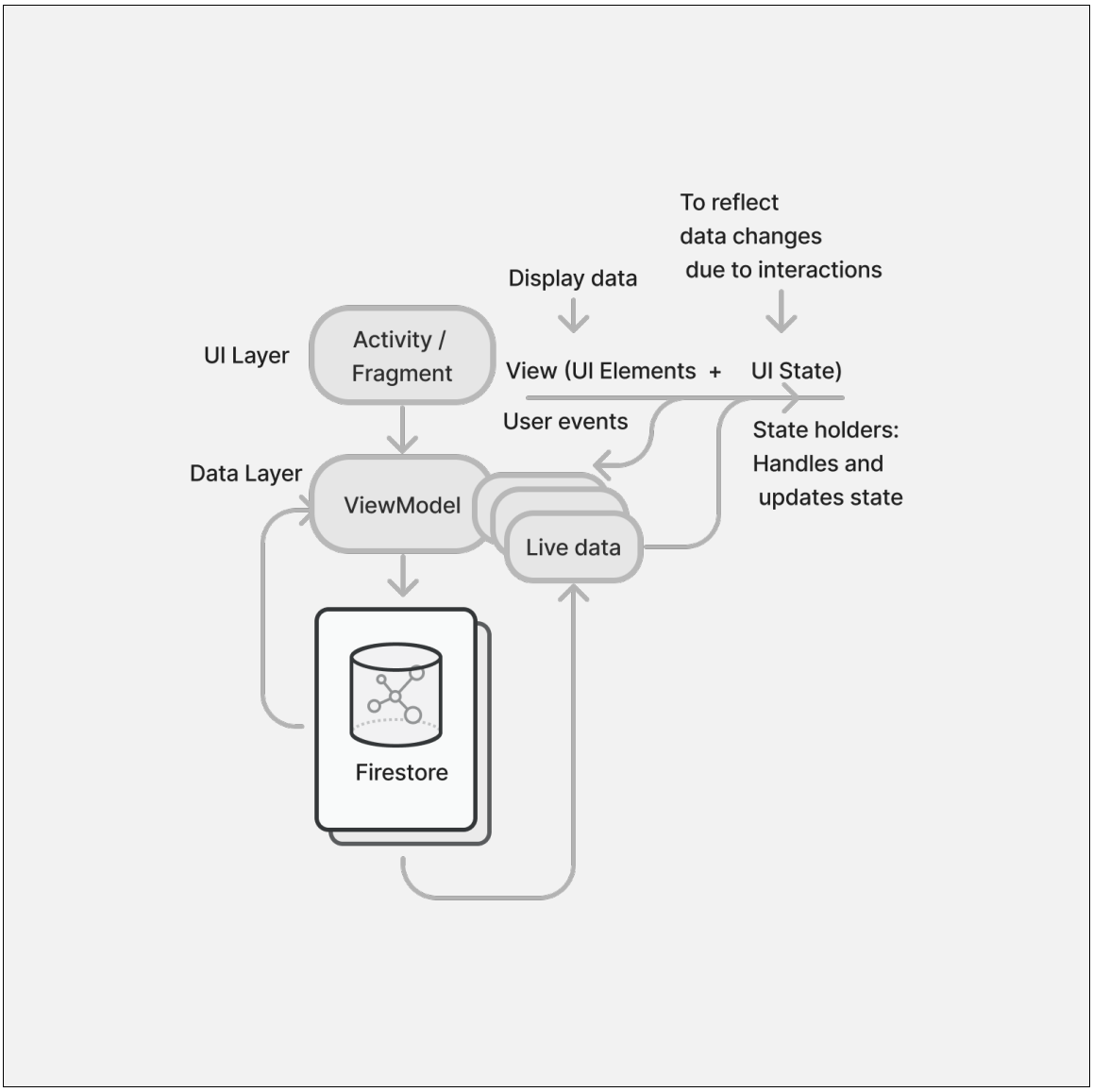


Figure 3.4: Framework

exposes this data (state) to the view (UI) in the LiveData. The updated LiveData object is exposed to the view (UI) to render it. The above process is repeated for

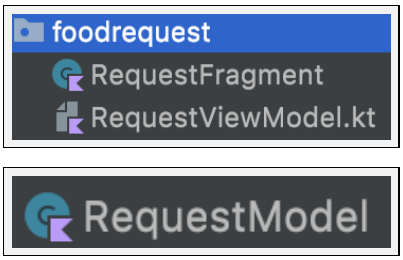


Figure 3.5: MVVM Architecture for Food Request class

every event needing a state change.

Chapter 4: Project setup

The GitHub link of the *Share What You can* project is attached here.

4.1 IDE Setup

For developing the application front end, I installed the latest version of Android studio from official website for Android developers

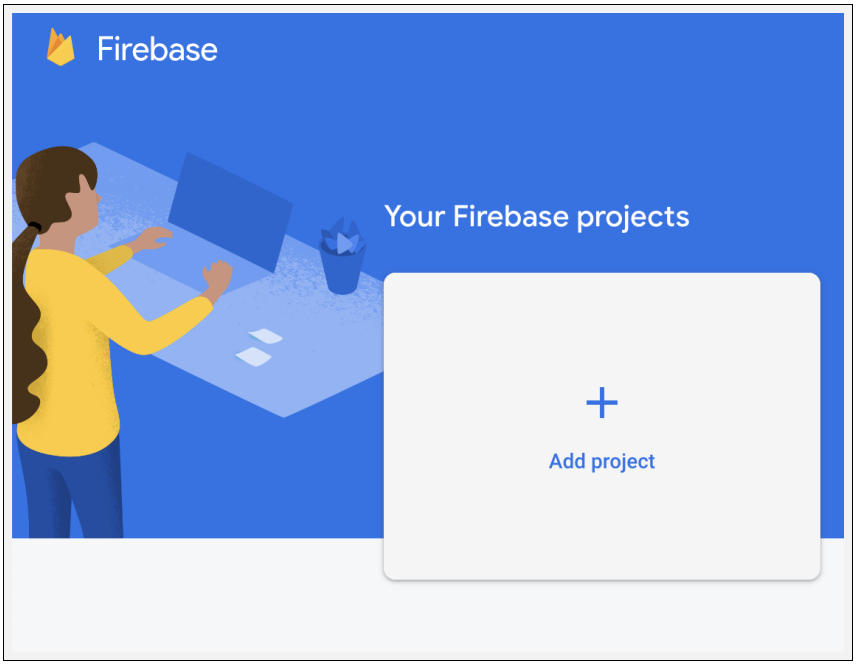


Figure 4.1: Create new project

4.2 Firebase Setup

The following steps are necessary for setting up the backend and connecting to the Android application: 1. Create a new Firebase Project 2. Connect the Android application to Firebase.

1) Create a new Firebase Project

As shown in Figure 4.1, I created a new Firebase project by clicking on "Add project" in the Firebase console.

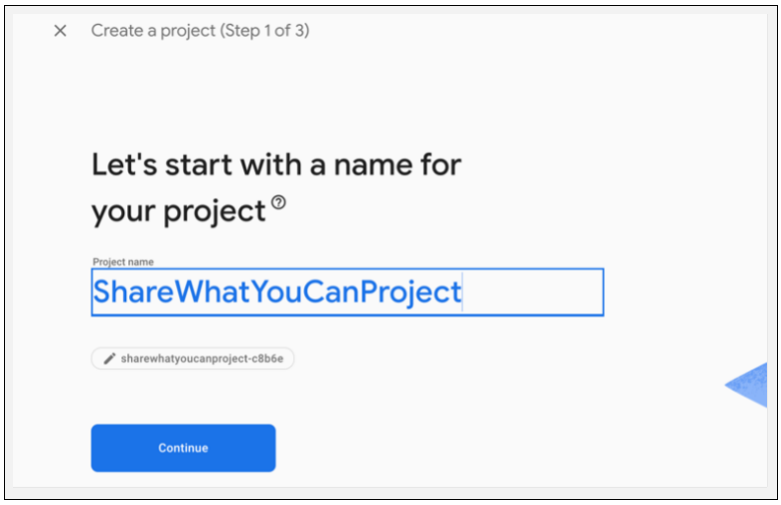


Figure 4.2: Name the project

Next, I added a project name (as shown in Figure 4.2).

In the final step, I selected the default Firebase account for configuring the Google Analytics Account, as shown in Figure 4.3

By clicking on Create Project, a new project with the previously mentioned application name is created.

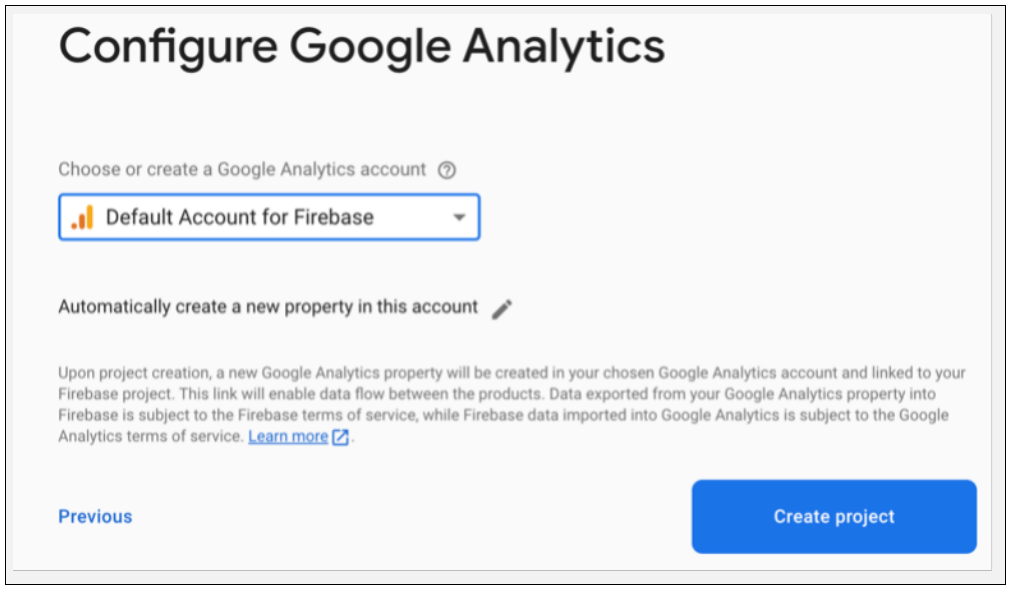


Figure 4.3: Choosing account for Google Analytics

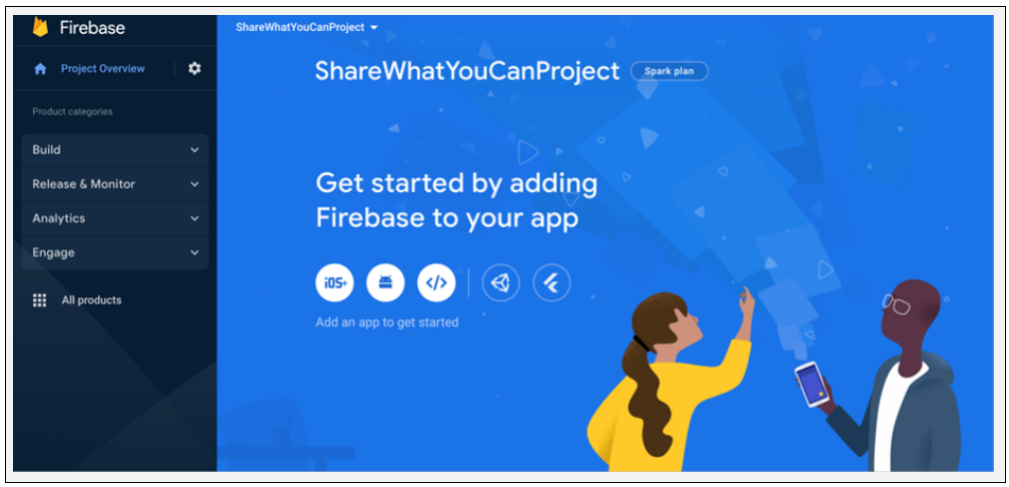


Figure 4.4: Adding Firebase to application

2) Connect the Android application to Firebase

To add the application to Firebase, we must register it with Firebase. The following steps describe how to do this.

Because I developed an Android application, the Android icon is selected, as shown in the Figure 4.4

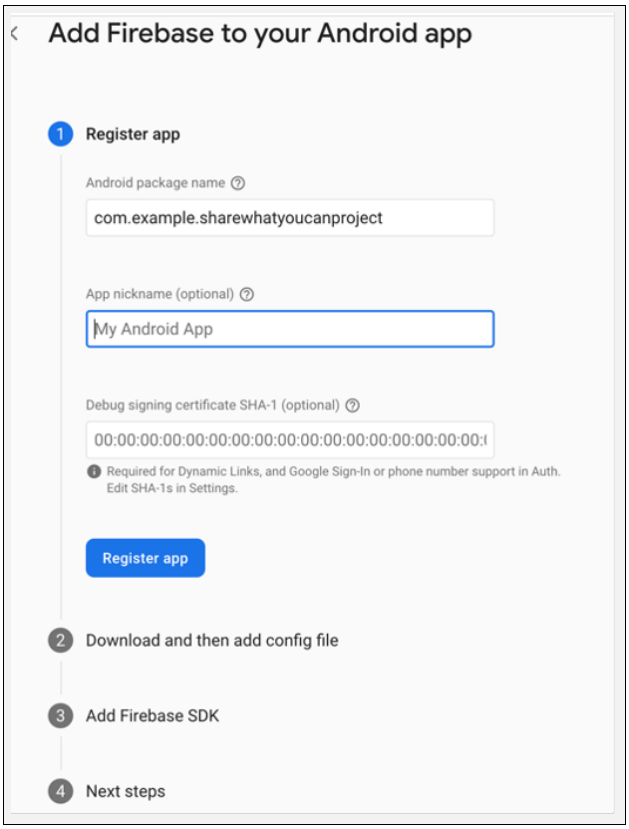


Figure 4.5: Adding package name

I added the package name of the application, as shown in Figure 4.5. After entering the package name, I registered the application with Firebase by clicking

on Register application (Figure 4.5).

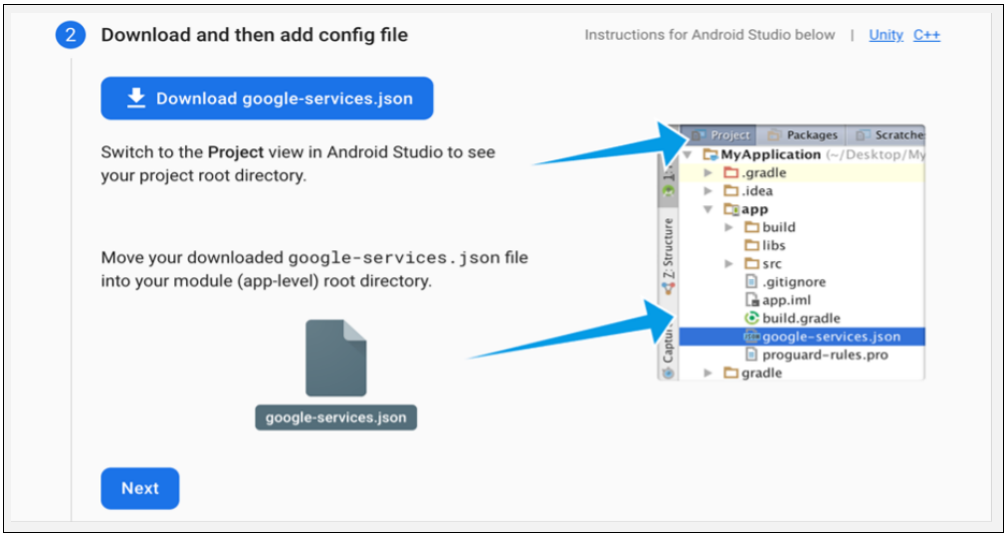


Figure 4.6: Downloading google-services.json file

Then, `google-services.json` file should be downloaded (as shown in Figure 4.6) and added to the application-level of the project (Figure 4.7)

To make the above JSON file accessible to Firebase SDK, google-services plugin should be added in the project-level and application-level `build.gradle` file.

Then the plugin needs to be added to the dependencies of the project-level `build.gradle` file.

```
classpath 'com.google.gms:google-services:4.3.13'
```

The google-services plugin should be added to the plugins section and both the google-services plugin and the Firebase-SDK should be added to the application-level build.gradle file.

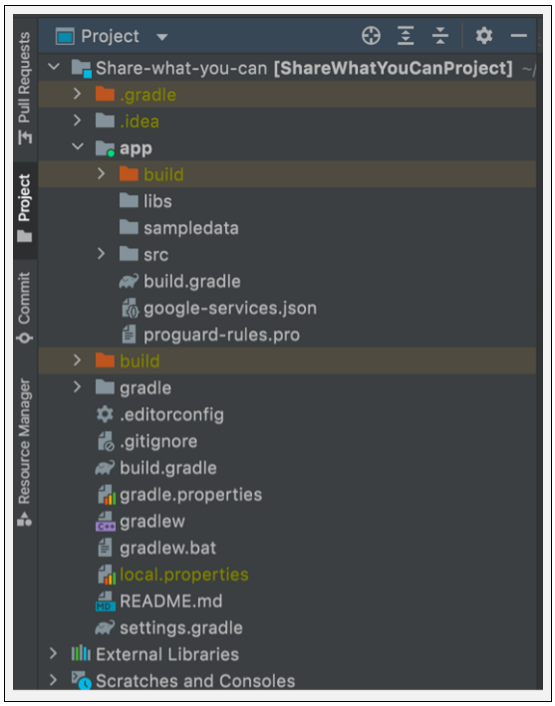


Figure 4.7: Adding google-services.json file

```
id 'com.google.gms.google-services'
```

At this point, all of the necessary Firebase SDKs have been added to the dependencies.

At the completion of the above steps, the application is registered with Firebase.

Application Structure

Android Studio automatically organizes the project into packages.

The following picture demonstrates the application structure of *Share What You Can*, as created by Android studio.

As shown in Figure 4.8, the application contains Gradle files (`build.gradle`) and JSON Files. Using these Gradle files, Android Studio imports all of the

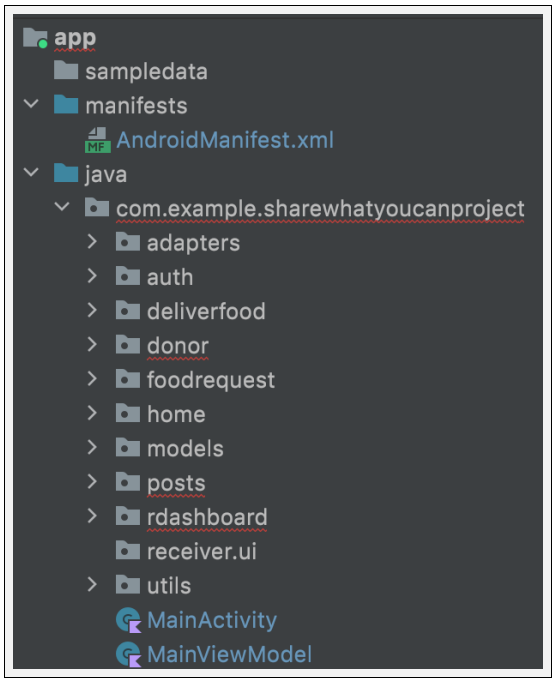


Figure 4.8: Project Code Structure

libraries required for this project.

"Example" is the package name, and *ShareWhatYouCanProject* is the name I gave to the project. To read the application's code, navigate to the following path.

`/app/src/main/java/com/example/ShareWhatYouCanProject`

I structured the entire project into the MVVM architecture. I also used adapters to fetch the data from models and display it in views.

I used the following packages in *Share What You Can*:

`adapter`

The adapter classes listed above take the data from the database (Firestore), model classes, and layouts (such as text view) and display it to the users in adapter view.

`auth`

The package *auth* contains the code related to user authentication implementation.

`utils`

The package contains classes that are used globally in the project, and this section contains extension methods, constants, and functions that are used globally.

`models`

I used three models in this application.

1. User Type: The User Type model identifies the type of user (*donor* or *receiver*).

2. Post Model: The Post model stores the data required for uploading food details.

3. Request Model: This model stores the data required for handling food requests.

I used the following packages for implementing the donor flow:

`donor`

This package contains the implementation of uploading food details.

`deliverfood`

This package contains the implementation of handling (view and approve) food requests sent by *receiver*.

`home`

This package contains implementation details of the home screen of *donor*.

`post`

This package contains the source code for displaying details of food posts uploaded by the *donor*.

I used the following packages used for implementing receiver flow:

`rdashboard`

This package contains the implementation of a view list of food posts.

`foodrequest`

Running the Application: Developers may run this application in an Emulator or in a physical device. To run the application, the *run application* button should be clicked.

Chapter 5: Unit-Testing

Software testing is one of the most crucial steps in designing a robust software system. To make sure all the individual components of the application are working properly, I performed Functionality testing to test the functionality of the application. I unit tested all of the Non-UI components of the project using the standard APIs provided by JUnit, ensuring that each of the view models passed the test before moving on because the project contains many views and view models. For testing, I first configured JUnit and then used JDK.

Chapter 6: Results

In this section, I discuss the application screens. Following are the screenshots of *Share What You Can*.

Choose the user role:

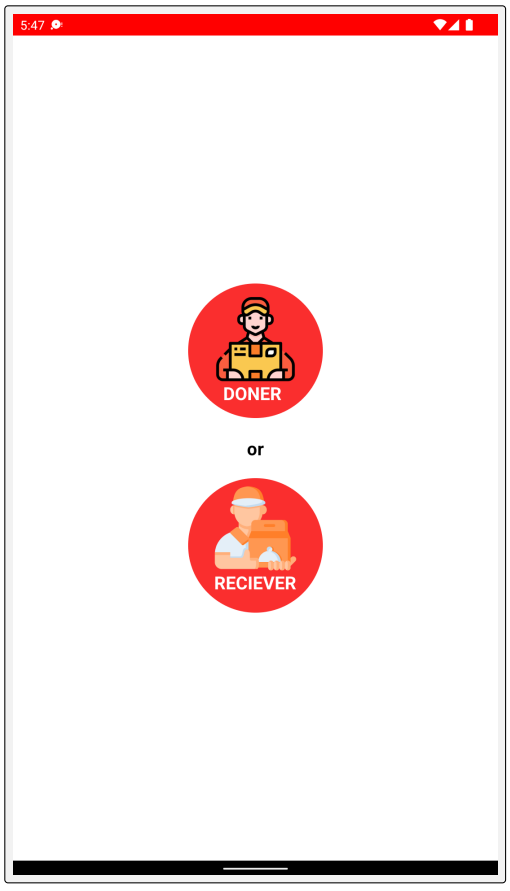
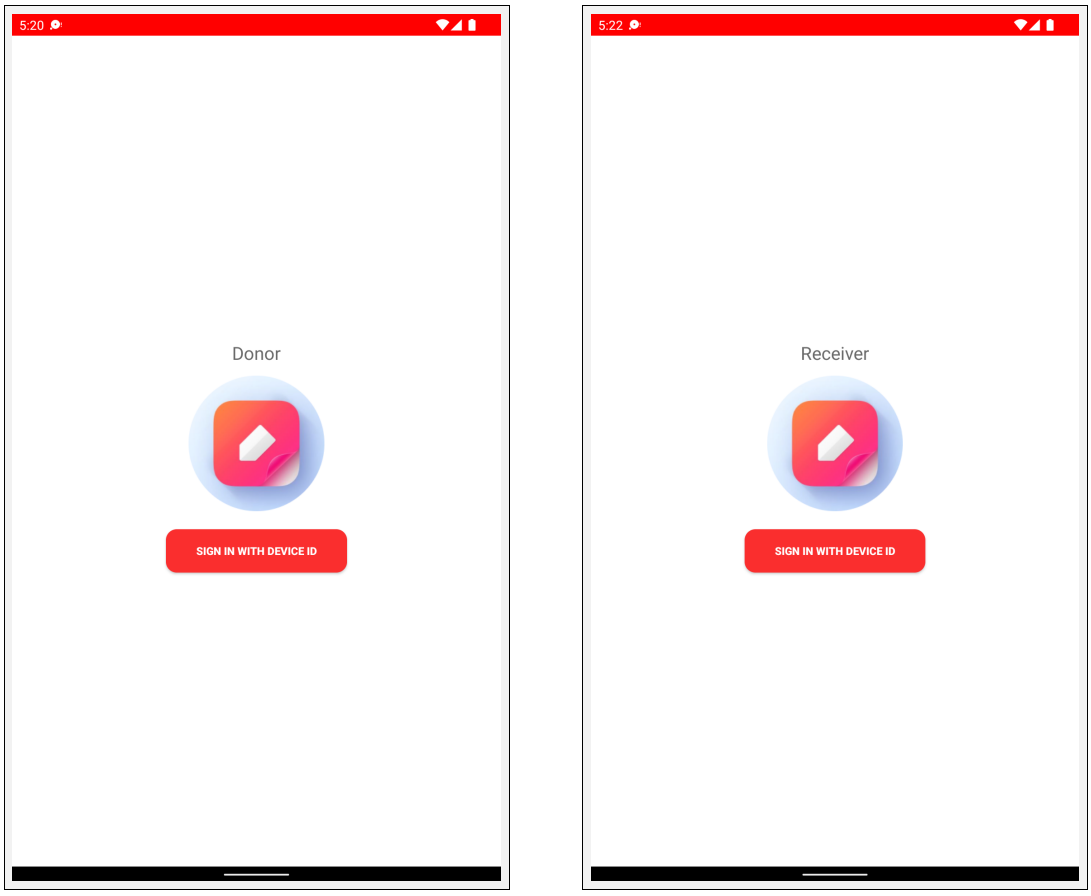


Figure 6.1: Select user type



(a) donor login (b) receiver login

Figure 6.2: User Authentication

On the first launch, the application prompts the user to login as the appropriate user type (either *donor* or *receiver*). Those who wish to donate food can login as *donor*, and anyone in need of food can login as a *receiver* (Figure 6.1).

Authentication:

After this, the application authenticates the user with their unique device ID. The device ID is the unique alphanumeric code generated for every phone once

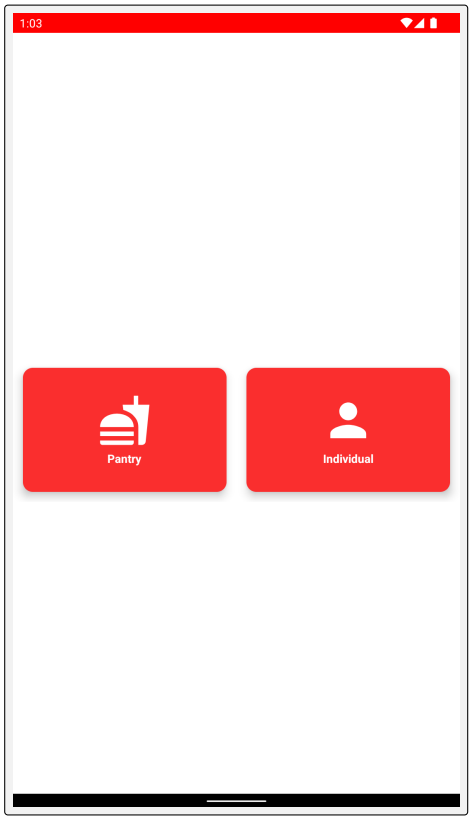


Figure 6.3: Select Donor Type

it is set up (Figure 6.2).

Select donor user type:

After the authentication process, the application gives the *donor* the option of selecting their type. Following are the categories for the donor user type.

1. Food pantries: Any nonprofit organization or welfare institution that would like to donate food can sign in as a *food pantry*.

2. Individual: A member of the general public can sign in as an *individual*

(Figure 6.3).

After the selection is made, the application stores the applicable details in Firebase and then navigates the user to the home screen, as shown in Figure 6.4.

Home Screen

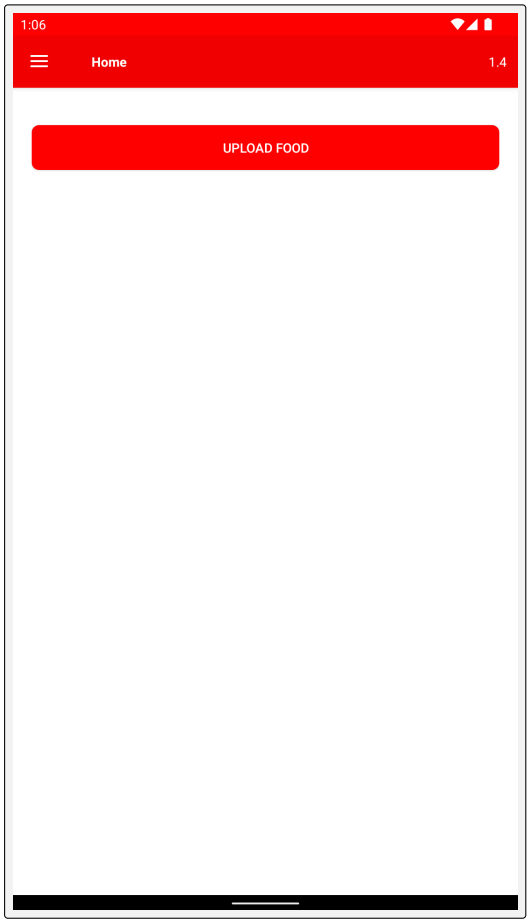


Figure 6.4: Donor Home Screen

The following order of screens displays when the application launches for the first time.

(a) Choose a user role (b) Authentication (c) Select Donor Type (d) Home Screen

If the application is already installed, then the home screen is displayed first after the launching the application. On the home screen, the application provides the *donor* with an upload food option and a drawer option.

Upload Food:

This screen allows the *donor* to donate food. Food image can be uploaded along with name and short description. The mobile application of *Share What You Can* performs the following validations of user input: missing image, empty text fields, and invalid user input in text fields.

Missing Image:

If the *donor* tries to upload the food details with a missing image, then the alert message, "Choose an image" is displayed.

Missing field/fields:

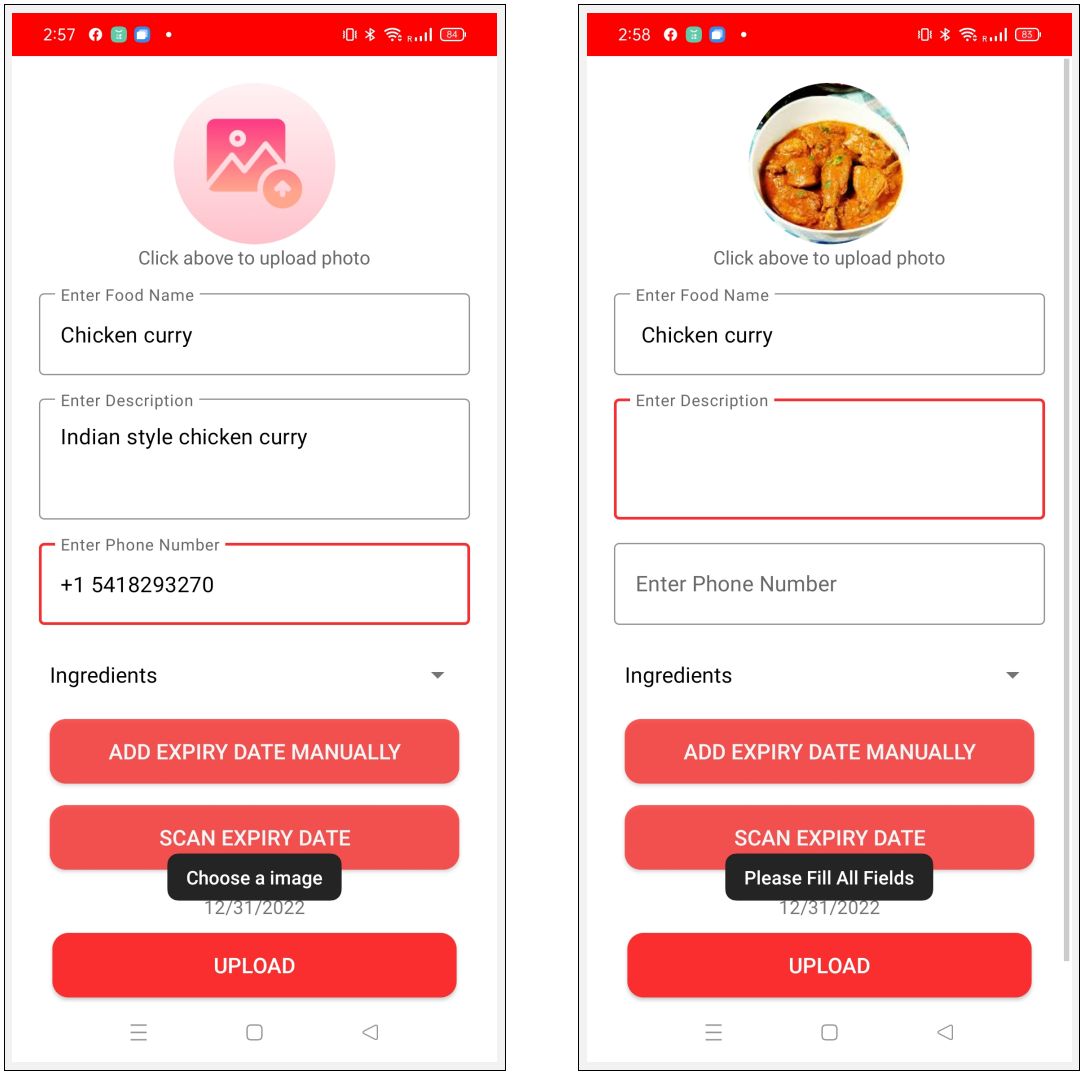
If the donation form is submitted with missing fields (food name, description, cooked hours, and/or food type), then the application displays the alert message, "Please fill all fields."

Invalid user input text fields:

If the *donor* attempts to upload details for expired food or food that has been cooked too long ago, then the alert message, "Food can't be donated" is displayed.

Drawer:

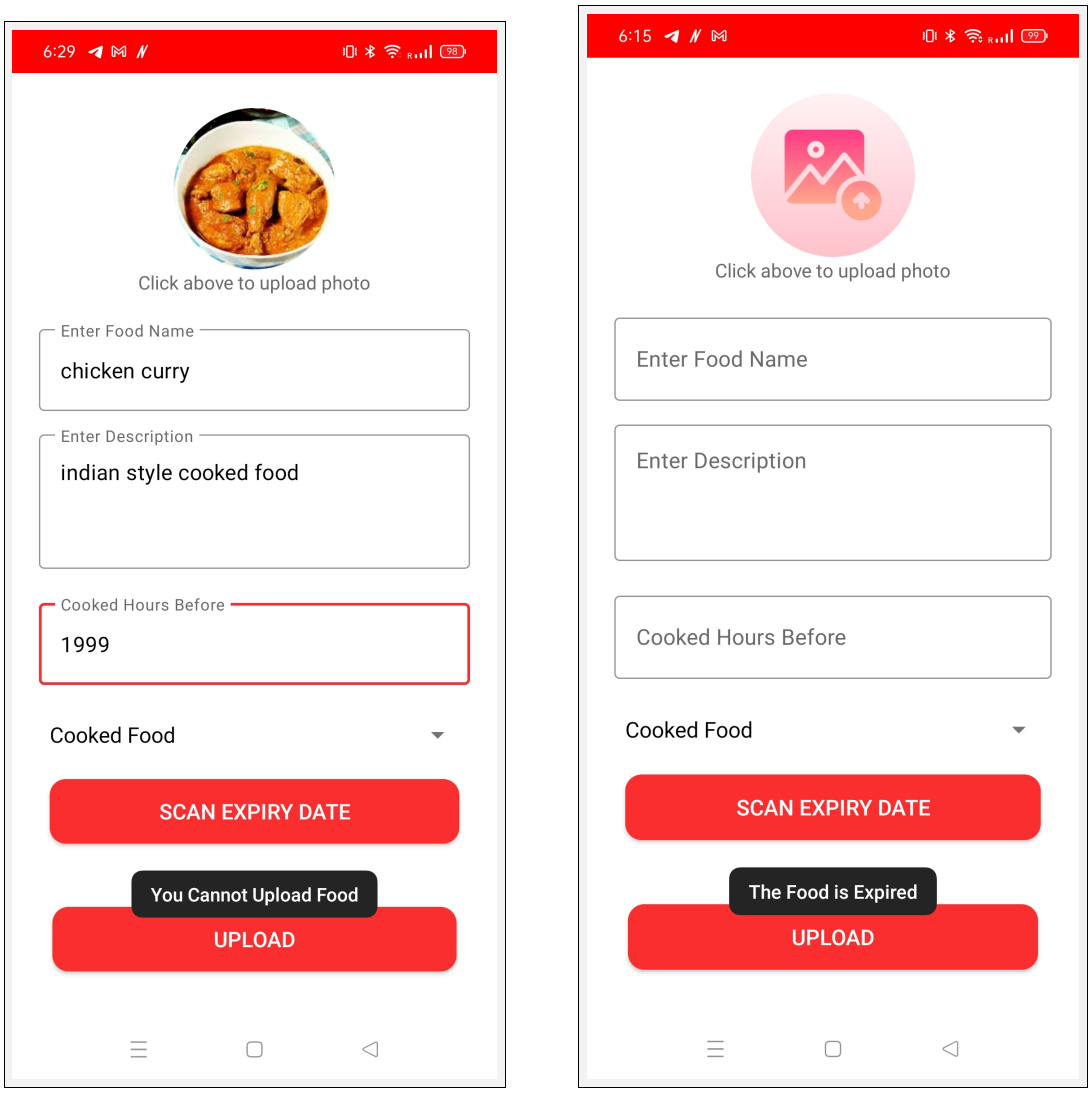
The application navigates the drawer to three screens (home screen, view food requests, and logout), as shown in Figure 6.7.



(a) Choose image

(b) Fill all fields

Figure 6.5: Validations for uploading food details



(a) Cant upload food

(b) Expired food

Figure 6.6: Validations for uploading food details

Food Request:

On the home screen, the *donor* can also view the requests made for food that they have listed. These requests are listed in a recycler view with the name of the food requester and the approve button.

Donation confirmation:

Before the food is available for donation, a confirmation dialogue box is displayed, as shown in Figure 6.7.

Food donated:

After the food is donated, the post is removed from the receiver's dashboard.

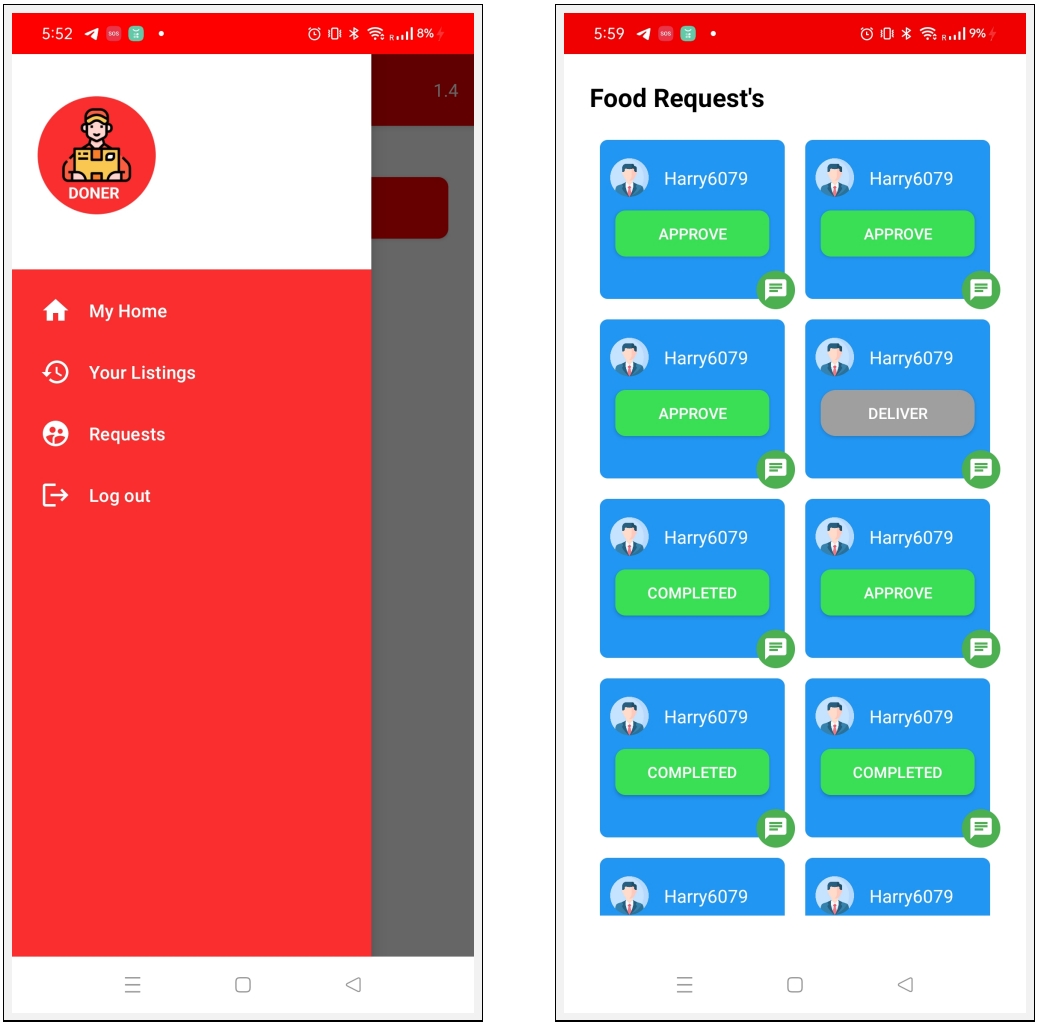
Logout:

When the user clicks the logout button, the application navigates them back to the select user type screen.

Receiver Dashboard:

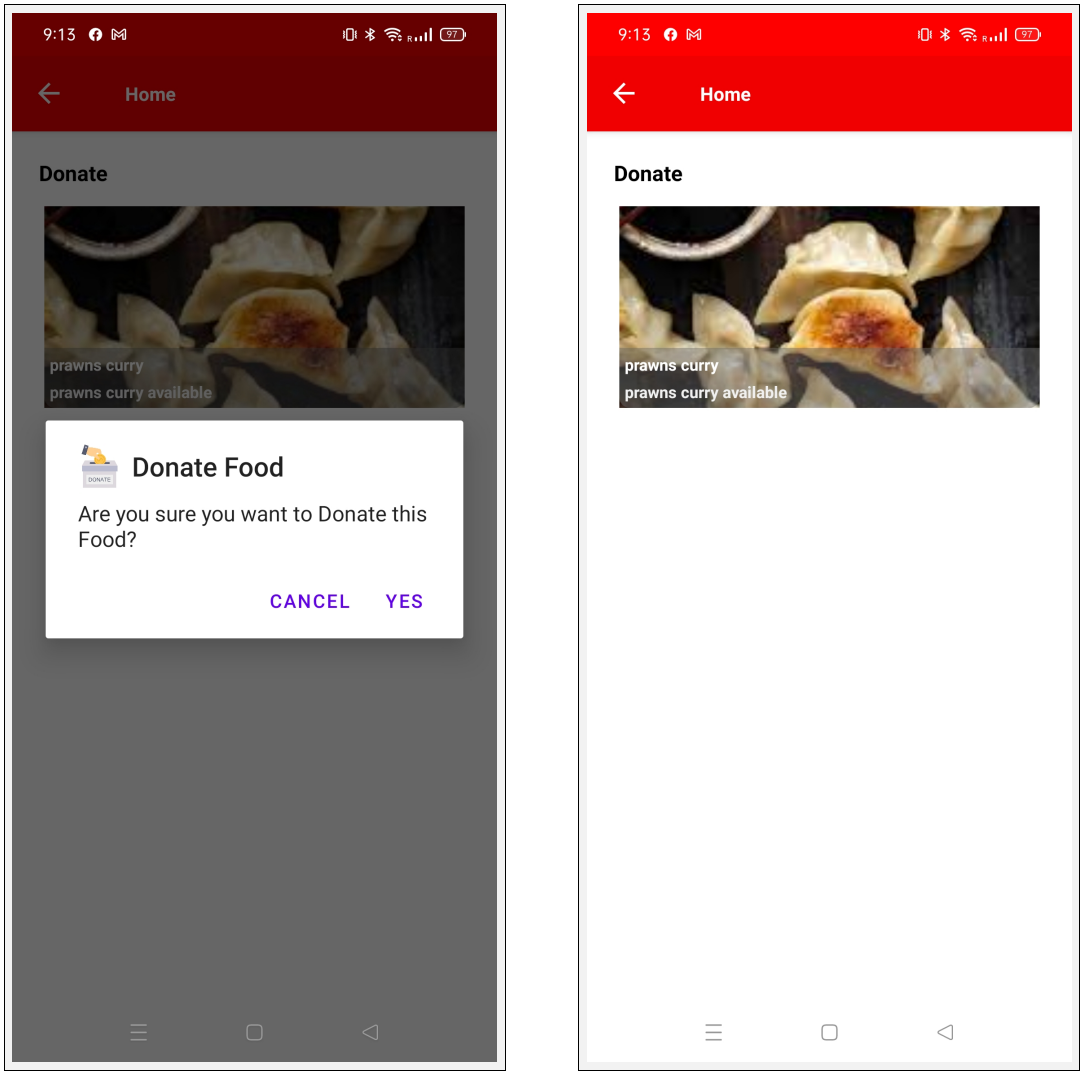
If the user is already logged in, then the above screen(Figure 6.9) becomes the starting point of the application. On the dashboard, the *receiver* will be able to view the list of food donation posts added by the *donor*. Each of these food donation posts include detailed information about the food item, such its name, an image, and a description of the item. Each time a food item is posted by a *donor*, the donation list automatically updates.

Generally, posts appear in descending chronological order, with the most recent post at the top of the list. Whenever a *donor* donates an item from the list, that item is automatically removed from the list of available food items. Clicking on a specific food item from the list results in navigating the *receiver* to the food details



(a) Donor drawer (b) View food requests

Figure 6.7: View food requests



(a) Donation confirmation (b) Donated food

Figure 6.8: Approve food requests

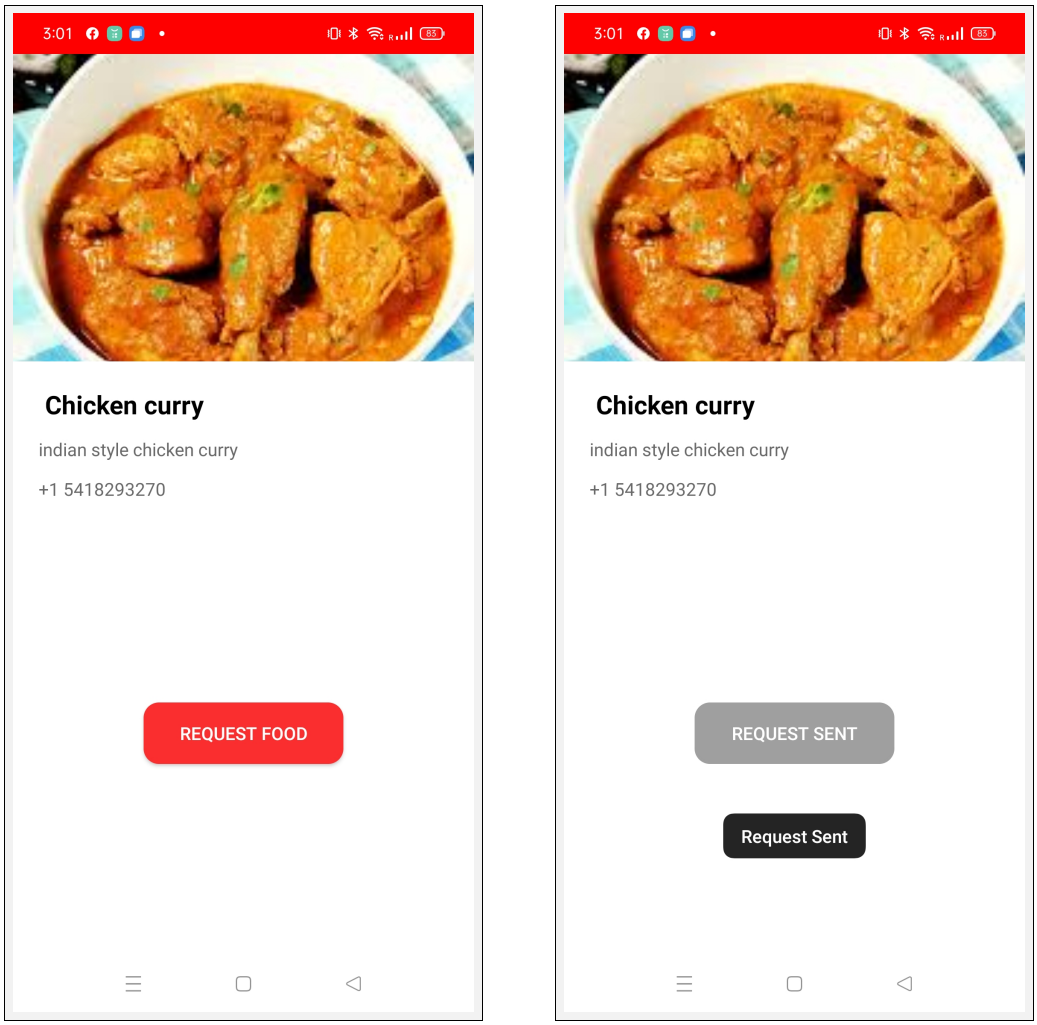
Figure 6.9: Food Donation posts

screen.

Food details:

The user can view the individual food item separately on this screen, which provides detailed information about the item, including its name, a photo, and a description. (Figure 6.10).

On click of the request food button, the *receiver* can request food for donation. Once this button is clicked, the application displays the toast message, "Request sent" to the *receiver*. The "Request Food" label is changed to "Already requested" when the *receiver* attempts to revisit the detailed view of the same food post (Figure 6.10).



(a) Food details (b) Food request sent

Figure 6.10: Sending food request

Chapter 7: Limitations

Share What You Can is an Android-compatible application and, therefore, is only accessible to users with Android smartphones; iOS users will not be able to use it. Second, users need to take into account some serious privacy concerns when utilizing this application. More specifically, fraudulent activities can occur because of the application's lack of security. One of the potential outcomes of this could be the misutilization of the personal details that are visible to the recipients on the donation posts. Adding a phone number to the food donation post increases the possibility of receiving repeated phone calls and requests for food from fraudsters, expanding the scope of internet scams. Furthermore, fraudulent users may use the provided phone numbers to obtain additional information about *donors*, including names, addresses, and other identifying details.

Chapter 8: Conclusion

A massive amount of food is wasted on a daily basis while millions of people suffer from malnutrition. With this in mind, I developed the Android application *Share What You Can* to provide a platform for both individuals and NGOs to donate food. People with surplus food can donate their food, and those in need can request the food from the food *donors*. This is significant because very few applications in the Play Store allow both food pantries and individuals to provide food. It is important to note that this application track the expiration date of food items once they have been uploaded; as a result, if a *receiver* doesn't claim an uploaded food item within a short period of time, it may expire. The expired food is automatically removed from the food donation posts.

Chapter 9: Future Scope

This application meets the basic requirement of assisting with the donation of food; however, there are some ways in which it could be improved. To begin with, in order to enhance the level of interaction between the *donor* and *receiver*, I could add a chat functionality that allows users to discuss the mode of donation (pick up / drop) and other details. Furthermore, I could create a feature that allows the user to use a long press to switch between the roles of *donor* and *receiver*, rather than having to log out and visit the login page again (just like in Instagram). To make the notification feature, I could also update it with the following features:

- Notifications for both users about the status of the food, the request sent by the recipient, and whether the request has been approved or rejected.
- The ability for the *Receiver* to opt to receive notifications if a new food donation is added to the listings.

Rather than making *Share What You Can* exclusive to Android, I could extend it to iOS as well, allowing a wider range of users to take advantage of its features. Finally, if the user were able to scan a QR code to provide the details of food, instead of manually typing them, it could save them considerable time. Adding this feature seems plausible since some food beverage packaging companies, including Nestle, Coca-Cola, Top Fruit, and Nestle Waters North America, are already using

it (Rotsios et al., 2022). By utilizing QR codes, these companies are able to determine the origin of the product.

References

Amarri, A., Corbi, A., Randall Smith, A., & Wu, A. (2015). *Feed children in need — ShareTheMeal — Charity donate.* Retrieved 2022-11-23, from https://sharethemeal.org?hl=en-US

Buzby, J. C., Farah-Wells, H., & Hyman, J. (2014). The Estimated Amount, Value, and Calories of Postharvest Food Losses at the Retail and Consumer Levels in the United States. *SSRN Electronic Journal*. Retrieved 2022-12-12, from http://www.ssrn.com/abstract=2501659 doi: 10.2139/ssrn.2501659

Chen, C. R., & Chen, R. J. C. (2019). Using Two Government Food Waste Recognition Programs to Understand Current Reducing Food Loss and Waste Activities in the U.S. *Sustainability*, *10*(8), 2760. Retrieved 2022-12-14, from https://www.mdpi.com/2071-1050/10/8/2760 doi: 10.3390/su10082760

Claire, B.-F., Byrdak, M., Fitzgerald, K., Hall, V., & Bernard, R. (2017). *MealConnect.* Retrieved 2022-11-30, from https://mealconnect.org/

Francis, P. (2013). *Pope Francis Quote.* Retrieved 2022-11-11, from https://www.azquotes.com/quote/1241311

Griffin, M., Sobal, J., & Lyson, T. A. (2022). An analysis of a community food waste stream. *Agriculture and Human Values*, *26*(1), 67–81. Retrieved from https://doi.org/10.1007/s10460-008-9178-1 doi: 10.1007/s10460-008-9178-1

Hall, K. D., Guo, J., Dore, M., & Chow, C. C. (2009). The Progressive Increase of Food Waste in America and Its Environmental Impact. *PLoS ONE*, *4*(11), e7940. Retrieved 2022-12-12, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2775916/ doi: 10.1371/journal.pone.0007940

Iaeme, I. (2013). ANDROID Vs iOS – AN ANALYSIS. *International journal of Computer Engineering & Technology (IJCET)*, *4*(1). Retrieved 2022-12-13, from https://www.academia.edu/2958151/ANDROID_Vs_iOS_AN_ANALYSIS

LiveData overview. (2022). Retrieved 2022-11-11, from https://developer.android.com/topic/libraries/architecture/livedata

Nunley, M. (2013). From Farm to Fork to Landfill: Food Waste and Consumption in America. *Pitzer Senior Theses*, *37*. Retrieved from https://scholarship.claremont.edu/pitzer_theses/37

Rotsios, K., Konstantoglou, A., Folinas, D., Fotiadis, T., Hatzithomas, L., & Boutsouki, C. (2022). Evaluating the Use of QR Codes on Food Products. *Sustainability*, *14*(8), 4437. Retrieved 2022-12-12, from https://www.mdpi

.com/2071-1050/14/8/4437 doi: 10.3390/su14084437

Schanes, K., Dobernig, K., & Gözet, B. (2018). Food waste matters - A systematic review of household food waste practices and their policy implications. *Journal of Cleaner Production*, *182*, 978–991. Retrieved 2022-12-14, from https://linkinghub.elsevier.com/retrieve/pii/S0959652618303366 doi: 10.1016/j.jclepro.2018.02.030

Schill, A. (2022). *Careit App: Donate Food, Feed People.* Retrieved 2022-11-23, from https://www.indiegogo.com/projects/2680406

U.S. Food Loss and Waste 2030 Champions. (2017). Retrieved 2022-12-13, from https://www.usda.gov/foodlossandwaste/champions