Home Food Delivery App Report

Prepared by Group 27:
Christian Gutierrez, Jordan Nguyen, Natalie Reyes, Wasay Ahmed

CS 440
University of Illinois Chicago
Spring 2022

Table of Contents

List of Figures

List of Tables

I Project Description

1 Project Overview

The Home Food Delivery application provides users with the ability to order home cooked meals directly to your home. To make this possible there are three users that are crucial to this process, the cook, the customer, and the driver. The cook will be preparing the ordered meals within their own kitchen that the customer has placed. The driver will then receive an order on their end so they can deliver the meal to the proper destination.

2 Project Domain

The application is used to connect three users in making, preparing, and delivering an order. We have all user and order information entered into the firebase so it can be retrieved and used when logging in and retrieving orders. Android studio provides its own way of testing each activity in the developers documentation under test your activities which may differ from how traditional tests are run and created.

3 Relationship to Other Documents

The Home Food Delivery Scenario one gave a brief description of what each user will be capable of doing as well as what is expected to be delivered by the time we have the first release.

Ahmed, Wasay, Reyes, Natalie , Gutierrez, Christian, Nguyen, Jordan et al. 2022, pp. 1–2, *Home Delivery App Scenario*.

The second release of the application is based on the second scenario that was written for the Home Food Delivery App.

Ahmed, Wasay, Reyes, Natalie , Gutierrez, Christian, Nguyen, Jordan et al. 2022, pp. 1–2, *Home Delivery App Scenario 2*.

The original concept creators and report published for the Home Food Delivery application.

Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 1–70, *Home Food Delivery App*.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Activity: One screen of the UI that android studio can create and edit to make a login or profile for example. “Activity” is not used to describe what someone does in the context of this project.

Application: Downloaded and installed onto a mobile device by the user to perform specific tasks as it is a small specialized program. Does not refer to form or request.

App: abbreviation of the word application.

UI: abbreviation of “User Interface”

4b UML and Other Notation Used in This Document

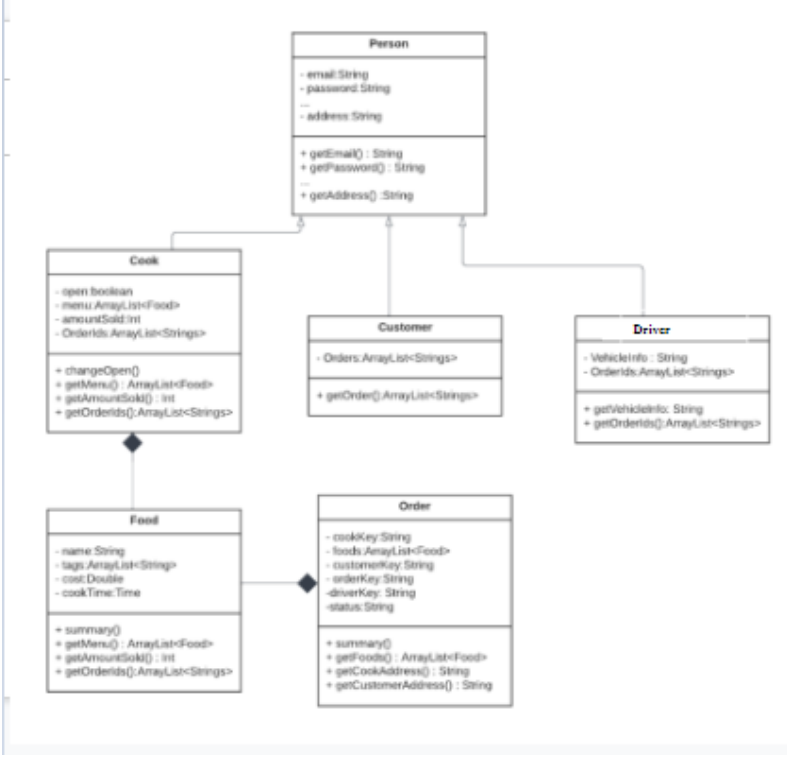

Figure 1. *Home Food Delivery App UML*

Each square represents classes that were used within the project: Person, Cook, Customer, Driver, Food, Order.
Below each class name is a description of what information was needed to create them. Pointed arrows within the diagram represent where all the three users were derived from which is the Person class in the middle where all three arrows point directly to. The addition symbol within each square represents the getters and shows what method was used to extract that user information, example: "+getOrder():ArrayList<Strings> in the Customer class. The filled in diamond symbol represents composition. This shows a dependent relationship as the Food class for example cannot exist without the cook class. The concept when looking at the Order and Food class, an order cannot exist alone as it depends on the Food class therefore the filled in diamond symbol is placed.

4c Data Dictionary for Any Included Models

Person = email + password + address +{get Email + get Password + get Address}
Retrieving personal information about each person/ each user.

Cook = menu:ArrayList<Food> + {get Menu + get AmountSold +get OrderIds}
List of food items given on the cooks personalized menus and when choosing stores the amount in dollars and provides an id to each order.
Customer = Orders Array List + {get Order}
Sees the orders in a list that have been made and when retrieved it is viewable by the customer user.

Driver = vehicle information + {get vehicle info + get order Ids}
Inserts important vehicle information in order to be eligible to be a driver, when the driver finds an order and accepts the order is given an id.

Food = name + tags + cost + cook time + {get menu + get amount sold + get order ids}
Each food item get a specific field that needs to be filled and used as data in the database. Fields include tags such as listing ingredient or allergy notification, cost, and how long it will take to prepare. Once a food item is chosen you can choose how much of that food you would like and add it to the order which the database then handles and adds to the whole order.

Order = cook key + customer key + driver key + status + {get foods + get cook address + get customer address}
Each user has a key that is used, in order to receive an order the status must represent that which is manually done by the cook and driver. This enters the database and the order is sent to the next user to continue the process of placing and delivering an order.

Food order:
• Pickup date and time
• Pickup location
• Delivery date and time
• Delivery time
• Driver name
• Drivers contact information
• HomeCook name
• HomeCook contact information
• HomeCook address
• Customer Name
• Customer contact information
• Items ordered
• Invoice number
• Amount charged for the items
• Service feed
• Tip to the driver

Driver Vehicle:
• Car model and year
• Owner name
• Insurance policy number

Cooked food instruction:
• Food ingredients
• Any allergies that the customer has mentioned
• Calories

• Date and time of cooking
• Duration to prepare the meal
• Price of the item
• Recipe provided by the HomeCook

[14] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 20, 7C Data Dictionary, *Home Food Delivery App*

II Project Deliverables

Our group has produced the Home Food Delivery Application which has been documented and released in two parts, release 1 and release 2. Throughout each release we have made a number of improvements to each user as well as the database that was made. Every aspect of functionality our group completed at the time has been documented under each release.

5 First Release

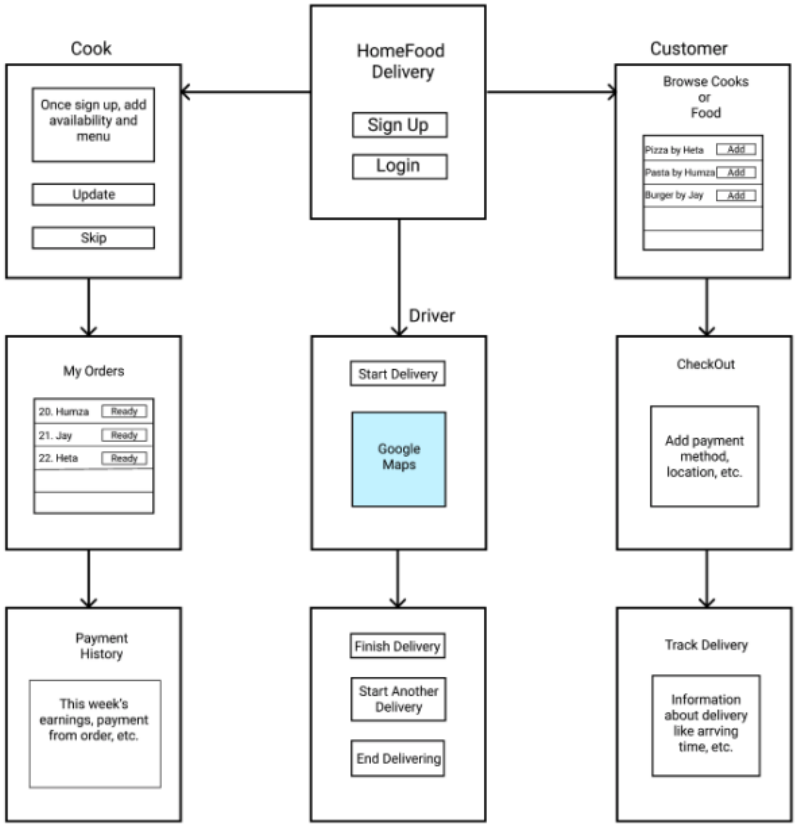

***Figure 2**: User Interface Diagram for Home Food Delivery App [7] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 55, 25F.*

The first release of the Home Food Delivery application focused on the functionality of the Login and the Cook user. Our team had created the cook object classes within the IDE that had been chosen which was Android Studio. Initially we had released a basic mapping of

the app that included all 3 profiles (cook, customer, and driver) and buttons that can go to and from each XML activity page as well as the order pages we developed.

6 Second Release

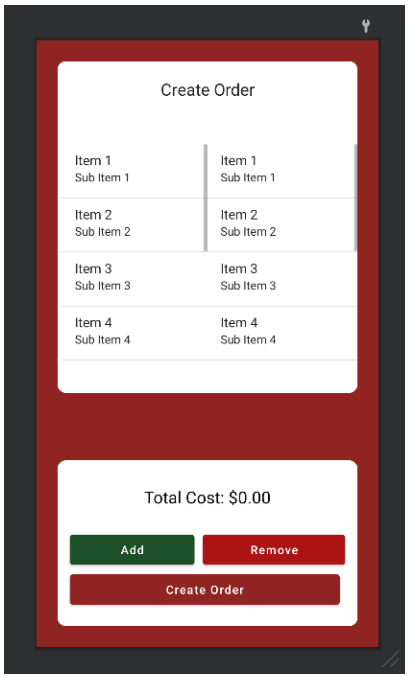

Figure 3. *CustomerViewMenu UI.*

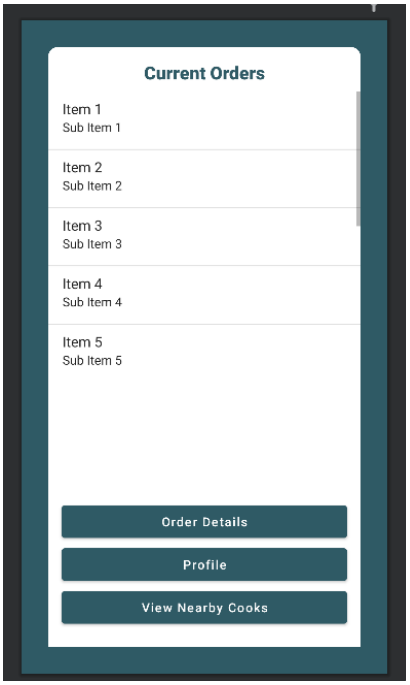

Figure 4. *CustomerMainActivity UI.*

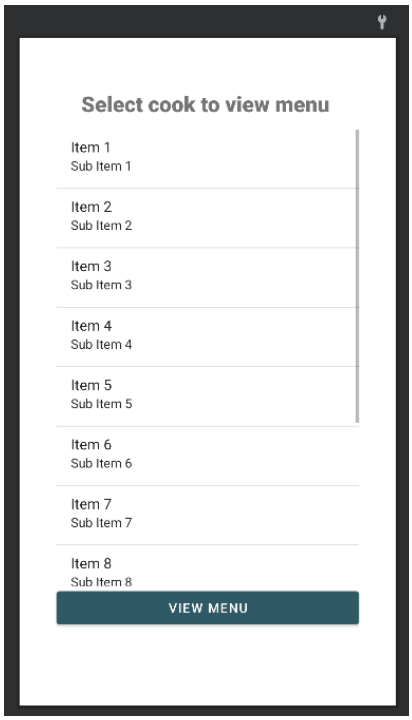

Figure 5. *CustomerViewCooksActivity UI.*

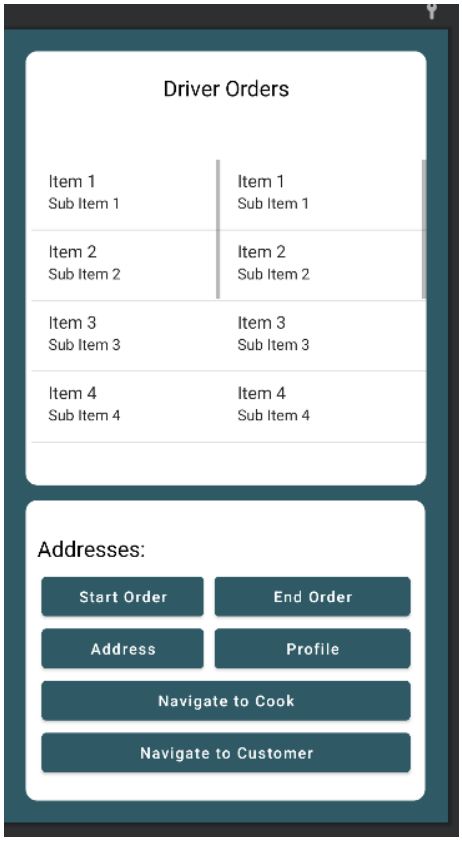

Figure 6. *DriverMainActivity UI.*

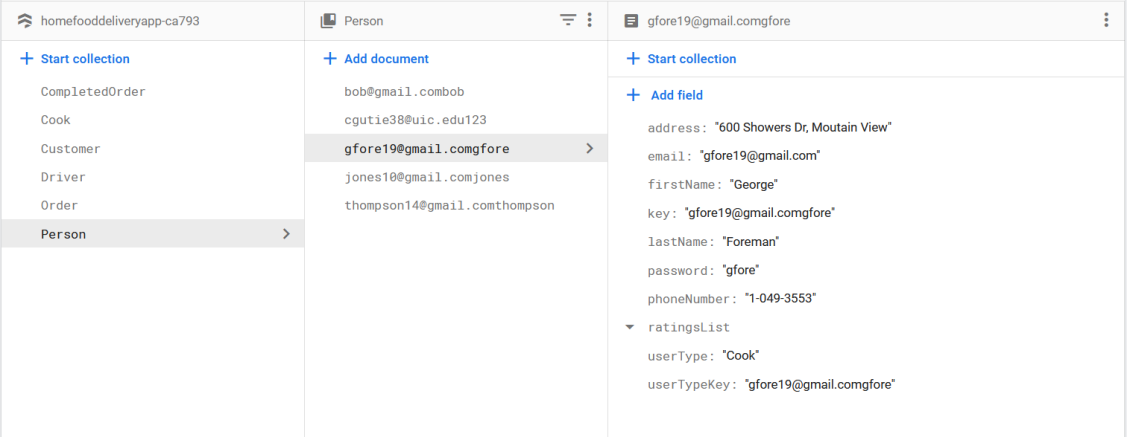

Figure 7. *Firebase Cloud Firestore Database.*

The second release of the project focused on the Customer, Driver, and database functionality. The Customer has the ability to view their order details and view nearby cooks functionality on the CustomerMain class. After viewing which cooks were available the customer is able to pull up the menu of that particular cook. This process takes place on the CustomerViewCookActivity and the CustomerViewMenuActivity. The DriverMain activity now works in real time, functionality of DriverMaps activity to certain locations works, the ability to start and end an order also has functionality as well.

7 Comparison with Original Project Design Document

There were numerous similarities and differences throughout our project compared to the original project. We have documented these findings in the table below along with references as to where these can be found in the original documentation by group 26. Figure 8 shows the current activity diagram for our Home Food Delivery app.

Similarities	Differences
• Customer Rates Cooks **[12] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 16, 6B Implementation Environment of Current Systems, *Home Food Delivery App*** • Google Map API **[13] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 16, 6C Partner or Collaborative Applications, *Home***	• Messaging capability between users • Realtime messaging between drivers, cooks, and customers. • Weather info for drivers **[15] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 15, 6 Mandated Constraints, *Home Food Delivery App*** • Cooked food recipe provided by homecook

Food Delivery App ● Login API **[14] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 20, 7C Data Dictionary, *Home Food Delivery App*** ● Updates database of user information ● Placed orders update in the database	● Food calories ● Detailed food ingredients **[16] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 15, 7C Data Dictionary, *Home Food Delivery App***

Table 1. *Comparison table*

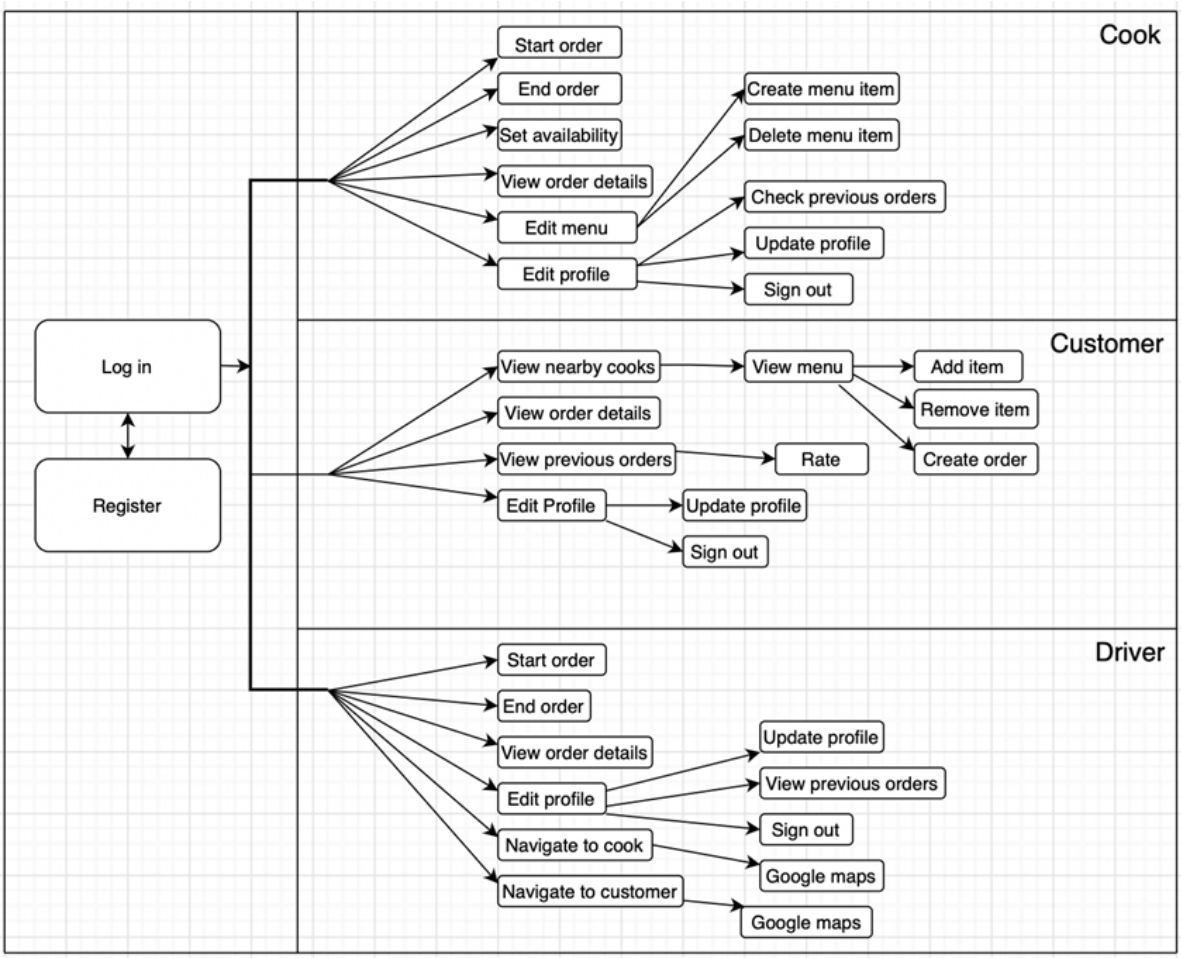

Figure 8. *Activity Diagram for Home Food Delivery App.*

III Testing

8 Items to be Tested

DriverMainActivity - Christian
LoginActivity - Natalie
CustomerViewMenu -Wasay
CookMain - Jordan

9 Test Specifications

ID1 - TestEditMenu (CookMainActivityTest)

Description: Test if edit menu button transition to cook edit menu activity
Items covered by this test: edit menu button functionality
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: No Outputs
Pass/Fail Criteria: Pass test if the activity switches to correct activity

ID2 - TestCookProfileEvent (CookMainActivityTest)

Description: Test if profile button transition to cook edit profile activity
Items covered by this test: profile button functionality
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: No Outputs
Pass/Fail Criteria: Pass test if the activity switches to correct activity

ID3 - TestStartOrderEvent(CookMainActivityTest)

Description: Test if orders can be started and reflected in database
Items covered by this test: start button,order list functionality and changestatus() in order
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library, firebase database
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: "START BUTTON FAILED" output if failed
Pass/Fail Criteria: Pass test order status changes to "accepted_cook" in database

ID4 - TestEndtOrderEvent(CookMainActivityTest)

Description: Test if orders can be ended and reflected in database
Items covered by this test: end button,order list functionality and changestatus() in order

Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library, firebase database
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: "END BUTTON FAILED" output if failed
Pass/Fail Criteria: Pass test order status changes to "finished_cook" in database

ID5 - TestToggleButtonEvent(CookMainActivityTest)
Description: Test if availability button updates self and database
Items covered by this test: availability button
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: "changing availability failed" if failed
Pass/Fail Criteria: Pass test if the availability changes in database and button

ID6 - TestCookOrderDetails(CookMainActivityTest)
Description: Tests that nothing happens if button pressed nothing selected
Items covered by this test: order details,order list button functionality
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: No Outputs
Pass/Fail Criteria: Pass test activity stays the same

ID7 - TestCookOrderDetails1(CookMainActivityTest)
Description: Test to go to order activity depending on selected order
Items covered by this test: order details,order list button functionality
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library
Intercase Dependencies: NA
Test Procedures: Run test script
Input Specification: No Inputs
Output Specifications: No Outputs
Pass/Fail Criteria: Pass test if the activity switches to order activity

ID8 - TestValidLogin (LoginActivityTest)
Description: Test and verify if a user can login to the app with valid credentials when signing in
Items covered by this test: sign in button functionality with account
Environmental needs:android studio

Intercase Dependencies: na
Test Procedures: Input credentials an account already made to see if sign in is eligible
Input Specification: input valid email and password or a user
Output Specifications: none
Pass/Fail Criteria: Pass, account is registered it allows sign in

ID9 - TestInvalid login (LoginActivityTest)
Description: Test if an account that does not exist is able to sign in
Items covered by this test:login unregistered account
Environmental needs:android studio
Intercase Dependencies: na
Test Procedures: input email and password that has never registered an account
Input Specification: put incorrect email and password
Output Specifications: none
Pass/Fail Criteria: Pass, user must register before logging in

ID10 - testProfileEvent (DriverMainTest)
Description: Test Profile button opens to DriverProfileActivity.
Items covered by this test: profileButton button functionality and database calls.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Activity layout change to DriverProfileActivity.
Pass/Fail Criteria: Pass test if the activity switches to correct activity.

ID11 - testDriverOrderDetails1 (DriverMainTest)
Description: Test Order Details button opens to DriverOrderDetailsActivity.
Items covered by this test: Order Details button functionality and database calls. Test if activity fails to open because order from list was not selected.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Activity layout does not change to DriverOrderDetailsActivity.
Pass/Fail Criteria: Pass test if the activity does not switch.

ID12 - testDriverOrderDetails2 (DriverMainTest)
Description: Test Order Details button opens to DriverOrderDetailsActivity.
Items covered by this test: Order Details button functionality and database calls.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA

Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Activity layout changes to DriverOrderDetailsActivity.
Pass/Fail Criteria: Pass test if the activity switches to correct activity.

ID13 - testStartOrderEvent (DriverMainTest)

Description: Test Start Order button changes order status.
Items covered by this test: Start Order button functionality and database calls.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Order moves to second list with status change.
Pass/Fail Criteria: Pass test if the order switches to the second list with status change and addresses.

ID14 - testEndOrderEvent (DriverMainTest)

Description: Test End Order button changes order status.
Items covered by this test: End Order button functionality and database calls.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Order moves to second list with status change and then removes order with final status change.
Pass/Fail Criteria: Pass test if order is removed and check if moved to CompletedOrders collection.

ID15 - testNavigate (DriverMainTest)

Description: Test Navigate buttons open to DriverMapsActivity and Google Maps.
Items covered by this test: Navigate buttons functionality.
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run DriverMainTest.
Input Specification: No Inputs.
Output Specifications: Activity layout change to DriverMapsActivity.
Pass/Fail Criteria: Pass test if the activity switches to correct activity and opens Google Maps.

ID16 - TestAdd (CustomerViewMenuTest)

Description: Test if a customer is able to add a menu item to the list of orders from the cook's menu.
Items covered by this test: add button functionality.
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library.

Intercase Dependencies: NA
Test Procedures: Run CustomerViewMenuTest.
Input Specification: No Inputs.
Output Specifications: No Outputs.
Pass/Fail Criteria: Pass test if the menu items from the cooks menu is added to the customers order list.

ID17 - TestRemove (CustomerViewMenuTest)
Description: Test if a customer is able to remove a menu item from the list of orders.
Items covered by this test: remove button functionality.
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: RunCustomerViewMenuTest.
Input Specification: No Inputs.
Output Specifications: No Outputs.
Pass/Fail Criteria: Pass test if item is removed from list of orders.

ID18 - TestCreateOrder (CustomerViewMenuTest)
Description: Test if order can be placed from list of orders.
Items covered by this test: create order button functionality.
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run CustomerViewMenuTest.
Input Specification: No Inputs.
Output Specifications: No Outputs.
Pass/Fail Criteria: Pass test if order can be placed from list of orders.

ID19 - TestBack (CustomerViewMenuTest)
Description: Test the back button can bring the user to the previous page.
Items covered by this test: back button functionality.
Requirements addressed by this test: NA
Environmental needs: jUnit4, espresso library.
Intercase Dependencies: NA
Test Procedures: Run CustomerViewMenuTest.
Input Specification: No Inputs.
Output Specifications: No Outputs.
Pass/Fail Criteria: Pass test if the activity switches to previous activity.

10 Test Results

ID1 - TestEditMenu (CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: activity change.

Actual Results: Activity did change.
Test Status: Pass

ID2 - TestCookProfileEvent (CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: activity change.
Actual Results: Activity did change.
Test Status: Pass

ID3 - TestStartOrderEvent(CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: order update status on database
Actual Results: order update status on database
Test Status: Pass

ID4 - TestEndOrderEvent(CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: order update status on database
Actual Results: order update status on database
Test Status: Pass

ID5 - TestToggleButtonEvent(CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: available update status on database
Actual Results: available update status on database
Test Status: Pass

ID6 - TestCookOrderDetails(CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: no activity change.
Actual Results: no activity change.
Test Status: Pass

ID7 - TestCookOrderDetails1(CookMainActivityTest)
Date of Execution: April 22, 2022
Staff conducting tests: Jordan Nguyen
Expected Results: activity change.
Actual Results: Activity change.
Test Status: Pass

ID10 - testProfileEvent (DriverMainTest):

Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: Activity change to DriverProfileActivity.
Actual Results: Activity changed to DriverProfileActivity.
Test Status: Pass

ID11 - testDriverOrderDetails1 (DriverMainTest)
Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: No activity change.
Actual Results: Activity did not change.
Test Status: Pass

ID12 - testDriverOrderDetails2 (DriverMainTest)
Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: Activity change to DriverOrderDetailsActivity.
Actual Results: Activity changed to DriverOrderDetailsActivity with the correct order data.
Test Status: Pass

ID13 - testStartOrderEvent (DriverMainTest)
Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: Order selected then moved to second list with status change.
Actual Results: Order moved as expected.
Test Status: Pass

ID14 - testEndOrderEvent (DriverMainTest)
Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: Order selected then moved to second list with status change then removed from list. Check order copy in CompletedOrders collection.
Actual Results: Order found in CompletedOrders collection and removed from list.
Test Status: Pass

ID15 - testNavigate (DriverMainTest)
Date of Execution: April 23, 2022
Staff conducting tests: Christian Gutierrez
Expected Results: Activity changes to DriverMapsActivity and opens Google Maps.
Actual Results: Activity changed to DriverMapsActivity and opened Google Maps.
Test Status: Pass

ID16 - TestAdd (CustomerViewMenuTest)

Date of Execution: April 23, 2022
Staff conducting tests: Wasay Ahmed
Expected Results: List of orders has order added from cooks menu.
Actual Results: Order added.
Test Status: Pass

ID17 - TestRemove (CustomerViewMenuTest)
Date of Execution: April 23, 2022
Staff conducting tests: Wasay Ahmed
Expected Results: List of orders has selected order removed.
Actual Results: Order removed.
Test Status: Pass

ID18 - TestCreateOrder (CustomerViewMenuTest)
Date of Execution: April 23, 2022
Staff conducting tests: Wasay Ahmed
Expected Results: Order is placed.
Actual Results: Order did not get placed.
Test Status: Fail

ID19 - TestBack (CustomerViewMenuTest)
Date of Execution: April 23, 2022
Staff conducting tests: Wasay Ahmed
Expected Results: Takes back to the previous page.
Actual Results: Brought to previous page.
Test Status: Pass

11 Regression Testing
NA.

IV Inspection
Login Activity (Christian)
Register (Natalie)
Cook Main (Jordan)
Driver Main/DriverMaps (Wasay)

12 Items to be Inspected
Login Activity by Christian Gutierrez:
- Sign in and database connection for cook, customer, and driver
- New intent is successfully created for cook, customer, and driver
- Button/functionality for sign in
- Button/functionality for registering an account
- Error checking for existence of an account
- Error checking for unfilled fields

Register Activity by Natalie Reyes:

- Account is successfully registered for cook, customer, and driver
- Account information is successfully stored in database for cook, customer, and driver
- Register button/functionality for cook, customer, and driver
- New intent is successfully created for cook, customer, and driver
- Error checking for leaving fields unfilled
- Checking validity of fields filled (such as valid phone number, email, address, etc…)
- Being able to go back

Cook Main Activity by Jordan Nguyen:

- Being able to view pending orders
- Being able to select and unselect multiple pending orders
- Start order button/functionality with multiple orders
- End order button/functionality with multiple orders
- Availability button/functionality
- Order details button/functionality
- Edit menu button/functionality
- Profile button/functionality
- Activity changes after pressing order details, edit menu, and profile button
- Availability being updated in database for user
- Being able to go back

Driver Main Activity/Driver Maps Activity by Wasay Ahmed:

- Being able to view orders
- Address retrieval
- Launching map intent
- Start order button/functionality with multiple orders
- End order button/functionality with multiple orders
- Order details button/functionality
- Being able to set availability
- Button to navigate to cook
- Button to navigate to customer
- Profile button/functionality
- Navigation works from current location to the desired location
- Being able to go back

13 Inspection Procedures

Checklist: https://research.cs.queensu.ca/home/elec372/as2/Checklist.html by research.cs.queensu.ca

Two meetings were held. In the first meeting, every group member discussed the piece of code they wanted the other group members to inspect. After that each member inspected the other group members' code on their own time. In the second meeting, the group members discussed what they discovered during inspection for each other's code. The meetings were held on Discord, so all group members discussed

electronically.

14 Inspection Results

Register Activity (Christian Gutierrez):
The entire registerActivity class along with its methods were inspected. The first problem I noticed was all the class variables were not declared either public or private. This would automatically make them public variables that could be accessed from outside of the class and be overwritten. Ideally, they should be declared as private variables so that only the class methods can access them. Another variable issue I noticed is docID which is an unused variable. The onCreate method only sets the layout for the activity. The EditText variables should be declared in the onCreate method by moving lines 34-39. Lines 47-51 are not necessary for the getData method to function. The setErrors method outputs an error to every EditText field in the activity when triggered by at least a single empty field. This method should be fixed and only output empty errors to the empty fields. The getData does not check for other besides empty EditText fields. It should check if the data submitted is valid.

Cook Main (Christian Gutierrez):
The entire CookMainActivity was inspected. All the class variables are public and should be made private. Line 76 is not necessary, just a simple cook object print method. The list view items should have a nicer look, maybe changing the strings used for the items. Order keys shouldn't be displayed in the list view. Line 99 else statement is not necessary.

DriverMainActivity and DriverMapsActivity (Christian Gutierrez):
The entire DriverMainActivity and DriverMapsActivity were inspected. All the DriverMainActivity class variables declared are public and some are unused. Spacing between and within the methods are not consistent and makes reading a little difficult. More organization after writing should be done. Commented out lines from 207 to 218 should be deleted.
For DriverMapsActivity there are many unused libraries and unnecessary comments. Method onMapReady has no implementation but is used. The onCreate method should finish the activity after opening the emulator's Google Maps app to fix the blank screen bug when returning the Home Food Delivery App.

Login Activity (Jordan Nguyen 4/22/22)
Having the database table labeled "Person" when login is a little confusing as we also have a class labeled person which all user types inherit from. Maybe instead of a "Person" table, names like "User" would be better. Also, the if -else statements would be better if they were made into switches. Code also is too jumbled, can be broken down into multiple functions, especially each functional part. For example, checking if login is valid by not being empty can be made into a separate function. Also, it could use some comments to explain some of the functionality of the code.

Register Activity (Jordan Nguyen 4/22/22)
Missing comments documentation. Some variable names are weird, for example

boolean g is not a descriptive name as it is one letter, and leaves others guessing. I think the register should also go further and make sure inputs are valid, I don't think duplicates are handled. Maybe check the database if the user already exists before adding or it will replace the existing user and lose data.

DriverMainActivity and DriverMapsActivity (Jordan Nguyen 4/22/22)
Variables used are undescriptive, ie. button1,button2,button3… would be better to have the button have names of their actual uses. Alot of repeating code especially between if statements (driverOrdersAcceptedList.getCount() > 0) and (driverOrdersList.getCount() > 0) . Can have a lot of helper functions to make code more readable and easier to debug. In the MapsActivity, a lot of commented stuff that doesn't have a description and makes it hard to read. Unsure what is happening in this activity unless looking very closely.

Login Activity (Natalie): 4/22/22 5pm
While inspecting the login activity there were no comments in the code which may be a good practice just so it is easier to read through quickly when referring back to code. Where we included the database we should comment and mention when different functions or database come in just to distinguish between what to look for when we need to. The sign in button and registering functionality works efficiently but may have been a good idea to include a message if the user is denied login depending on if username or login was incorrect.

CookMain (Natalie): 4/22/22 5pm
After inspection of Cook Main the functionality of the buttons such as the viewing pending orders availability button, profile button, as well as the order details button all work properly. Each button is named accordingly to line 36 which makes it easier when inspecting but not many comments were included. Most of the code has remained public accept the threading done on line 222 but we would benefit by changing this as then the access to the classes will remain open to other classes not just within itself.

DriverMainActivity/MapActivity (Natalie): 4/23/22 5pm
Inspection of the Driver Main Activity has been completed as well as the Map Activity. The button event names could be more specific to what exactly or who exactly they are referring to that way it is easier at first glance to see which user you are looking at quickly. Instead of a profile button event for example we could say it's specifically for the driver, therefore it could be called driver profile event instead. Below the start and end order buttons aesthetically speaking we could takeout any code that is not in use or being used as a comment, such as line 207-218 just so we can clean up the code a bit.

Cook Main Activity (Wasay) 4/22/22:
All methods inside Cook Main Activity were inspected. An observation made is that at line 100 there is an unnecessary print statement which can be removed. Furthermore lines 228, 234, 248, and 258 can be removed because they are

commented out lines of code which are useless. Each method can use comments to explain what that method does. The names of the methods and variables are short and meaningful as they give a good idea of what they do, thus making it helpful in understanding the code. Threading is done well which can help prevent issues with multiple cooks. Loops are being used to access elements within listview and arrayLists. If there is a lot of data, this can affect efficiency, so the way the loops are used can be improved to help with efficiency.

Register Activity (Wasay) 4/22/22:
All methods inside Register Activity were inspected. An observation made is that the variable names for the text fields can be more meaningful. Furthermore lines 58-68 can be made into their own function for the database. Method names are meaningful and help figure out what each method does. Methods “registerAsCook”, “registerAsCustomer”, and “registerAsDriver” have repeating code of adding that user to the database. A function can be created for this, so code is not repeated. Lines 81, 98, and 114 have a variable called “g” which is not a meaningful variable name. Method setErrors is a nice way to go about error checking as it is organized and makes it easier to read and understand the code.

Login Activity (Wasay) 4/22/22:
All methods inside Login Activity were inspected. An observation made is that there are three main methods such as onCreate, signIn, and createAccount. This is good as it helps organize the code specifically for that functionality, however code can be broken into more functions for further organization. Lines 72-92 contains a lot of repeating code for accessing the database for that user. This can be broken into its own function. Variable names are meaningful and help understand the code.

V Recommendations and Conclusions

All of our testing passed so there are no bugs to fix, however, we noticed a lot of issues when we inspected our code. Common problems seen across all reviews were lack of documented code, bad variable names, and lack of encapsulation/helper functions leading to lengthy code which is hard to decipher. The improvements are straightforward, we should add comments to the code and choose better variable names that are more specific. To improve encapsulation and readability, we should create helper functions of repeated code so the logic is easier to understand.

VI Project Issues

15 Open Issues

The main open issue for this project are the legal questions for the cooks and the overall business. Since the cooks will most likely be considered as their own business making food in their home kitchens, they will be subject to their state’s Cottage Food Laws. These laws set regulations and permits that must be followed before operating and limit the types of food allowed to be sold. For example, the state of Illinois does not allow home cooks to sell any product that contains meat [10]. Therefore, the Home Food Delivery App business must research the Cottage Food Laws of the specific states that it wants to operate in and help cooks meet the regulations.

16 Waiting Room

The app requires a payment system for all three user types to properly function. The customer must make payments for their orders using their cards. Cooks and Drivers should be paid for their services with the money made from the order and the Home Food Delivery App will make money with a surcharge to the customer.

The cloud database charges based on the number of reads and writes to the database along with the amount of data stored. To lower operating costs for the app, developers need to lower the amount of database reads and writes in the app once the number of users scales up.

The driver should be able to see the google maps navigation to cook and customer within the app. This will make the experience smoother and safer for the driver when on the road.

The appearance of the app and the UI experience should be remade and must look and feel nice for all users. This includes optimizing the app for a much smoother and reliable experience.

The Home Food Delivery App must have an iOS version that works together with the Android version without any issues. This would greatly expand the number of potential customers, cooks, and drivers to use the app.

17 Ideas for Solutions

Payment APIs already exist that could easily be implemented to work with the app. The selected API should be able to automatically make payments to the cook and driver. It is up to future developers to select an API and make it cost effective to operate.

To optimize the number of the database reads and write calls, future developers can remove most of the database calls and rely on passing initial data from a single call in the login or registration. The data can be easily passed locally from activity to activity. Once an event happens that requires an update to the database, then a database call should be made. Currently, some activities have unnecessary database calls that would drive up the cost of operating the app when many users are active.

Xcode, SwiftUI, and UIKit can be used to develop the iOS app. Maybe the Firebase Cloud Firestore database is not compatible with iOS devices. Therefore, a different cloud database that is compatible with both Android and iOS devices is required for a central cloud database and allows the apps to work together without issues.

18 Project Retrospective

In the beginning, our group didn’t have experience developing an Android application. We started our project with only the guidance of the previous groups development

report to give us ideas how the app should look and function. We didn't plan enough as we should have at the beginning of the project, which caused us some problems that cost us time. Our first major problem was setting up and using a cloud database. This problem delayed the project for almost a month until the Cloud Firestore database was used and allowed us to properly develop the app. Our second main problem was working with the Google Maps API to allow the driver to navigate to the cook and customer. We eventually learned and got the API to work with the app, but it took more time than expected. As the app development progressed, the UI had to be edited to fit the new features.

Working on Home Food Delivery App made us realize the importance of planning software development. Even if we don't know exactly what the client or in this case ourselves want. There should always be a far more comprehensive plan of a software project that every member understands for a better time developing and to save time. The plan should include a design pattern that the developers agree upon and must follow throughout development. For applications, most of the UI should be made first to make it much easier for the developers to fill in the functionality. UI changes should be minimal during the development unless required.

VII Glossary

Activity: One screen of the UI that android studio can create and edit to make a login or profile for example. "Activity" is not used to describe what someone does in the context of this project.

Application: Downloaded and installed onto a mobile device by the user to perform specific tasks as it is a small specialized program. Does not refer to form or request.

App: abbreviation of the word application.

UI: abbreviation of "User Interface"

VIII References / Bibliography

[1] Robertson and Robertson, Mastering the Requirements Process.

[2] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, Ninth ed., Wiley, 2013.

[3] J. Bell, "Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports," Underwater Archaeological Society of Chicago, Chicago, 2012.

[4] M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

[5] Ahmed, Wasay, Reyes, Natalie , Gutierrez, Christian, Nguyen, Jordan et al. 2022, pp. 1–2, *Home Delivery App Scenario*.

[6] Ahmed, Wasay, Reyes, Natalie , Gutierrez, Christian, Nguyen, Jordan et al. 2022, pp. 1–2, *Home Delivery App Scenario 2*.

[7] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 1–70, *Home Food Delivery App*.

[8] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 55, 25F User Interface, *Home Food Delivery App*

[9] Ahmed, Wasay, Reyes, Natalie , Gutierrez, Christian, Nguyen, Jordan et al. 2022, *Home Food Delivery App images from Android Studio*.
[10] *Cottage Food Laws*. Forrager. (n.d.). Retrieved April 23, 2022, from https://forrager.com/laws/
[11] *Software Quality Inspection Checklist for Java Classes.* Research.cs.queensu.ca. Retrieved April 23, 2022, from https://research.cs.queensu.ca/home/elec372/as2/Checklist.html
[12] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 16, 6B Implementation Environment of Current Systems, *Home Food Delivery App*
[13] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 16, 6C Partner or Collaborative Applications, *Home Food Delivery App*
[14] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 20, 7C Data Dictionary, *Home Food Delivery App*
[15] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 15, 6 Mandated Constraints, *Home Food Delivery App*
[16] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, pp. 15, 7C Data Dictionary, *Home Food Delivery App*
[17] Habibullah, Humza, Patel, Heta, Garg, Jay et al. Chicago, 2021, logo, *Home Food Delivery App*

IX Index

NA.