



**Module Code & Module Title**

**CS6PO5 Final Year Project**

**Assessment Weightage & Type**

**75% Project Final Report**

**2020 Spring**

**Student Name: Deepan Singh**

**London Met ID: 17031147**

**College ID: np01cp4a170225**

**Assignment Due Date: 5<sup>th</sup> June 2020**

**Assignment Submission Date: 4<sup>th</sup> June 2020**

**Internal Supervisor: Subeksha Shrestha**

**External Supervisor: Ishwor Shrestha**

**Word Count: 6500 (approx.)**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked.*

*I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

## Abstract

The final report contains the development process and life cycle for the Canteen app project. The report describes the current scenario and how the project could improve the situation. It also contains comparisons with already existing similar systems and survey results pointing out how the current system can be improved.

The selected methodology is briefly described with reasons for its selection then development process is documented along with screenshots of the progress at each iteration of the selected methodology. And finally, the developed final product is tested and made production ready with user instructions.

## Table of Contents

CHAPTER 1: INTRODUCTION.....	1
1.1 PROJECT DESCRIPTION.....	1
1.2 CURRENT SCENARIO AND PROBLEM DOMAIN.....	2
1.3 PROJECT AS A SOLUTION .....	2
1.4 AIM AND OBJECTIVES.....	3
CHAPTER 2: BACKGROUND .....	4
2.1 ABOUT THE END USERS.....	4
2.2 UNDERSTANDING THE SOLUTION.....	4
2.3 SIMILAR PROJECTS.....	5
2.4 COMPARISONS .....	6
CHAPTER 3: DEVELOPMENT .....	8
3.1. CONSIDERED METHODOLOGIES.....	8
3.2. SELECTED METHODOLOGY .....	9
3.3. PHASES OF METHODOLGY .....	9
3.4 INCEPTION .....	10
3.4.1 PRE-DEVELOPMENT SURVEY RESULTS .....	10
3.4.2 REQUIREMENT ANALYSIS .....	12
3.5 ELABORATION .....	19
3.5.1 WORKTABLE .....	19
3.5.2 MILESTONES.....	20
3.5.3 Gantt Chart .....	21
3.6 CONSTRUCTION .....	22
3.6.1. FIRST ITERATION .....	22
3.6.2. SECOND ITERATION .....	31
3.6.3. THIRD (FINAL) ITERATION .....	42
CHAPTER 4: TESTING AND ANALYSIS.....	53
4.1 TEST PLAN .....	53
4.1.1 MANUAL INCREMENT TEST PLAN .....	53
4.1.2 MANUAL SYSTEM TEST PLAN.....	53
4.2 MANUAL INCREMENT TESTING .....	54
4.2.1 Statistics API.....	54
4.2.2 Test 2: Fire base Implementation .....	55
4.2.3 Test 3: Notification .....	57
4.2.3 Test 3: Instruction Overlays .....	58

4.2.4 System Test 4: Tutorial Repetition.....	59
4.3 SYSTEM TESTING.....	60
4.3.1 System Test 1: Disconnection handling .....	60
4.3.2 System Test 2: Menu Update .....	61
4.4 CRITICAL ANALYSIS .....	62
CHAPTER 5: CONCLUSION .....	63
5.2 ADVANTAGES .....	63
5.4 LIMITATIONS .....	64
5.3 FUTURE WORK .....	65
CHAPTER 6: REFERENCES.....	66
CHAPTER 7: BIBLIOGRAPHY .....	67
CHAPTER 8: APPENDIX .....	68
8.1 APPENDIX A: SURVEY .....	68
8.2 APPENDIX C: SAMPLE CODES .....	70
8.2.1 SAMPLE CODE OF THE WEB APP .....	70
8.2.2 SAMPLE CODE OF THE MOBILE APP .....	77
8.3 APPENDIX D: DESIGNS .....	83
8.3.1 GANTT CHART .....	83
8.3.2 WORK LOG .....	84
8.3.3 WIREFRAME.....	86
8.4 APPENDIX E: SCREENSHOTS OF THE SYSTEM .....	90
8.4.1 MOBILE APP .....	90
8.4.2 WEB APPLICATION .....	93
8.6: USER MANUAL .....	98

## Figures

Figure 1: Bhoj app .....	6
Figure 2: Foodmandu app .....	7
Figure 3: Gantt Chart.....	21
Figure 4: Use case diagram - First Iteration .....	23
Figure 5: ERD - first iteration .....	24
Figure 6: Implementation Screenshot (API connectors and packages used) .....	27
Figure 7: Implementation Screenshot (Navigation page and tray page code) .....	28
Figure 8: Implementation Screenshot (Home page and Tray.....	28
Figure 9: Implementation Screenshot (web routes and API ) .....	29
Figure 10: Implementation Screenshot (User Controller and web menu code) .....	29
Figure 11: Implementation Screenshot (web home page) .....	30
Figure 12: Implementation Screenshot (Menu creation and edit page) .....	30
Figure 13: Use Case - Second Iteration .....	32
Figure 14: ERD – Second Iteration.....	33
Figure 15: Implementation Screenshot (API Connector and flutter packages user) .	37
Figure 16: Implementation Screenshot (Orders logic code and login logic code)....	38
Figure 17: App in development(from left: home, options drawer, tray drawer) .....	38
Figure 18: Implementation Screenshot (Web routes API routes) .....	39
Figure 19: Implementation Screenshot (node packages and order VUE template)..	39
Figure 20: Implementation Screenshot (Home page and orders logic) .....	40
Figure 21: Implementation Screenshot (home page) .....	40
Figure 22: Implementation Screenshot (Menu) .....	41
Figure 23: Implementation Screenshot – Users .....	41
Figure 24: Use Case – Third Iteration .....	43
Figure 25: Data Table - Users .....	44
Figure 26: Data Table - Roles .....	44
Figure 27: Data Table - Foods .....	44
Figure 28: Data Table – Category Food .....	44
Figure 29: Data Table - Orders .....	45
Figure 30: Data Table – Ordered Item.....	45
Figure 31: Data Table - Payments.....	45
Figure 32: Data Table – Pay methods.....	45
Figure 33: Implementation Screenshot (API Connector and flutter packages user) .	46
Figure 34: Implementation Screenshot (main.dart and navi.dart ).....	46
Figure 35: Implementation Screenshot (Scoped models and custom libraries).....	47
Figure 36: Implementation Screenshot (API modules and navigation logic) .....	47
Figure 37: Implementation Screenshot (Orders Logic and Payment logic) .....	48
Figure 38: Implementation Screenshot (home widget and menu component) .....	48
Figure 39: Implementation Screenshot (tray widget and orders widget).....	49
Figure 40: Implementation Screenshot (tray instruction, tray, payment ) .....	49
Figure 41: Implementation Screenshot (React routes and API Routes).....	50
Figure 42: Implementation Screenshot (NPM packages and Reducers for REDUX)50	
Figure 43: Implementation Screenshot (actions and API requesting services) .....	51

Figure 44: Implementation Screenshot – Home Page statistics with REDUX states	51
Figure 45: Implementation Screenshot – Transactions with REDUX states .....	52
Figure 46: Implementation Screenshot – User Detail with REDUX states.....	52
Figure 47: Test – Statistics API test .....	54
Figure 48: Test – error case .....	55
Figure 49: Test – fix.....	55
Figure 50: Test – FCM package for flutter.....	55
Figure 50: Test – Build Success.....	55
Figure 50: Test – FCM implementation .....	56
Figure 51: Test – changing order status.....	57
Figure 52: Test - mobile notification .....	57
Figure 53: Test - mobile Instruction Overlay.....	58
Figure 54: Test – Tutorial repetition.....	59
Figure 55: Test - disconnection .....	60
Figure 56: Test – Menu Update.....	61

## Tables

Table 1: User classes and Characteristics .....	13
Table 2: PFM-1 Login.....	14
Table 3: PFM-2 Menu.....	14
Table 4: PFM-3 Recommend .....	14
Table 5: PFM-4 Tray .....	15
Table 6: PFM-2 Menu.....	15
Table 7: PF-5 Pending Orders .....	15
Table 8: PF-6 Notification.....	15
Table 9: PFM-3 Recommend .....	16
Table 10: PFM-3 Recommend .....	16
Table 11: PFW-1 Login .....	16
Table 12: PFW-2 Menu .....	16
Table 13: PFW-4 Statistics.....	17
Table 14: PFW-3 Recommendation .....	17
Table 15: PFW-4 Tray .....	17
Table 16: PFW-5 Pending Orders .....	17
Table 17: PFW-3 Order History.....	18
Table 18: PFW-3 Payment .....	18
Table 19: PFW-2 User.....	18
Table 20: devised worktable.....	19
Table 21: Milestones .....	20
Table 22: user Details Table .....	25
Table 23: Foods table.....	25
Table 24: Daily Menu Table .....	25
Table 25: Food categories.....	25
Table 26: orders Table .....	26
Table 27: orders details.....	26
Table 28: payment methods.....	26
Table 29: Payments .....	26
Table 30: user Details Table .....	35
Table 31: Foods table.....	35
Table 32: Daily Menu Table .....	35
Table 33: Food categories.....	35
Table 34: orders Table .....	36
Table 35: orders details.....	36
Table 36: payment methods.....	36
Table 37: Payments .....	36
Table 38: Notification.....	36

## CHAPTER 1: INTRODUCTION

### 1.1 PROJECT DESCRIPTION

The project hereby documented is a system inclusive of a web administration portal and a mobile application which collectively functions as a food ordering system for a targeted venue such as our college canteen. During the test phase, given permission, the application was intended to be live tested on the said canteen, thus the project name: Canteen App.

The project was originally planned with Laravel framework for PHP at the server end due to the framework's innate ability to easily handle web API, their authentication middleware and database Object Relational Mapping. Although, Laravel works with the principles of a Model-View-Controlled architecture, the user views are still rendered with PHP pages and templates. As such, to compensate for the scalability and to trigger re-renders of components from the business logic, React Js is implemented along with Laravel.

Similarly, the mobile application was planned with dart language and flutter framework as they supported cross platform compatibility and had large community driven libraries. After much consideration, research and trial implementations, the scoped model was implemented as a state management method for the final iteration of the application.

## 1.2 CURRENT SCENARIO AND PROBLEM DOMAIN

Considering only the facts, the targeted venue, Islington college, despite being a reputed IT college does not have any sort of digital interface for the customers when it comes to ordering their meal in the cafeteria. Although, there is a pre-existing software operated by the staff, its usability does not extend to the students. Each student or more usually, a group of students must go to the kitchen counter to place orders. According to the survey conducted late last year (available in the appendix), most students are indecisive and spend five to ten minutes deciding the menu. When there are multiple groups of students, this cause a pileup. However much the traffic, it is still accepted by only one operator. During times with large number of students this process elongates the ordering time.

Consequently, the chefs have to prepare the order as they come and limited by the speed of the order placements, in general items are prepared one order group at a time increasing the waiting period for the customers. That in itself is an issue, but it also creates more. While the students are waiting, they occupy the limited number of tables. When the dining area is full, customers are discouraged to order since there will not be any places to sit. As the predevelopment survey can testify, the students tend to venture outside for their meals. This phenomenon combined with the conventional time-consuming process of ordering and waiting might affect the tardiness of students when arriving to class thereafter.

This is not to say a better system is absolutely necessary, but it might help.

## 1.3 PROJECT AS A SOLUTION

The proposed project is a system inclusive of an app that allows self-ordering privileges to the user. That by itself reduces the collective ordering time at the centralized counter. The app sends the order requests to the server application and are sorted for the staff when multiple items of the same type is received at the same time frame, the chefs can prepare them together further reducing the time take for the entire process.

Since most problems mentioned occurs due to improper management of time, reducing the said wasted time should solve most of the problems. But the project has more to offer than that. The Staff can keep better inventory of sales and stock All the orders and payments are kept track individually by the users as well as collectively by the administration. And considering all the receipts that won't have to be printed, it might as well be saving the environment while reducing paper costs.

## 1.4 AIM AND OBJECTIVES

The intended aim of this project is to enable easier interaction and requesting meal orders through smart phones which, in general, people always carry with them. With the application, the users can browse the availability of menu without having to visit the kitchen counter itself. After placing orders, the user can better keep track of their food and expenditure. The Users will also gain another option in case they lose or forget their wallets. Even in case the user does not have their phones with them, they will still be able to request food from the counter as before through the web application.

The system also includes a web application, which will increase the efficiency of the canteen staff. With collectively sorted order requests, the items can be produced in bulks reducing the time required to prepare each item. With statistics of the weekly sales the expected amount of orders can be predicted with some level of accuracy. Moreover, the statistics data can be used in variety of ways.

As such, along with the obvious food menu, the mobile application will also have recommendation based on past user activity and recent overall sales. The application will have a readily available tray from which the user can edit their selection without changing pages. The users can view their personal order and payment history. Such user data on each app will be protected by a password and communication through the API will be authenticated by a token.

Furthermore, the staff can also benefit from the system with easier menu creation and management. The Users can be easily notified of any changes on the menu or the status of their requested meal. The web application also keeps records of all sales which can further put to better use via system analytics. The system will generate statistics of the daily and weekly sales then based on the numbers, suggests probable food item to be ordered.

## CHAPTER 2: BACKGROUND

### 2.1 ABOUT THE END USERS

This particular software was designed with intention of implementing in Islington college. Therefore, the end users will mostly be students and teachers with average or above understanding of technologies involved or necessary to operate the application. The end users are expected to be able to intuitively use the app to some degree. The server application will mostly be operated by the kitchen staff.

### 2.2 UNDERSTANDING THE SOLUTION

To better understand the system, the following are some of the recurring terms.

JSON or JavaScript Object Notation is a data type with light weight design for faster yet understandable interchange of data. The dictionary-like notations allow humans to read and write easily while also enabling easy generation and parsing for machines.

API, Application Programming Interfaces are essentially functions that allows access to features of a system to another. Usually, data are requested from a system through API calls and receives a response, usually, in JSON format.

State management is an ideology while programming to keep track of information that may be shared between components. However, State management also extends to redrawing a component when the data is changed. Therefore, most state management methods tend to handle the change itself. Some of the state managing practices are pragmatic and scoped models for flutter, the former tested in earlier iteration of the app and latter implemented towards the final iteration, whereas Redux was implemented in the web front end.

Also, not to confuse, front end refers the section of the application the user interacts with while the back end refers to the business logic section that processes the data and interacts with the database.

## 2.3 SIMILAR PROJECTS

Currently, there are a lot of food ordering app on the app stores where most provide services from multiple vendors and usually with delivery systems. Some of them, namely, Bhoj and Foodmandu were selected for comparison and research for this project. Also necessary to mention, the canteen already has a system, albeit just the staff portal and nearly nothing for the customer.

Bhoj is a location-based application that takes in the users location and finds the catering offers around them. Users can browse the menu from variety of different food places and offers a lot of deals the user. However, the application does not offer ordering service. Users can get direction, and they have to personally visit and order from the place. Bhoj also offers delivery services within Kathmandu, Lalitpur and Bhaktapur where the bhoj staff “BhojRider” visits and orders from the location for the user then delivers them for a small commission.

Foodmandu is an app dedicated to delivery. It has a select group of restaurants and cafés where the users can order from and are delivered by the employees. Other than the prepared food, foodmandu also offers to deliver groceries if the users desire.

Not much can be said about the software already prevailing in college canteen, and due to recent lockdowns in effect investigation is hard if not impossible. But still some assumptions can be made. We know that it is capable of accepting orders and printing out receipts. However, the sole users of the system are the staff of the canteen.

## 2.4 COMPARISONS

The Bhoj app differs from this project in lot of different aspects. This project is targeted to a specific food place while bhoj provides services from lot of different location. Bhoj app has a lot of location where user can only view the menu but are not allowed to order. In some cases where delivery is possible, they charge a small commission. This project allows ordering and does not support delivery or charge anything extra other than the price of the menu item. Furthermore, other than food, the bhoj app also delivers utensils and kitchen appliances.

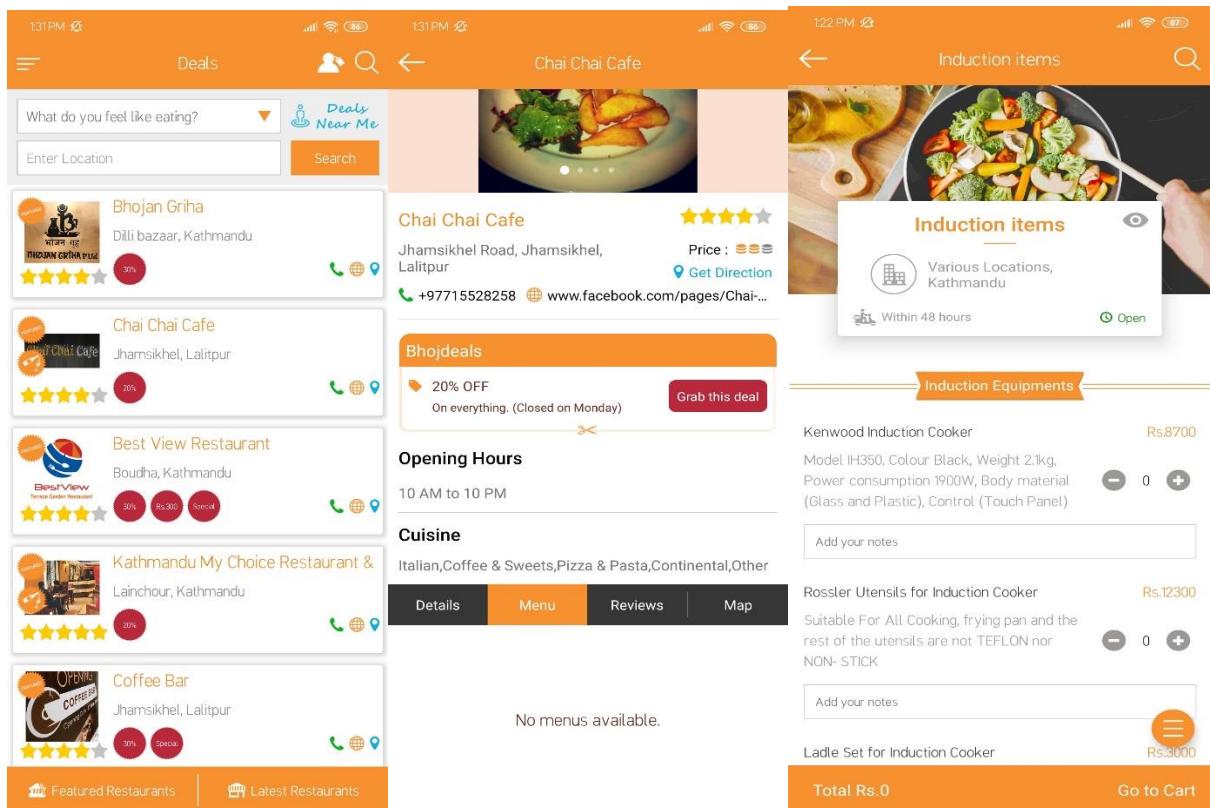


Figure 1: Bhoj app

Foodmandu app, disregarding the option to choose the origin restaurant of the food, most of the functionalities are similar. The user selects a food item and it moves to a "basket" as opposed to a "tray" in our case. However, If the user wanted to check or correct their orders, they would have to move to a different page entirely and return back to the original page to order more. In Contrast, the tray will be available at any given moment in our intended application. The user can add to the item by tapping on it or remove the item by holding the item directly from the side panel

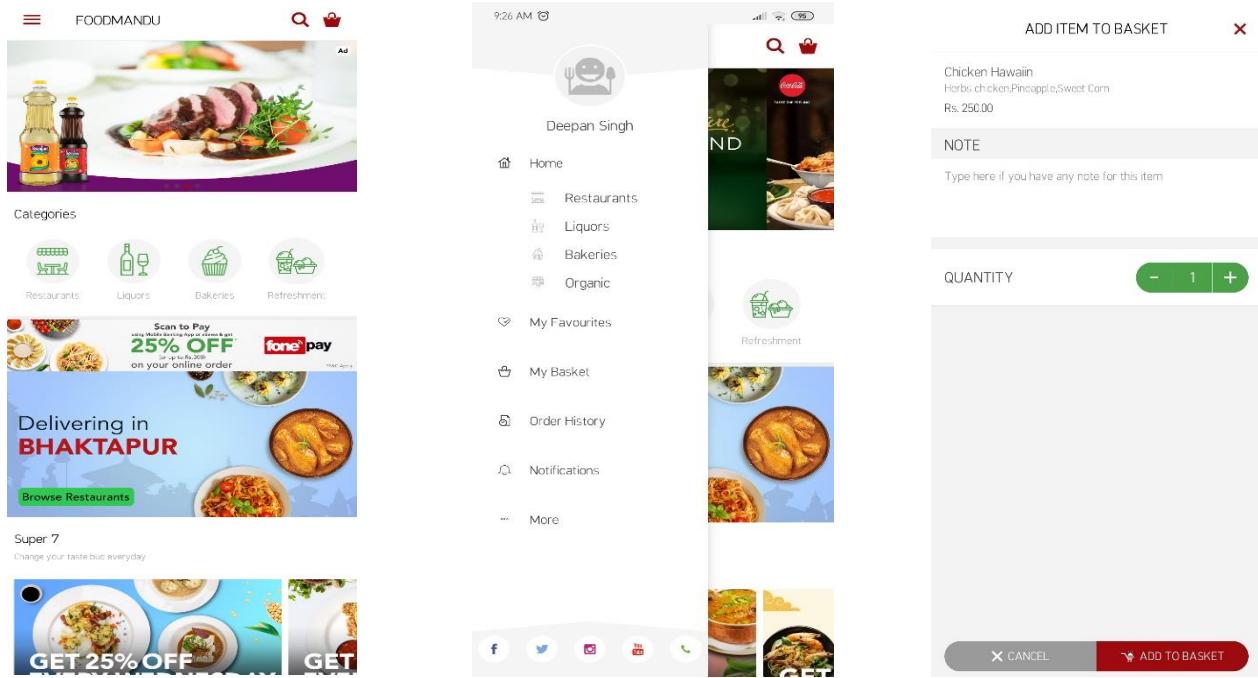


Figure 2: Foodmandu app

(From left splash screen, home page, option panel)

## CHAPTER 3: DEVELOPMENT

### 3.1. CONSIDERED METHODOLOGIES

The first option to be considered and discarded was waterfall methodology. Compared to other methods, it is very simple and has a linear sequential lifecycle. However, despite being easy to understand and follow, the waterfall approach has its own drawbacks. For instance, to start any phase in the life cycle, the previous phase must be completed and having completed the previous phase, simply going back phases to change some details is not possible. To modify the previously completed phase the entire life cycle must be started from the beginning. Thus, in view of the disadvantages, other approaches were explored.

Prototyping is also one of the simpler development approaches available, and as the name suggests, the system is developed in prototypes which is used as a model for inspection, feedback gathering and improvement for the next prototype. These prototypes aren't given much detail during the early iterations and developed rapidly with intention throwing away in mind. It is a straightforward yet flexible method, however due to the enormous amounts of recreating the prototype at each iteration, it is very time consuming. Moreover, as mentioned before, the previous iteration ultimately ends up being thrown away and unused.

Going by the books, the idea of using the agile principles was also entertained for a while. According to many established programmers, it is one of the best approaches to develop a system by. The primary benefit of the agile method as well as a drawback in my case, is the frequent collaboration between the team members. The Agile approach builds the system in iteration, the team undergoes regular scrum meetings to improve or fix defects from early in the project. But, lacking a team in the project, the major aspect of the agile method would have been wasted if implemented. Other than that, the agile approach also requires a commitment and frequent interaction with user. Each planned feature must be completed within an allotted time frame thus the rush leaves very little time for documentation.

From the previous models, it became obvious that increments were good, or at least better than starting the development over. Furthermore, there happens to be just the method focused on increments. The Incremental method starts the development early on with an initial release of a software and improves them in increments. It is very similar to prototyping, but instead of restarting the development incremental improves on the existing progress. This method was very close to selection until a better alternative was found.

### 3.2. SELECTED METHODOLOGY

After much deliberation, the final method selected for the project is iterative incremental approach. It incorporated many of the advantages of previous methods while retaining as little drawback as possible.

The iterative incremental approach also starts with an initial release of the application and since each increment is added to a copy of the previous successful build, at any point, there will be a working prototype on hand. It is a combination of iterative and incremental design with a very flexible patterns of both development methods. Admittedly, there are a lot of variation of iterative incremental method, such as the RUP, EUP and USDG however the base concept remains the same which is applied in the project.

The iterative incremental method differs from the prototyping and other iterative methods because this does not require a complete start over to add an increment and differs from the plain incremental because iterative incremental is also allows re iteration when necessary. During development phase, any new feature or module are considered as an increment and add to the system. After certain increments, when the code gets complicated and difficult to understand, usually a recode is issued.

### 3.3. PHASES OF METHODOLOGY

The iterative Incremental method can be divided into four major phases.

The Inception phase identifies the requirements, project scopes and models at a high level with just enough detail to estimate the deliverables. If it is the second or higher iteration more elaborate plans are procured based on requirements from the past iteration.

The Elaboration phase gives shape to the requirements and produces definitive deliverables while accommodating to the non-functional requirements and major risks.

The construction phase focuses on developing an increment, rather than the entire project. The increment can be a module or a new feature to be added to the current iteration. The phase is further divided into sub sections where the tasks are analysed , designed, implemented then tested to create a production ready code that fulfils the functional requirements.

However, during any of the construction phases the code becomes complex and adding in new increments becomes difficult a new iteration is started.

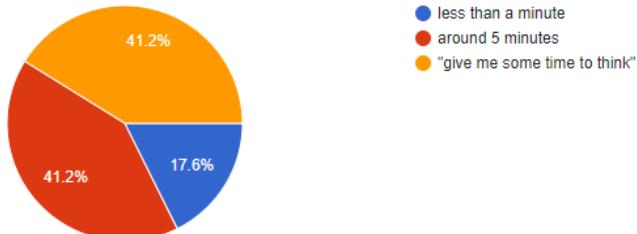
The final Transition phase delivers the iteration into the production environment.

## 3.4 INCEPTION

### 3.4.1 PRE-DEVELOPMENT SURVEY RESULTS

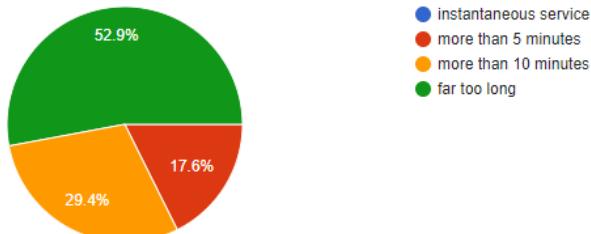
How long do you usually spend choosing meals in-front of the counter?

17 responses



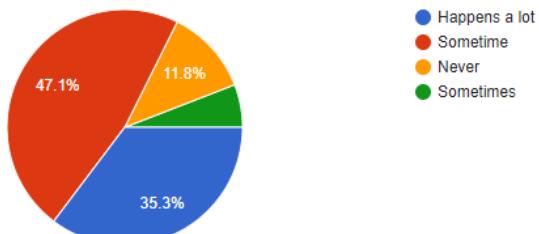
How long do you usually wait for meals once ordered in the college cafeteria?

17 responses



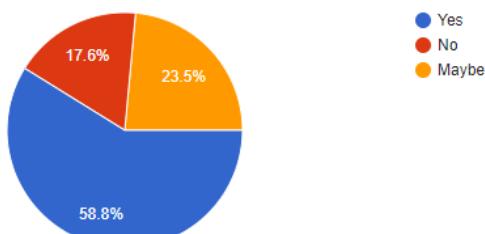
Have you had to move elsewhere because the cafeteria was too crowded?

17 responses



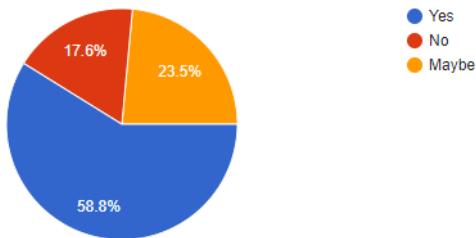
Do you have trouble deciding what to order often?

17 responses



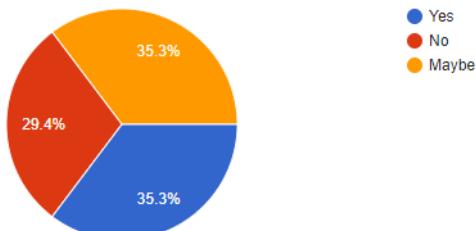
Does what other people order change your mind?

17 responses



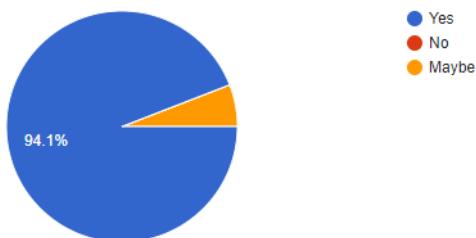
do you tend to avoid interaction with strangers?

17 responses



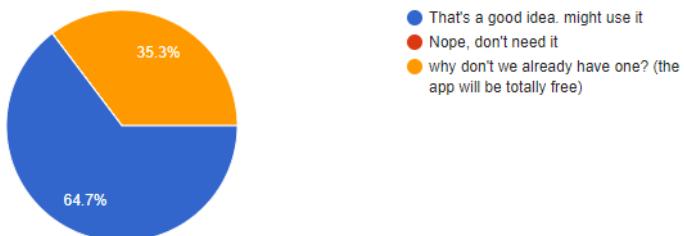
Would you want someone to place your orders and notify you when its ready?

17 responses



would it help if the college cafeteria had a mobile app?

17 responses



## 3.4.2 REQUIREMENT ANALYSIS

### 3.4.2.1 Primary Features for Mobile Application

PFM-1: Login

PFM-2: Menu

PFM-3: Recommendation

PFM-4: Tray

PFM-5: Order

PFM-6: Pending orders

PFM-7: Notification

PFM-8: Order History

PFM-9: Payment

### 3.4.2.1 Primary Features for Web Application

PFW-1: Login

PFW-2: Menu

PFW-3: Statistics

PFW-4: Recommendation

PFW-5: Tray

PFW-6: Orders

PFW-7: Order History

PFW-8: Payment

PFW-9: User

### 3.4.2.3 User Classes & Characteristics

User Class	Modules	Roles
<b>UC-1 Customer (Mobile)</b>	Login	UCC-1-1. Sign in UCC-1-2. Forgot password
	Menu	UCC-1-3. View All Menu UCC-1-4. Add menu item to tray
	Recommend	UCC-1-5. Recommend menu item.
	Tray	UCC-1-6. View item added to tray. UCC-1-7. Modify item added to tray.
	Order	UCC-1-8. Order items added to tray.
	Pending Orders	UCC-1-9. View all pending orders
	Notification	UCC-1-10. Get notified when item status is changed.
	Order History	UCC-1-11. View all past order history
	Payment	UCC-1-12. View all past payment Transactions
<b>UC-2 Staff (web)</b>	Login	UCC-2-1. Sign in UCC-2-2. Forgot password
	Menu	UCC-2-3. View All Menu items. UCC-2-4. Add/edit new items. UCC-2-5. Change item availability status
	Statistics	UCC-2-6. View daily and weekly sales statistics.
	Tray	UCC-2-7. Add Items to tray UCC-2-8. Place orders from the web portal.
	Orders	UCC-2-9. View all pending orders and update periodically UCC-2-10. Change order status and notify customer
	Order History	UCC-2-11. View every past order
	Payment	UCC-2-12. View list of all transactions made
	Users	UCC-2-13. View user list UCC-2-14. View User detail UCC-2-15. View Specific User's orders UCC-2-16. View Specific User's transactions

Table 1: User classes and Characteristics

### 3.4.2.4. System Features

#### PFM-1 Login

Req. ID	Requirement Description	Priority	Complexity
FR-01	<b>User will enter authentication details and log into their account</b>	High	High
SR-01	when logging in only valid accounts must be authorized		
SR-02	User detail will be remembered for next login		
SR-02	when 'forgot password' button is clicked, user gets new token in their email.		

Table 2: PFM-1 Login

#### PFM-2 Menu

Req. ID	Requirement Description	Priority	Complexity
FR-02	<b>User can view list of available Menu Item</b>	High	High
SR-01	Menu item must be addable to the tray		
SR-02	If an item already exists, another object must not be created. only quantity is to be increased		

Table 3: PFM-2 Menu

#### PFM-3 Recommend

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>The application recommends items to the user</b>	Low	High
SR-01	When adding to tray, same object must be used as the menu		

Table 4: PFM-3 Recommend

### PFM-4 Tray

Req. ID	Requirement Description	Priority	Complexity
FR-04	User can view and add items to the Tray	High	High
SR-01	Tray must be easily accessible from all pages		
SR-02	Adding item to the tray must instantaneously update data		

Table 5: PFM-4 Tray

### PFM-5 Order

Req. ID	Requirement Description	Priority	Complexity
FR-05	User can send Order request to the web server	High	High
SR-01	The order items and payment method must be validated		
SR-02	Successful order request must update pending items		

Table 6: PFM-2 Menu

### PFM-6 Pending Orders

Req. ID	Requirement Description	Priority	Complexity
FR-05	User can view all pending orders	High	Low

Table 7: PF-5 Pending Orders

### PFM-7 Notification

Req. ID	Requirement Description	Priority	Complexity
FR-06	User will be notified of any change in order status	High	High

Table 8: PF-6 Notification

### PFM-8 Order History

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>User will be able to view al order history</b>	Low	High
SR-01	One user can only view their own personal history		

Table 9: PFM-3 Recommend

### PFM-9 Payment History

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>User will be able to view al Payment history</b>	Low	High
SR-01	One user can only view their own personal history		

Table 10: PFM-3 Recommend

### PFW-1 Login

Req. ID	Requirement Description	Priority	Complexity
FR-01	<b>Staff will enter authentication details and log into their account</b>		
SR-01	when logging in only valid admin accounts must be authorized	High	High
SR-02	If the remember me option is selected, the user detail will be remembered for next session		
SR-02	when 'forgot password' button is clicked, user gets new token in their email.		

Table 11: PFW-1 Login

### PFW-2 Menu

Req. ID	Requirement Description	Priority	Complexity
FR-02	<b>Staff can view list of Menu Item</b>		
SR-01	Menu item must be addable to the tray		
SR-02	If an item already exists, another object must not be created. only quantity is to be increased	High	High
	Staff can create new menu item		
SR-02	Staff can modify If an item will be available on the menu.		

Table 12: PFW-2 Menu

### PFW-3 Statistics

Req. ID	Requirement Description	Priority	Complexity
FR-04	<b>The system generates and displays the daily and weekly statistics</b>	High	High
SR-01	The statistics must be displayed graphically		

Table 13: PFW-4 Statistics

### PFW-4 Recommendation

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>Based on the Statistics the application recommends bestselling items</b>	Low	High
SR-01	When adding to tray, same object must be used as the menu		

Table 14: PFW-3 Recommendation

### PFW-5 Tray

Req. ID	Requirement Description	Priority	Complexity
FR-04	<b>Staff can view and add items to the Tray</b>		
SR-01	Tray must be easily accessible from all pages		
SR-02	Adding item to the tray must instantaneously update data	High	High
	Staff can order the item in tray and accept payment at the counter.		

Table 15: PFW-4 Tray

### PFW-6 Orders

Req. ID	Requirement Description	Priority	Complexity
FR-05	<b>Staff can view all pending orders</b>		
	Staff can change the status of pending items	High	Low
	Changing item status must notify the associated user		

Table 16: PFW-5 Pending Orders

### PFW-7 Order History

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>Staff can view order history of all users</b>	Low	Low
SR-01	One user can only view their own personal history		

Table 17: PFW-3 Order History

### PFW-8 Payment

Req. ID	Requirement Description	Priority	Complexity
FR-03	<b>Staff will be able to view al Payment history</b>	Low	High

Table 18: PFW-3 Payment

### PFM-9 User

Req. ID	Requirement Description	Priority	Complexity
FR-05	<b>Staff can view list of all users</b>		
	The list of user can be searched through.	High	High
	Selecting a user shows the user details		
SR-02	Selecting a user shows individual order and transaction history		

Table 19: PFW-2 User

## 3.5 ELABORATION

### 3.5.1 WORKTABLE

<b>Tasks</b>	<b>Details</b>	<b>Description</b>	<b>Approximate Duration</b>
<b>Initiation</b>	Requirement analysis	Research the requirements for the application	1 weeks
	Risk analysis	Estimate and extrapolate a viable solution	1 week
	Use case and data models	Models for all components of the project	1 week
<b>Database</b>	inception	Plan an appropriately normalized database	1 week
	creation	Create and populate the database for initial testing	1 week
	integration	Integration of data base to the web app	1 week
<b>Web app</b>	Initial Web Application	Implementation of Laravel framework to create a front end for the web application	6 weeks
	Testing	Tests will be conducted on the web application	2 weeks
<b>Mobile App</b>	initial mobile application	Create a UI using flutter	4 weeks
	integration	Integration between database, Web and mobile app	4 weeks
<b>Prediction Algorithms</b>	Research and Implement	Research various algorithms and create an implementable	4 weeks
<b>Final Testings</b>	Unit Test	Test functionality of each unit.	2 weeks
	Integration Test	Test functionality of integration with each other.	2 weeks
<b>Documentation</b>	User Doc.	Instruction on using the system	4 weeks
	Description Doc.	Documentation on features and capabilities of the system	4 weeks

Table 20: devised worktable

### 3.5.2 MILESTONES

Milestones	Details	Expected due
<b>Entity diagrams</b>	visualization depicting relation between necessary table	Oct-20
<b>Databases</b>	Creation of table models which can be translated to another desktop if necessary	Nov-01
<b>API</b>	Application interface necessary for web to mobile communication	Nov-10
login	user authentication with temporary data	Nov-15
menu	view available and create new menu	Nov-20
order	push orders to database through web API	Dec-10
history	view order history and balance	Dec-15
<b>App</b>	initial completion of mobile app	Dec-20
<b>Web</b>	add remaining features to web app	Jan-01
<b>Notification</b>	Add notification services to notify when meals are ready	Jan-10
<b>UI</b>	finalize UI	Jan-20
<b>Eloquence</b>	Optimize the code	Feb-20
<b>Prediction algorithm</b>	implement prediction algorithms	Mar-01
<b>Beta Test</b>	Initial release of the app	Mar-25
<b>Documentation</b>	Fully revised and completed documentation	Apr-20

Table 21: Milestones

### 3.5.3 Gantt Chart

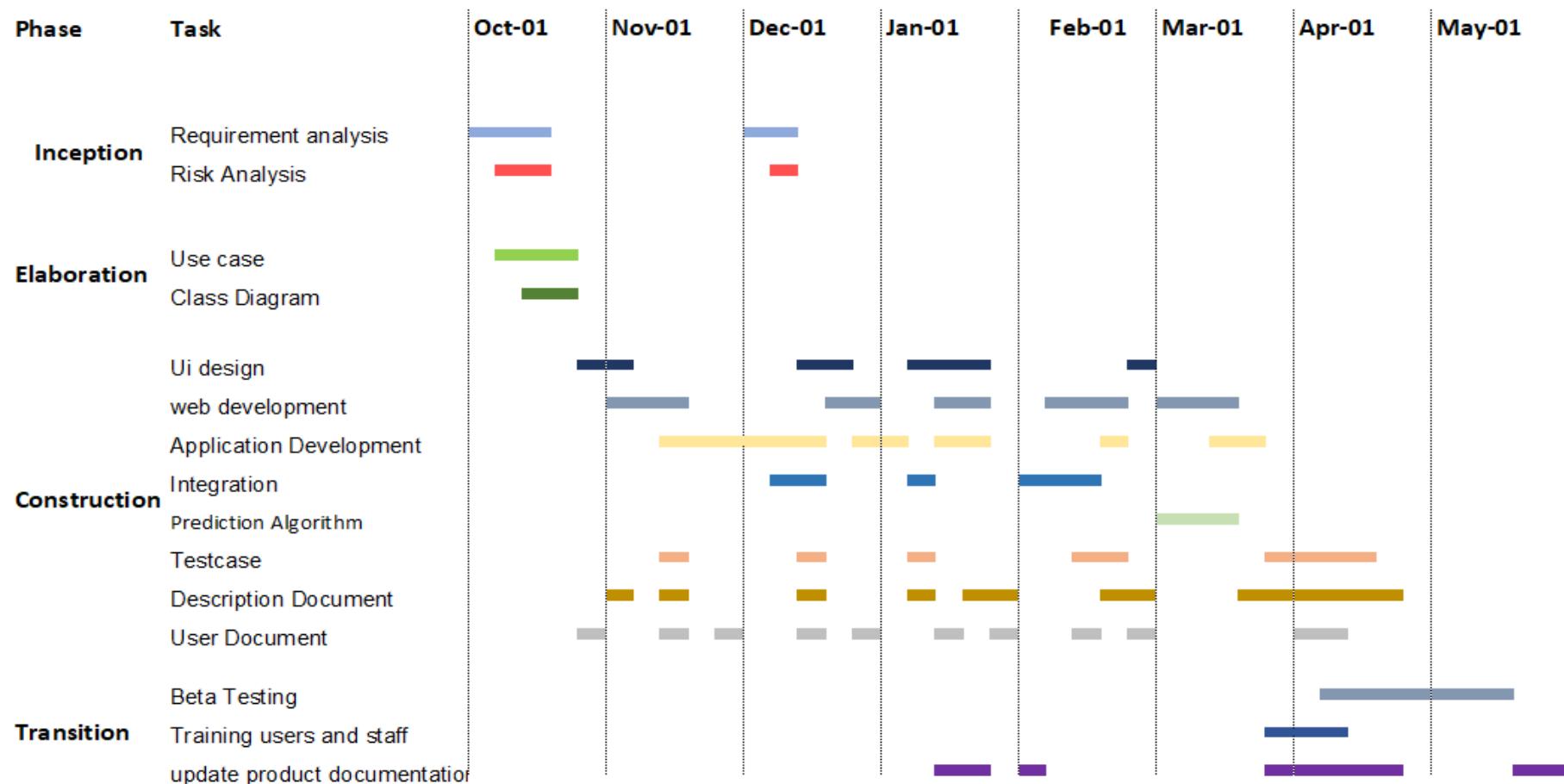


Figure 3: Gantt Chart

## 3.6 CONSTRUCTION

### 3.6.1. FIRST ITERATION

#### 3.6.1.1 ANALYSIS

The first iteration is mainly about creating an early module. It does not require much features, just that an initial product is developed fast and in working condition. With that as a starting point other modules and features can be added.

The features planned to be added to the first iteration were only basic requirements necessary for the system to function and are listed below.

- login,
- navigation
- create / view Menu
- view pending Orders and history

As for the mobile application, the basic functions would include:

- Login
- navigation
- API interaction module
- view menu
- tray
- Place and view orders

### 3.6.1.2. DESIGN

#### Use Case Diagram

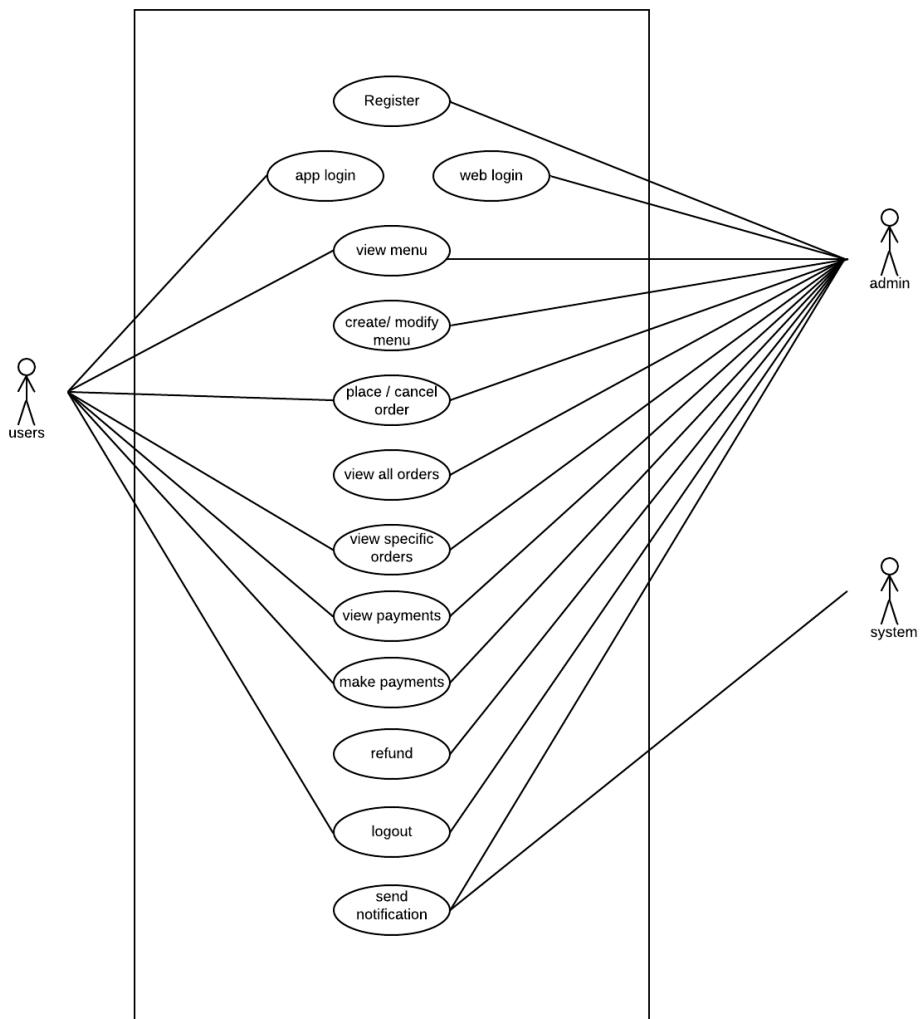


Figure 4: Use case diagram - First Iteration

## Entity Relation Diagram

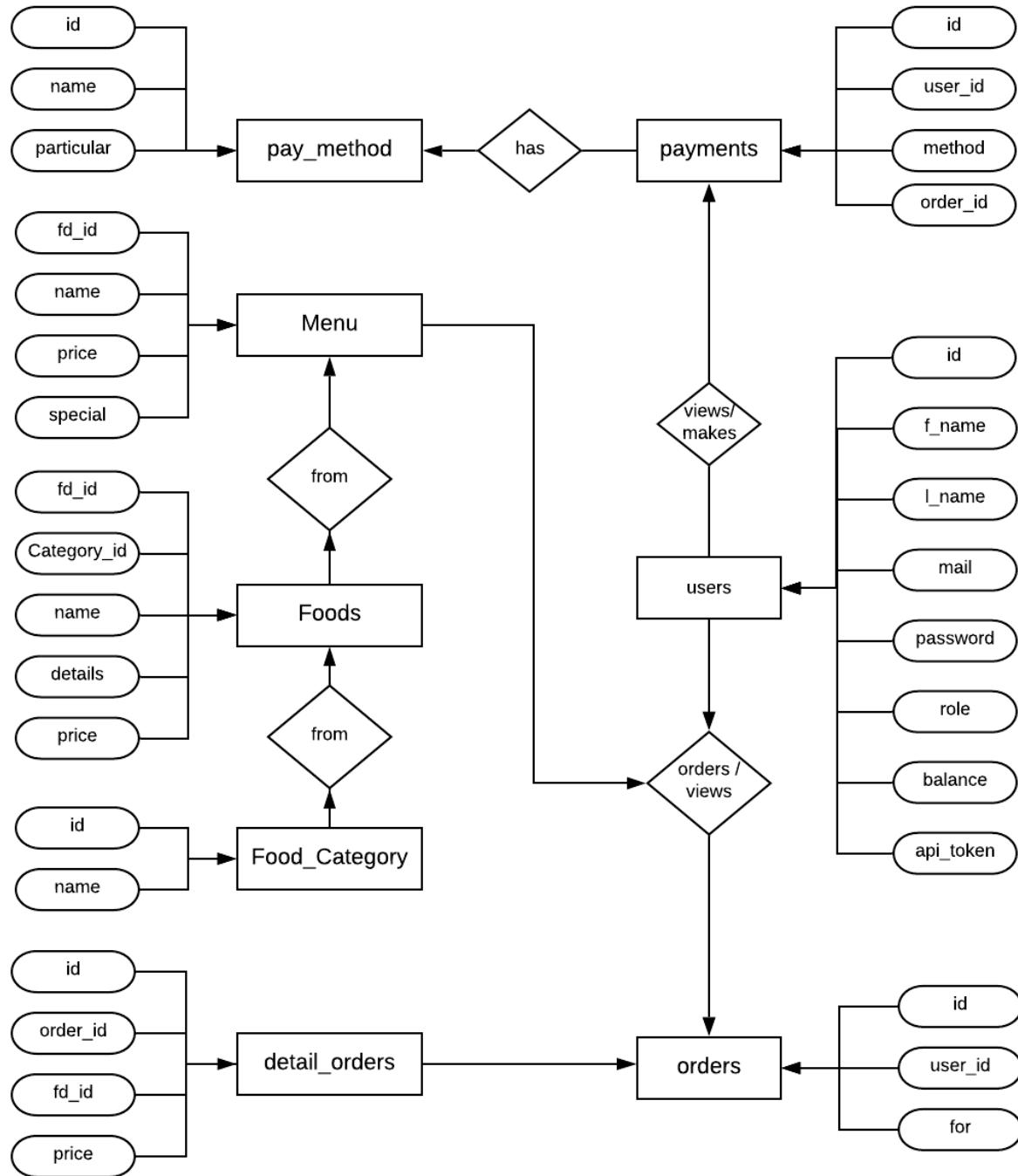


Figure 5: ERD - first iteration

## Data Dictionary

users						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Primary, Foreign	Student id	Integer	8	Yes	
F_name		Student name	String	50	Yes	
L_name		Student name	String	50	Yes	
mail	unique	Student's email address	String	50	Yes	
password		Student's address	String	100	No	
balance		Student's salary	Integer	12	No	
Api_token	Unique	Api Authenticator	String	60	No	
role		Privileges (staff or student)	String	50	No	

Table 22: user Details Table

Foods						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
food_id	Primary	used uniquely to identify each dish	Integer	4	Yes	
name		Name of the dish served	String	50	Yes	
Details		information about food	String	50	Yes	
Category_id		Type of the food (e.g. breakfast, drink)	String	50	Yes	
Price		Price of the item	Integer	5	Yes	

Table 23: Foods table

Menu						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
Item id	Primary	unique identifier for menu items	Integer	4	Yes	
food id		Food if from foods table	String	4	No	
special		Available only on set date	Boolean	1	No	
price		Daily price (may differ from the foods table)				

Table 24: Daily Menu Table

Category Food						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Primary	Account identifier	Integer	8	Yes	
name	unique	Name to Category,	String	50	Yes	

Table 25: Food categories

orders					
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>
id	Unique	Order identifier	Int	11	Yes
user_id		Student id	Int	11	Yes
on_Time		Time of the order	Date Time		Yes

Table 26: orders Table

Detail Orders					
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>
id	Unique	Order identifier	Int	11	Yes
Order_id		order id	Int	11	Yes
fd_id		Food id	Int	11	Yes
price		Price when ordered	int	11	Yes

Table 27: orders details

Pay method					
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>
id	Unique	Payment method identifier	Int	11	Yes
name		Name of payment method	Int	11	Yes
particular		Attribute specific to the method	Date Time		Yes

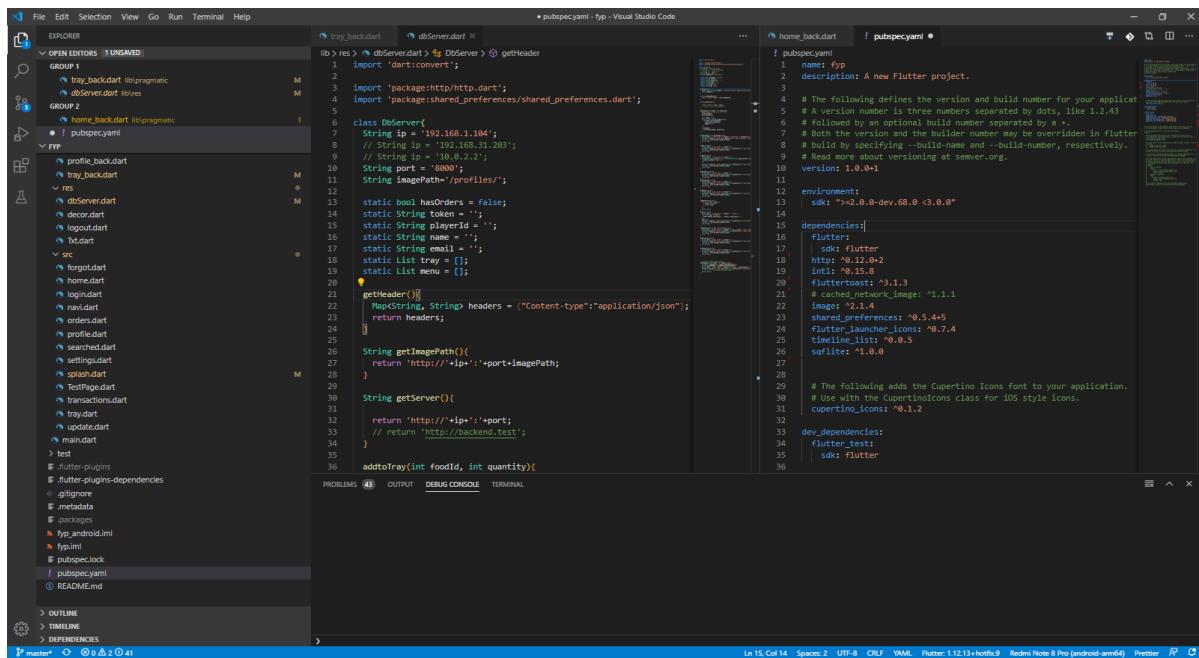
Table 28: payment methods

Payments					
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>
id	Unique	Order identifier	Int	11	Yes
user_id		User id	Int	11	Yes
method_id		Food id	Int	11	Yes
Order_id		Price when ordered	int	11	Yes

Table 29: Payments

### 3.6.1.3. IMPLEMENTATION

After completing the planned features, some other functionalities were sequentially added to the application. For instance, order history and settings pages were add to the mobile application. However, during implementation of the additional features, it became a hassle to trigger a change in a component from a completely different component. As more modules were added the flow of the program was getting mixed up, thus another iteration was necessary with proper way to manage data, State management.



The screenshot shows the Visual Studio Code interface with two tabs open. On the left, the file `dbServer.dart` is displayed, containing Dart code for a database server. On the right, the file `pubspec.yaml` is displayed, containing YAML configuration for a Flutter project. The code editor shows syntax highlighting and code completion suggestions. The bottom status bar indicates the current file is `pubspec.yaml`.

```
dbServer.dart
import 'dart:convert';
import 'package:http/http.dart';
import 'package:shared_preferences/shared_preferences.dart';

class DBServer{
    String ip = "192.168.1.104";
    // String ip = "192.168.31.203";
    // String ip = "10.0.2.2";
    String port = "8000";
    String imagePath = "/profiles/";

    static bool hasOrders = false;
    static String token = '';
    static String playerId = '';
    static String name = '';
    static List orders = [];
    static List tray = [];
    static List menu = [];

    getHeader(){
        Map headers = {"Content-type": "application/json"};
        return headers;
    }

    String getImagePath(){
        return "http://"+ip+":"+port+imagePath;
    }

    String.getServer(){
        return "http://"+ip+":"+port;
        // return "http://backend.test";
    }

    addtoTray(int foodie, int quantity){
        if(hasOrders)
            hasOrders = false;
        else
            hasOrders = true;
    }
}

home_back.dart
name: 'FYP'
version: '1.0.0+1'
environment:
  sdk: '>2.0.0-dev.68.0 <3.0.0'
dependencies:
  flutter:
    sdk: flutter
  http: ^0.12.0+2
  intl: ^0.15.8
  fluttertoast: ^3.1.3
  cached_network_image: ^1.1.1
  image: ^2.1.4
  shared_preferences: ^0.5.4+5
  flutter_launcher_icons: ^0.7.4
  timeline_list: ^0.0.5
  sqflite: ^1.0.0

# The following adds the CupertinoIcons font to your application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^0.1.2
```

Figure 6: Implementation Screenshot (API connectors(left) and packages used(right))

The screenshot shows a Visual Studio Code interface with several tabs open across different panes:

- File Explorer**: Shows the project structure with files like `meMenuButton.dart`, `recommendations.dart`, `splash.dart`, `nav.dart`, `custombar.dart`, `tray.dart`, `profile.dart`, `dbServer.dart`, `pubspec.yaml`, and `home.dart`.
- Editor**: The main code editor pane displays `tray.dart`. It contains Dart code for a `Tray` widget that builds a `Container` with `BoxDecor` and `BoxDecoration`. A yellow dot at line 31 indicates a breakpoint.
- Output**: Shows the output of the build process, including imports for `package:flutter/res/xr.dart`, `package:provider/provider.dart`, and `package:provider/provider.dart`.
- Terminal**: Shows the command `tray.dart -tvp` being run.
- Problems**: Shows 43 errors or warnings.

Figure 7: Implementation Screenshot (Navigation page(left) and tray page code(right))

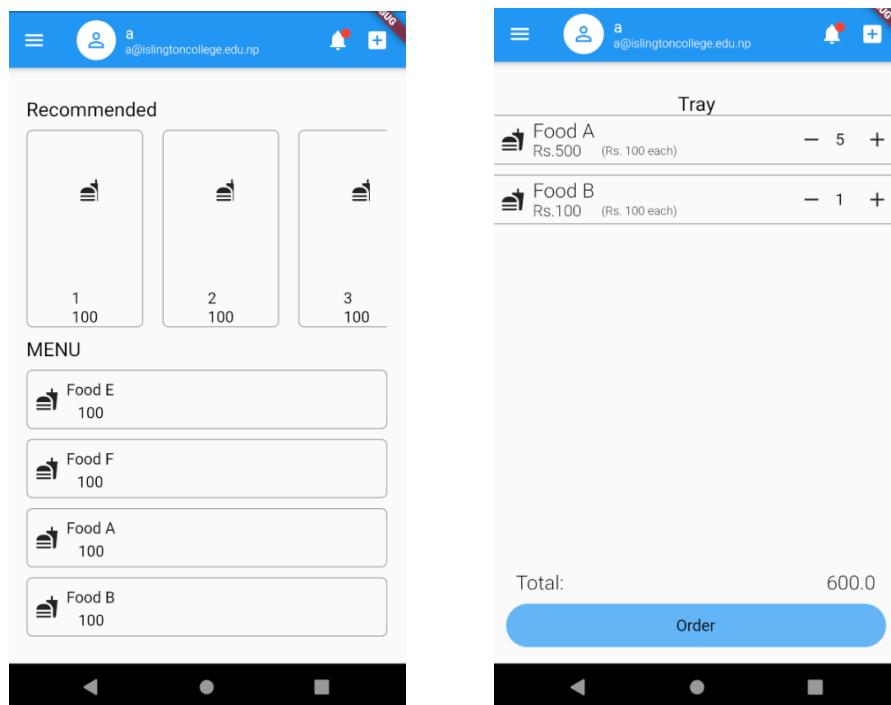


Figure 8: Implementation Screenshot (Home page(left) and Tray(right))

The screenshot shows two side-by-side code editors in Visual Studio Code. The left editor displays the file `web.php`, which contains web routes for controllers like `HomeController`, `OrderController`, and `UserController`. The right editor displays the file `api.php`, which contains API routes for controllers like `FoodController`, `MenuController`, and `UserForgotPasswordController`. Both files show Laravel's route definition syntax.

Figure 9: Implementation Screenshot (web routes(left) and API (right))

The screenshot shows two side-by-side code editors in Visual Studio Code. The left editor displays the `UserController.php` file, which contains PHP code for handling user login and logout requests. The right editor displays a `menu.blade.php` file, which contains Blade template code for generating a food menu with categories and items.

Figure 10: Implementation Screenshot (User Controller(left) and web menu code(right))

**Frequent**

Food E 100	Food F 100	Food A 100	Food B 100
---------------	---------------	---------------	---------------

**Menu**

Search <input type="text"/> Q			
Food E 100	Food F 100	Food A 100	Food B 100

**Orders**

- User: b
 

1 Food E	x 1
2 Food F	x 1
- User: a
 

3 Food E	x 1
----------	-----
- User: a
 

4 Food E	x 1
----------	-----
- User: a
 

5 Food E	x 3
----------	-----
- User: a
 

6 Food E	x 3
7 Food F	x 1
- User: a
 

8 Food E	x 3
9 Food F	x 1
- ...

Figure 11: Implementation Screenshot (web home page)

**Menu**

Search <input type="text"/> Q		
<b>BreakFast</b>	Food A <sup>100</sup>	
<input checked="" type="checkbox"/>	Food B <sup>200</sup>	
<input checked="" type="checkbox"/>	Food G <sup>100</sup>	
<input type="checkbox"/>	Food G <sup>100</sup>	
<input checked="" type="checkbox"/>	nn <sup>100</sup>	
<input checked="" type="checkbox"/>	naya <sup>120</sup>	
<b>Lunch</b>	<input checked="" type="checkbox"/>	Food C <sup>100</sup>
<input checked="" type="checkbox"/>	Food D <sup>100</sup>	
<input checked="" type="checkbox"/>	Food E <sup>100</sup>	
<input checked="" type="checkbox"/>	Food F <sup>100</sup>	
<input type="checkbox"/>	Food H <sup>100</sup>	
<input checked="" type="checkbox"/>	Food H <sup>100</sup>	

**Create**

name <input type="text"/>
category <input type="text" value="BreakFast"/>
price <input type="text"/>
details <input type="text"/>

**Create**

Figure 12: Implementation Screenshot (Menu creation and edit page)

### 3.6.2. SECOND ITERATION

#### 3.6.2.1 ANALYSIS

On the second iteration, the most important aspect is state management, and the method select for is Pragmatic Providers. Pragmatic method separates the component that trigger change as provider and the affected component as consumers. With the proper state management all previous codes will be restructured for mobile application.

On the other hand, web application also needs a self-reloading component when new orders are received. For this, angular java scripts will be implemented.

Therefore, objectively, the features initially planned as a must for this iteration are listed below.

- login,
- navigation
- Menu
- self-reloading Orders list
- payment
- Payment history

As for the mobile application, the basic functions would include:

- State managed
- Login
- navigation
- API interaction module
- menu
- quick access tray
- Place and view orders
- payment
- Payment history

### 3.6.3.2. DESIGN

#### Use Case

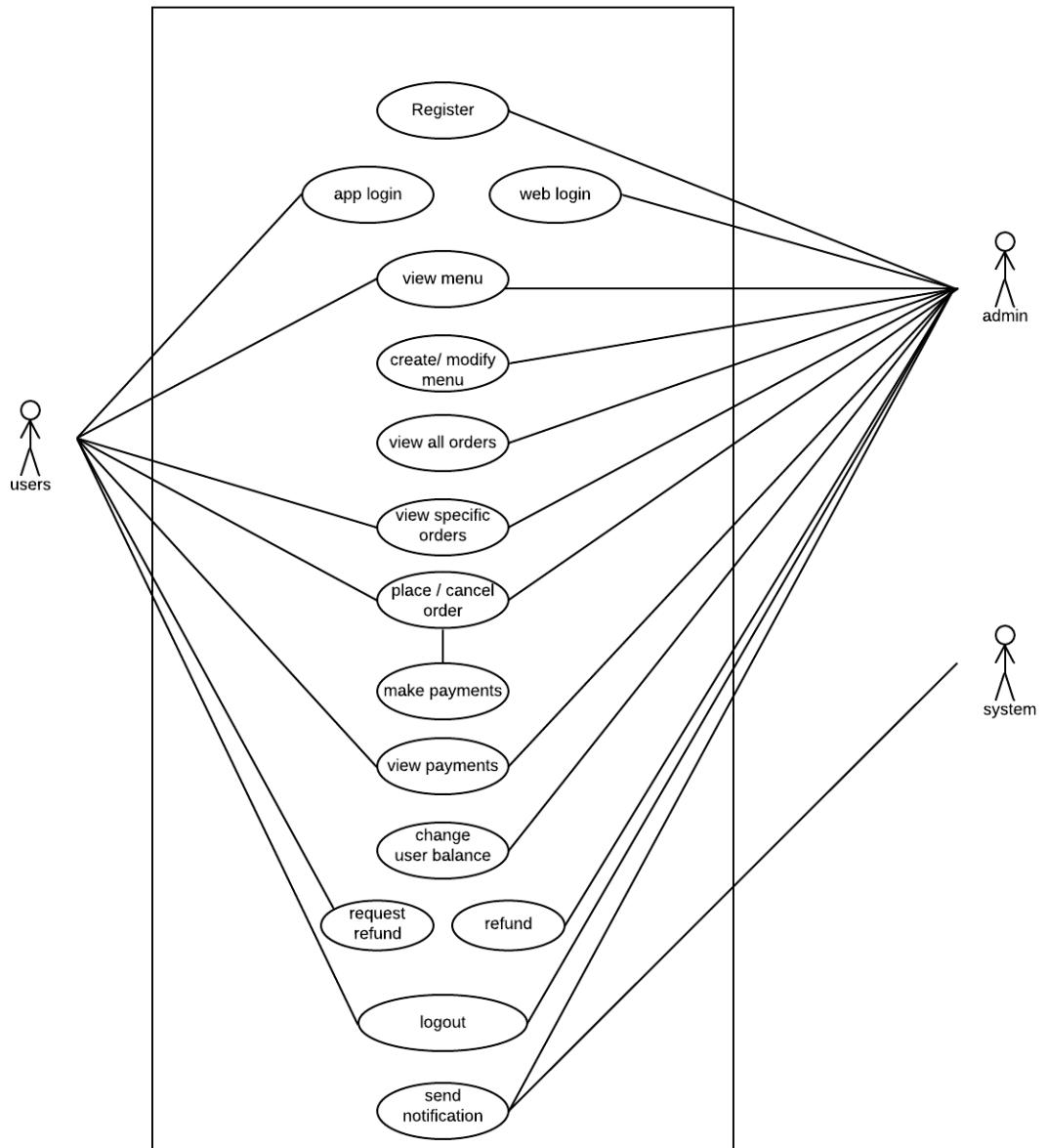


Figure 13: Use Case - Second Iteration

## Entity Relational Diagram

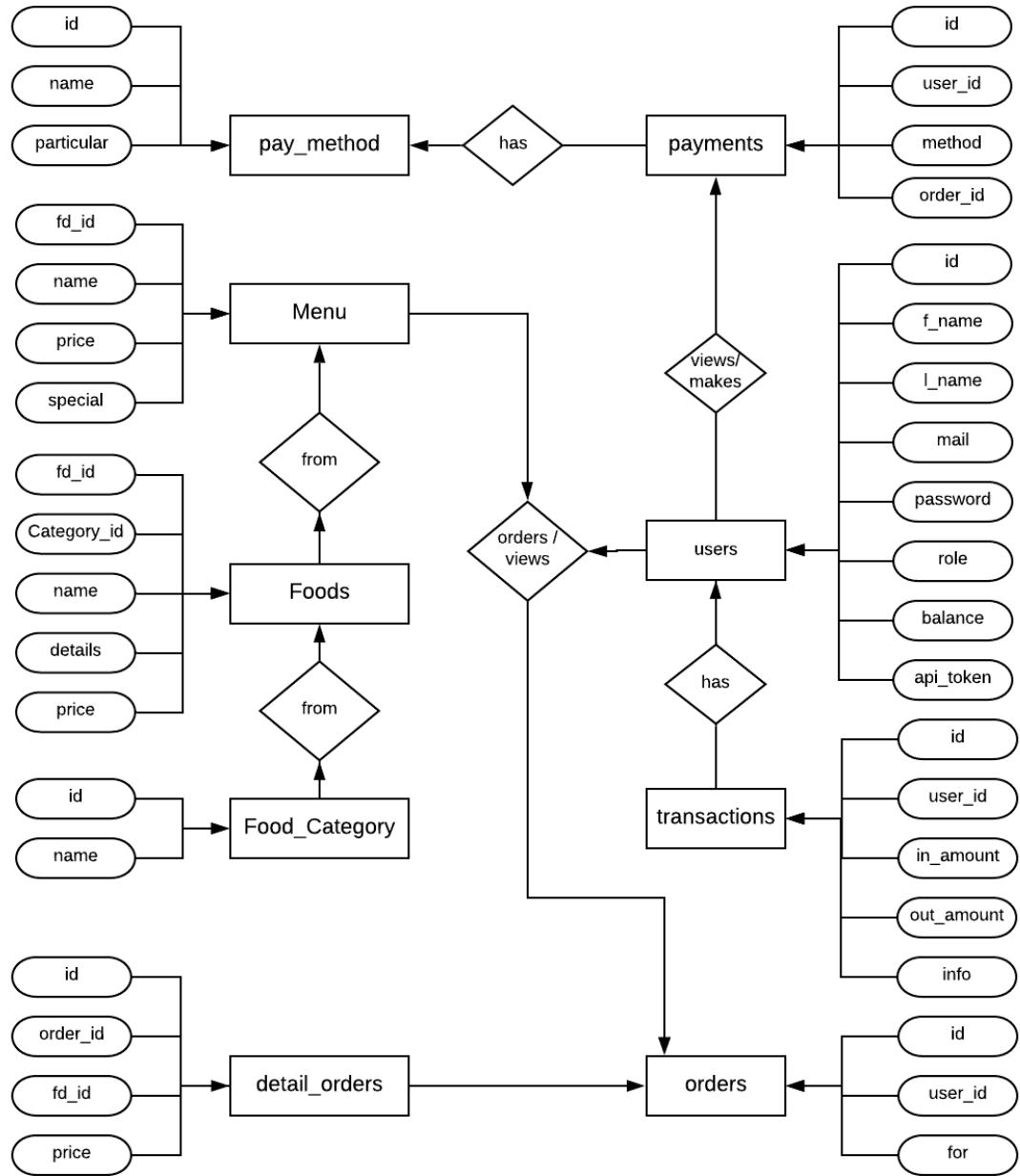
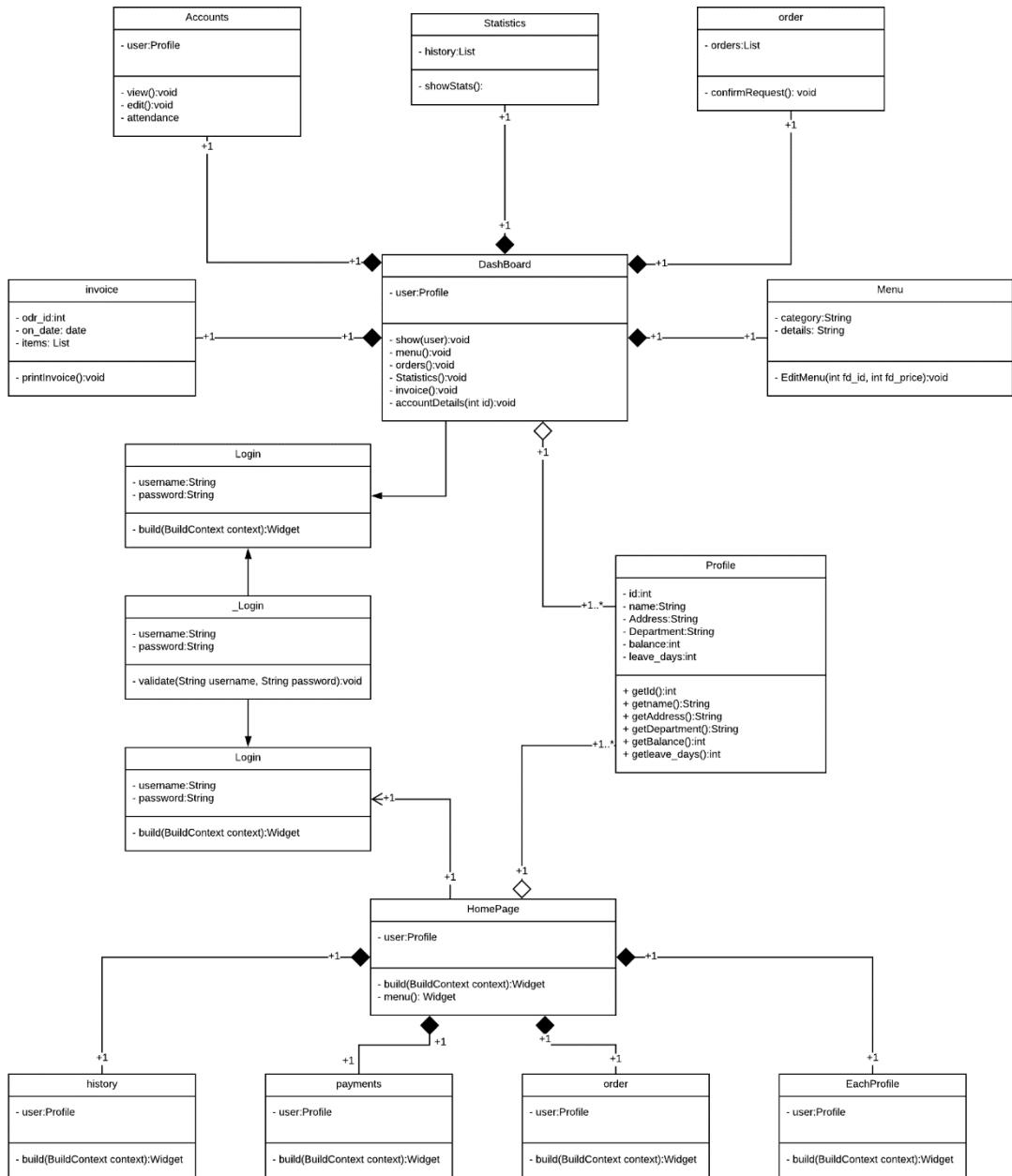


Figure 14: ERD – Second Iteration



## Data Dictionary

users						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Primary, Foreign	Student id	Integer	8	Yes	
F_name		Student name	String	50	Yes	
L_name		Student name	String	50	Yes	
mail	unique	Student's email address	String	50	Yes	
password		Student's address	String	100	No	
balance		Student's salary	Integer	12	No	
Api_token	Unique	Api Authenticator	String	60	No	
role		Privileges (staff or student)	String	50	No	

Table 30: user Details Table

Foods						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
food_id	Primary	used uniquely to identify each dish	Integer	4	Yes	
name		Name of the dish served	String	50	Yes	
Details		information about food	String	50	Yes	
Category_id		Type of the food (e.g. breakfast, drink)	String	50	Yes	
Price		Price of the item	Integer	5	Yes	

Table 31: Foods table

Menu						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
Item id	Primary	unique identifier for menu items	Integer	4	Yes	
food id		Food if from foods table	String	4	No	
special		Available only on set date	Boolean	1	No	
price		Daily price (may differ from the foods table)				

Table 32: Daily Menu Table

Category Food						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Primary	Account identifier	Integer	8	Yes	
name	unique	Name to Category,	String	50	Yes	

Table 33: Food categories

orders						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Unique	Order identifier	Int		11	Yes
user_id		Student id	Int		11	Yes
on_Time		Time of the order	Date Time			Yes

Table 34: orders Table

Ordered Item						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Unique	Order identifier	Int		11	Yes
Order_id		order id	Int		11	Yes
fd_id		Food id	Int		11	Yes
quantity		Number of items ordered	int		11	yes
price		Price when ordered	int		11	Yes

Table 35: orders details

Pay method						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Unique	Payment method identifier	Int		11	Yes
name		Name of payment method	Int		11	Yes
particular		Attribute specific to the method	Date Time			Yes

Table 36: payment methods

Payments						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Unique	Order identifier	Int		11	Yes
user_id		User id	Int		11	Yes
method_id		Food id	Int		11	Yes
Order_id		Price when ordered	int		11	Yes

Table 37: Payments

Notifications						
<b>name</b>	<b>key</b>	<b>Description</b>	<b>Datatype</b>	<b>Length</b>	<b>Required</b>	
id	Primary	unique identifier for notifications	Integer		4	Yes
type						
notifiable						
user_id		User id of the notification receiver	String		4	yes
data		event details	String		50	Yes
on_date		occurrence of event date	String		50	Yes

Table 38: Notification

### 3.6.2.3. IMPLEMENTATION

Implementing State management was not easy; however, the tasks were completed with satisfactory results. The most observable development was the tray component that could update when the items were added from the menu component. In addition to the planned tasks, notification was also implemented. Being a flutter project, firebase cloud services was the first choice and it turned out perfectly

Situations usually don't go as planned. In case of web application implementing Angular turned harder than expected. Even after much research, trial and error, the progress did not go any further. So, alternatives were considered. Upon research, VUE JS was found to be better than angular despite both of them being variations of java script.

## Mobile App

```

dbServer.dart
29     String getServer(){
30         return 'http://'+ip+':'+port;
31         // return 'http://backend.test';
32     }
33
34 }
35
36
37 void addToTray(int foodId, int quantity){
38     print("Add to tray =====");
39     print(foodId);
40     print(quantity);
41     bool added= false;
42     for(int i=0; i

```

pubspec.yaml
1 name: fyp
2 description: A new Flutter project.
3
4 # The following defines the version and build number for your application.
5 # A version number is three numbers separated by dots, like 1.2.45
6 # A build number adds an optional second number separated by a +
7 # Both the version and the build number can be overridden in flutter
8 # build by specifying -v build-name and -b build-number, respectively.
9 # Read more about versioning at https://flutter.dev/deep-dive/#versioning
10 version: 1.0.0+
11
12 environment:
13   sdk: >2.0.0-dev.68.0 (<3.0.0"
14
15 dependencies:
16   flutter:
17     | flutter
18     |+ firebase_messaging: ^9.0.0
19     |+ flutter_local_notifications: ^9.0.2
20     |+ sqflite: ^1.0.0
21     |+ http: ^0.12.0+2
22     |+ intl: ^0.15.8
23     provider: ^2.0.0+1
24     fluttertoast: ^3.1.3
25     image_picker: ^0.4.10
26     |+ flutter_staggered_grid_view: ^1.1.1
27     |+ rxdart: ^2.1.4
28     shared_preferences: ^0.5.4+5
29     flutter_launcher_icons: ^0.7.4
30     timeline_list: ^0.0.5
31     sqflite: ^1.0.0
32     file_testing: ^2.0.3
33     |+ flutter_local_notifications: ^9.0.4+3
34
35
36 # The following adds the CupertinoIcons font to your application.

```


```

Figure 15: Implementation Screenshot (API Connector (left) and flutter packages user(right))

```

orders_back.dart
lib> pragmatic > orders_back.dart > OrderBack @ update
1 import 'package:flutter/material.dart';
2 import 'package:typ/res/dbServer.dart';
3
4 class OrderBack with ChangeNotifier{
5   List allOrders = [];
6   update(){}
7   void DBServer().getOrders().then(val){
8     if(val!=null){
9       allOrders=[];
10      print("val");
11      print(val);
12      int order_id = 0;
13      List each = [];
14      int status = 0;
15      int i = 0;
16      for(var item in val){
17        i++;
18        print("item");
19        print(item);
20        if(order_id != item['order_id']){
21          print("object");
22          print(each.length);
23          if(each.length>0){
24            print('here');
25            allOrders.add(each, status, i);
26          }
27          order_id=item['order_id'];
28          each=[];
29        }
30        each.add([
31          item['created_at'],
32          item['food'][name],
33          item['quantity'],
34          item['price'],
35          item['status'],
36        ]);
}
}
}

```

```

log_back.dart
lib> pragmatic > log_back.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:typ/res/dbServer.dart';
3
4 class LogBack with ChangeNotifier{
5   bool logged = false;
6   bool loading = false;
7
8   changeStatus(bool s){
9     loading = s;
10    notifyListeners();
11  }
12  changeLogged(bool s){
13    logged = s;
14    notifyListeners();
15  }
16  Login(){
17    DBServer().login().then(val){
18      if(val==null){
19        if(val['result']=="success"){
20          DBServer.token=val['token'];
21        }
22      }
23    });
24  }
}

```

Figure 16: Implementation Screenshot (Orders logic code(left) and login logic code(right))

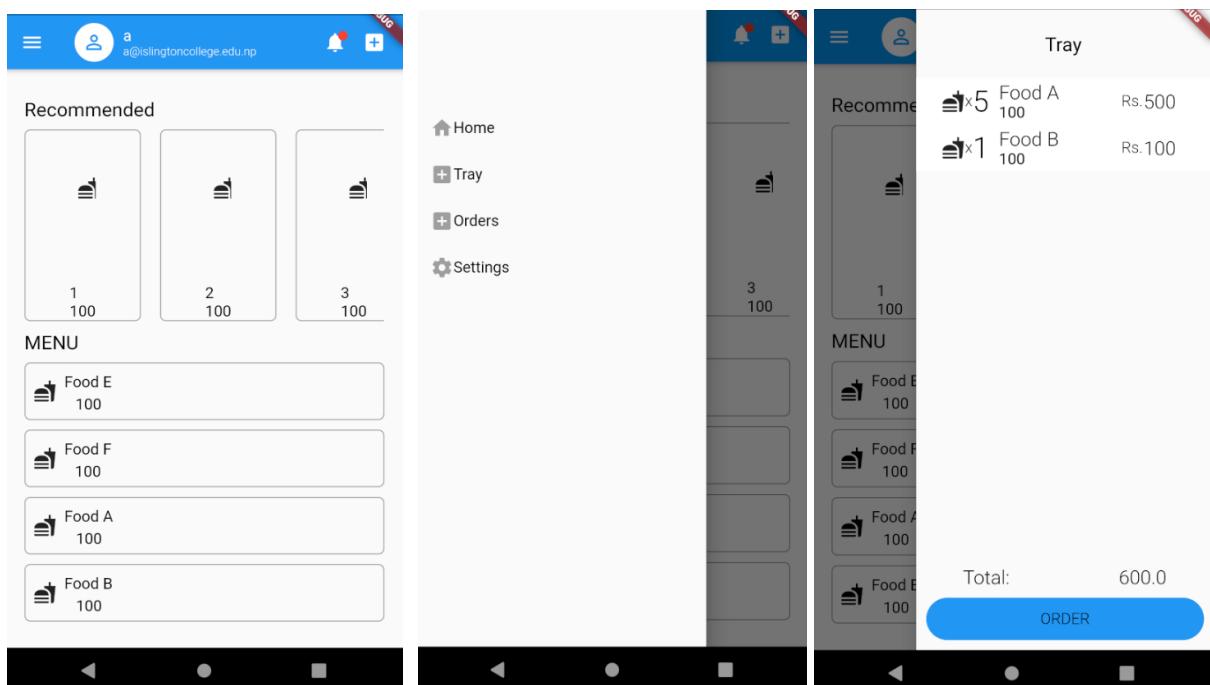


Figure 17: App in development(from left: home, options drawer, tray drawer)

## Web App

The screenshot displays two open tabs in Visual Studio Code:

- api.php - BackEnd - Visual Studio Code**: This tab contains PHP code for defining routes and controllers. It includes routes for 'web.php' and 'api.php', with specific endpoints like '/home' and '/order'. It also includes logic for handling file uploads and returning JSON responses.
- HomeOrders.vue - app.blade.php - apiphp - Visual Studio Code**: This tab contains a混合代码示例，结合了Vue.js的逻辑和Blade PHP的视图。它显示了一个名为'HomeOrders.vue'的组件，该组件使用了Blade语法（如{{ \$user }}）。

At the bottom of the interface, there is a terminal window titled "powershell" showing the command "PS E:\Assignments\Y3Computing\sys\BackEnd".

*Figure 18: Implementation Screenshot (Web routes (left) API routes(right))*

The screenshot displays two open tabs in Visual Studio Code, both titled 'BackEnd - Visual Studio Code'. The left tab contains PHP code for 'OrderItem.vue', and the right tab contains Vue.js code for 'OrderedItem.vue'. Each tab also includes its corresponding package.json file.

**OrderItem.vue - BackEnd - Visual Studio Code**

```
File Edit Selection View Go Run Terminal Help
REPL
EXPLORER
OPEN EDITORS
GROUP 1
  web.php
    package.json
  package.json
GROUP 2
  api.php
    routes
  package.json
  ORDEREDITEM.VUE
    resources\js\components\OrderedItem.vue
  ORDEREDITEM.VUE
    resources\js\components\ExampleComponent.vue
    HomeOrders.vue
    OrderedItem.vue
  app.js
JS bootstrap.js
JS homeOrders.js
lang
sass
views
auth
layouts
  appblade.php
home.blade.php
menu.blade.php
user.blade.php
welcome.blade.php
routes
  api.php
  channels.php
  console.php
  web.php
storage
tests
vendor
OUTLINE
  () OrderItem.vue
    script
    id
    template
      div.container
        div.row.justify-content-center
          div.col-md-8
            div.card
              div.card-header
                order_id
              div.card-body
                name
              div.order
    script
      export default {
        id() {
          return '1';
        }
      }
  NPM SCRIPTS
```

**OrderedItem.vue - BackEnd - Visual Studio Code**

```
File Edit Selection View Go Run Terminal Help
REPL
REORDEREDITEM.VUE
  package.json
  package.json
  ORDEREDITEM.VUE
    resources\js\components\OrderedItem.vue
    resources\js\components\ExampleComponent.vue
    HomeOrders.vue
    OrderedItem.vue
  app.js
JS bootstrap.js
JS homeOrders.js
lang
sass
views
auth
layouts
  appblade.php
home.blade.php
menu.blade.php
user.blade.php
welcome.blade.php
routes
  api.php
  channels.php
  console.php
  web.php
storage
tests
vendor
OUTLINE
  () OrderedItem.vue
    script
    id
    template
      div.container
        div.row.justify-content-center
          div.col-md-8
            div.card
              div.card-header
                order_id
              div.card-body
                name
              div.order
    script
      export default {
        id() {
          return '1';
        }
      }
  NPM SCRIPTS
```

Figure 19: Implementation Screenshot (node packages used (left) and order VUE template(right))

The screenshot shows the Visual Studio Code interface with two code editors open. The left editor contains the file `home.blade.php`, which includes PHP and Blade templating code for displaying a search bar and a list of food items. The right editor contains the file `homeOrders.js`, which is a Vue.js component for managing orders. The code in `homeOrders.js` includes axios requests to fetch data and update the state.

```

<div class="input-group mb-1 flex-row-reverse">
  <div class="input-group-append">
    <span class="input-group-text" id="basic-addon2">
      </span>
    </div>
  <input type="text" class="form-control col-md-3" placeholder="Search" aria-label="Search" aria-describedby="basic-addon2" />
</div>
<?php
$Category = CategoryFood::all();
foreach ($Category as $c) {
  echo "<h6 class='fpx-m-20 float-btn-'>{$c->name}</h6>";
  $Food = Food::where('category_food_id', $c->id)
  >>> where('served_today', 1)
  foreach ($Food as $item) {
    <!-- <div class="card col-md-3" style="margin-right:10px" >
      <div class="card-image" style="width:110px; height:170px; margin-bottom:10px; border-radius:10px; background-size:cover; background-position:center; background-color:#fff; position: relative; z-index:1; <img alt="{$item->image}" style="width:100%; height:100%; object-fit:cover; border-radius:10px; position: absolute; z-index:0; >
        <div class="card-body" style="padding-top:10px; padding-bottom:10px; position: relative; z-index:1; >
          <h6 class="card-title">{{ $item->name }}</h6>
          <p class="card-text" style="color: gray">{{ $item->description }}
        </div>
      </div>
    </div>
  <?php
}
echo "</div>";
}

```

```

4   new Vue({
5     el: "#HomeOrders",
6     data: {
7       orders: [
8         {
9           order_id: 1,
10           user: "name",
11           foodname: "food",
12           quantity: 1,
13         }
14       ],
15     },
16     methods: {
17       getOrders() {
18         window.axios.defaults.headers.common[
19           "X-Requested-With"
20         ] = "XMLHttpRequest";
21         window.axios.get('/orderdItems')
22           .then(({ data }) => {
23             data.forEach(orderedItem => {
24               this.orderdItems.push(new OrderedItem(orderedItem));
25             });
26           })
27         );
28       },
29       components: {
30         HomeOrders,
31       },
32       created() {
33         this.getOrders();
34       }
35     }
36   });
37 });

```

Figure 20: Implementation Screenshot (Home page(left) and orders logic(right))

The screenshot shows a web application interface. At the top, there is a navigation bar with links for Home, Menu, Transactions, and Users. Below the navigation bar, there are two main sections: 'Frequents' and 'Menu'. The 'Frequents' section displays four images of spaghetti with meat sauce, labeled 'Food A', 'Food B', 'Food C', and 'Food D', each with a price of 100. The 'Menu' section displays a grid of breakfast and lunch items. The breakfast section shows 'Food A' and 'Food B' with prices 100 and 200 respectively. The lunch section shows 'Food G' with a price of 100 and 'naya' with a price of 120. To the right of these sections is a sidebar titled 'Orders //sortable by time, user , food'. It lists several user orders with their details and a summary at the bottom.

User	Food	Quantity
User: Counter	Food A	x 2
User: a a	Food B	x 1
3	Food A	x 1
4	Food B	x 1
5	Food A	x 1
6	Food B	x 1
7	Food A	x 1
8	Food B	x 1
User: a a	10Food B	x 1
11Food A	x 1	
User: a a	13Food A	x 1
14Food B	x 1	
15Food A	x 1	

Figure 21: Implementation Screenshot (home page)

The screenshot shows a web-based application interface for managing a menu. On the left, there is a sidebar with navigation links: Home, Menu, Transactions, and Users. The main content area has a header "Menu" with a search bar and a "Q" icon. Below this, there are two sections: "BreakFast" and "Lunch".

- BreakFast:**
  - Food A<sup>100</sup>
  - Food B<sup>200</sup>
  - Food G<sup>100</sup>
  - Food G<sup>100</sup>
  - nn<sup>100</sup>
  - naya<sup>120</sup>
- Lunch:**
  - Food C<sup>100</sup>
  - Food D<sup>100</sup>
  - Food E<sup>100</sup>
  - Food F<sup>100</sup>
  - Food H<sup>100</sup>
  - Food H<sup>100</sup>

To the right of the meal lists is a "Create" form for adding new food items. The form fields include:

- name: (input field)
- category: BreakFast (dropdown menu)
- price: (input field)
- details: (text area)

A green "Create" button is located at the bottom of the form.

Figure 22: Implementation Screenshot (Menu)

The screenshot shows a web-based application interface for managing users. On the left, there is a sidebar with navigation links: Home, Menu, Transactions, and Users. The main content area has a header "Users" with a search bar and a "Q" icon.

The "Users" section displays a list of three entries:

- Counter CanteenAdmin@islingtoncollege.edu.np [X]
- a a@islingtoncollege.edu.np [X]
- b b@b.b [X]

To the right of the user list is a "Create" form for adding new users. The form fields include:

- First name: (input field)
- Last name: (input field)
- Email: (input field)

A green "Create" button is located at the bottom of the form.

Figure 23: Implementation Screenshot – Users

### 3.6.3. THIRD (FINAL) ITERATION

#### 3.6.3.1 ANALYSIS

The third iteration had to start mainly for one reason, better state management. In case of the mobile app, although the pragmatic was working fine, when sharing data between providers the task was getting tricky, multi providers and proxy providers were considered but sharing context was turning out very similar to without using a state management method. As such the development for mobile application was re iterated with scoped model as primary state manager. Also, in respect to the time contrast this is probably the final iteration, therefore, the visual aspect of the application must also be considered.

This also applies to the web application. Other than the visual user interface quality, the importance of state management is clear. To solve the problem of self-reloading components and proper state data management, React JS was implemented with Redux state manager. Learning and research phase took a considerable amount of time, but the expected end result was achieved.

Since this is the probable final iteration, all the tasks devised in the requirement analysis must be included in the project.

### 3.6.3.2. DESIGN

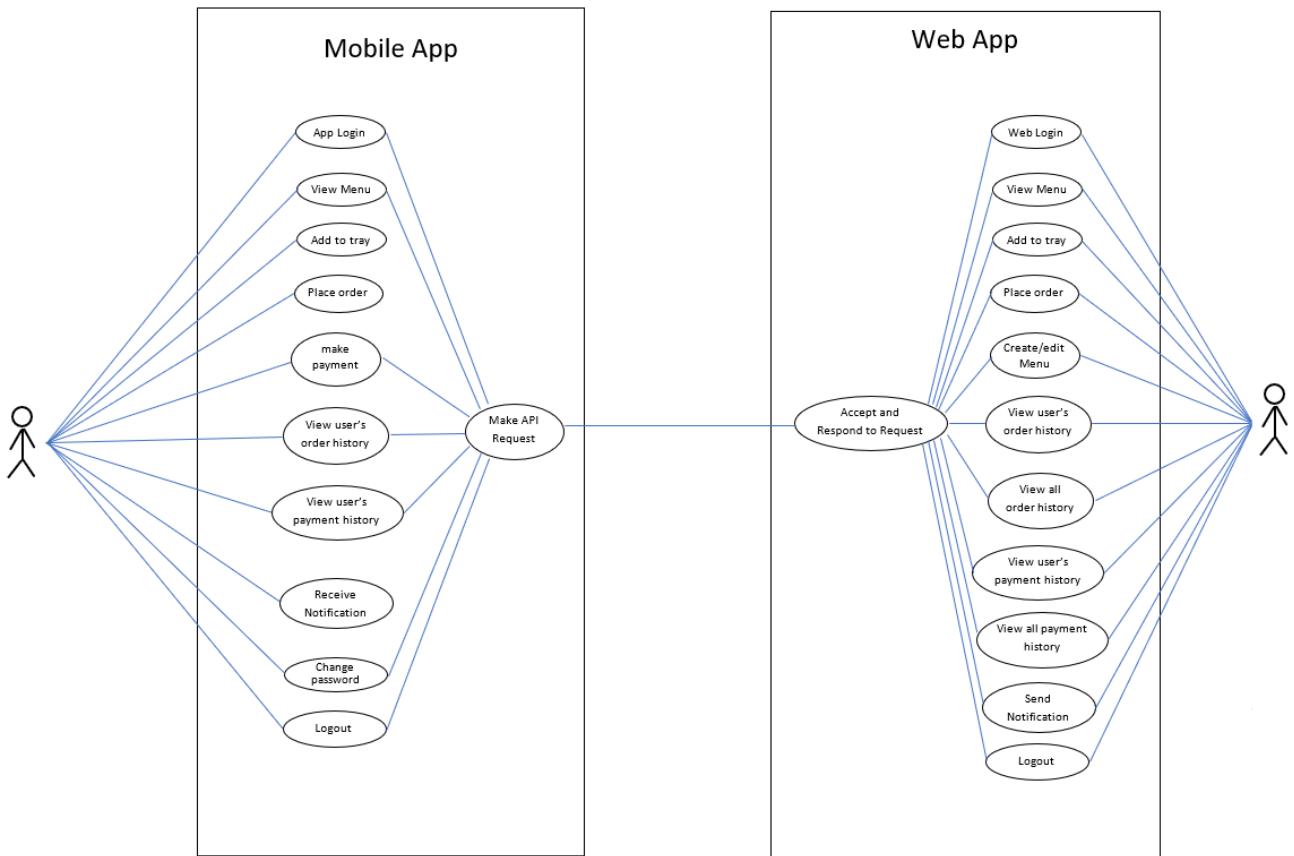


Figure 24: Use Case – Third Iteration

## Data Dictionary

### Users

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	<b>20</b>	<input type="checkbox"/>	<b>AUTO_INCREMENT</b>
2	name	VARCHAR	191	<input type="checkbox"/>	No default
3	email	VARCHAR	191	<input type="checkbox"/>	No default
4	api_token	VARCHAR	191	<input checked="" type="checkbox"/>	NULL
5	password	VARCHAR	191	<input type="checkbox"/>	No default
6	role_id	VARCHAR	191	<input type="checkbox"/>	'1'
7	balance	VARCHAR	191	<input type="checkbox"/>	'0'
8	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
9	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 25: Data Table - Users

### Roles

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>INT</b>	<b>11</b>	<input checked="" type="checkbox"/>	NULL
2	title	VARCHAR	50	<input checked="" type="checkbox"/>	NULL

Figure 26: Data Table - Roles

### Foods

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	<b>20</b>	<input type="checkbox"/>	<b>AUTO_INCREMENT</b>
2	category_food_id	INT	11	<input type="checkbox"/>	No default
3	name	VARCHAR	191	<input type="checkbox"/>	No default
4	served_today	INT	1	<input checked="" type="checkbox"/>	0
5	details	VARCHAR	191	<input type="checkbox"/>	No default
6	image	VARCHAR	150	<input checked="" type="checkbox"/>	NULL
7	price	INT	11	<input type="checkbox"/>	No default
8	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
9	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 27: Data Table - Foods

### Category Food

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	<b>20</b>	<input type="checkbox"/>	<b>AUTO_INCREMENT</b>
2	name	VARCHAR	191	<input type="checkbox"/>	No default
3	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
4	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 28: Data Table – Category Food

## Orders

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	20	<input type="checkbox"/>	<b>AUTO_INCREME...</b>
2	user_id	INT	11	<input checked="" type="checkbox"/>	No default
3	for	DATETIME		<input checked="" type="checkbox"/>	No default
4	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
5	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 29: Data Table - Orders

## Ordered\_items

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	20	<input type="checkbox"/>	<b>AUTO_INCREME...</b>
2	order_id	INT	11	<input checked="" type="checkbox"/>	No default
3	food_id	INT	11	<input checked="" type="checkbox"/>	No default
4	quantity	INT	11	<input checked="" type="checkbox"/>	No default
5	price	INT	11	<input checked="" type="checkbox"/>	NULL
6	status	INT	2	<input checked="" type="checkbox"/>	0
7	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
8	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 30: Data Table – Ordered Item

## Payments

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	20	<input type="checkbox"/>	<b>AUTO_INCREME...</b>
2	user_id	INT	11	<input checked="" type="checkbox"/>	No default
3	pay_method_id	INT	11	<input checked="" type="checkbox"/>	No default
4	order_id	INT	11	<input checked="" type="checkbox"/>	NULL
5	amount	INT	11	<input checked="" type="checkbox"/>	0
6	description	VARCHAR	222	<input checked="" type="checkbox"/>	" "
7	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
8	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 31: Data Table - Payments

## Pay Methods

#	Name	Datatype	Length/Set	Allow NULL	Default
1	<b>id</b>	<b>BIGINT</b>	20	<input type="checkbox"/>	<b>AUTO_INCREME...</b>
2	name	VARCHAR	191	<input checked="" type="checkbox"/>	No default
3	particular	VARCHAR	191	<input checked="" type="checkbox"/>	No default
4	created_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL
5	updated_at	TIMESTAMP		<input checked="" type="checkbox"/>	NULL

Figure 32: Data Table – Pay methods

### 3.6.4.3. IMPLEMENTATION

#### MOBILE APP

The screenshot shows two side-by-side code editors in Visual Studio Code. The left editor contains the `connection.dart` file, which includes code for connecting to a REST API and handling login requests. The right editor contains the `pubspec.yaml` file, listing dependencies such as `flutter`, `http`, `shared_preferences`, `oneSignal_flutter`, and `firebase_messaging`. Below the code editors is a terminal window displaying log output related to the Flutter engine and its connection to the host process.

```

connection.dart
pubspec.yaml

```

Figure 33: Implementation Screenshot (API Connector (left) and flutter packages user(right))

The screenshot shows two side-by-side code editors in Visual Studio Code. The left editor contains the `main.dart` file, which defines the application's entry point and handles navigation between screens. The right editor contains the `navi.dart` file, which implements a navigation state management class (`_NaviState`) and a scaffold for the application. Below the code editors is a terminal window displaying log output related to the Flutter engine and its connection to the host process.

```

main.dart
navi.dart

```

Figure 34: Implementation Screenshot (main.dart (left) and navi.dart (right))

Figure 35 displays the implementation of scoped models and custom libraries. The left side of the screen shows the code for `connection.dart`, `mainModel.dart`, and `mainModelData.dart`. The right side shows the code for `pubspec.yaml`, `nav.dart`, `values.dart`, and `values.dart`. The bottom pane shows the terminal output of the application's log.

```

connection.dart
mainModel.dart
mainModelData.dart
pubspec.yaml
nav.dart
values.dart
values.dart

```

```

I/LED (28335): [G1]_getProcNameProcess pid(28335) ...
I/LED (28335): [G1]_getprocess name(crm.example.fyp_app)
I/mali (28335): [G1] ret(1) stt status(00000000) aniso debug level(0) gt aniso max level(16) ani so mask(00000001) tri mask(00000002)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/mali_winsys (28335): EGLint new_window_surface(egl_winsys_display *, void *, EGLSurface, EGLConfig, egl_winsys_surface **, EGLBoolean) returns 0x3000
I/libEGL (28335): [MIX Game SDI] low_latency_mode(0 pid(-1)) property(-1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/flutterView (28335): Detaching from a FlutterEngine: io.flutter.embedding.engine.FlutterEngine@5a1le7
V/PhoneWindow(28335): DecorView setVisibility visibility = 4, Parent = android.view.ViewRootImpl@60beed, this = DecorView@46a3901[MainActivity]
D/FlutterEngine(28335): Destroying.
D/View (28335): [Warning] assignParent to null: this = DecorView@46a3901[MainActivity]
Application finished.
Exited (sigterm)

```

Figure 35: Implementation Screenshot (Scoped models (left) and custom libraries(right))

Figure 36 displays the implementation of API request modules and navigation logic. The left side of the screen shows the code for `connection.dart`, `mainModel.dart`, and `modelNavigator.dart`. The right side shows the code for `modelNavigator.dart`. The bottom pane shows the terminal output of the application's log.

```

connection.dart
mainModel.dart
modelNavigator.dart
pubspec.yaml
nav.dart
values.dart
values.dart

```

```

I/LED (28335): [G1]_getProcNameProcess pid(28335) ...
I/LED (28335): [G1]_getprocess name(crm.example.fyp_app)
I/mali (28335): [G1] ret(1) stt status(00000000) aniso debug level(0) gt aniso max level(16) ani so mask(00000001) tri mask(00000002)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/mali_winsys (28335): EGLint new_window_surface(egl_winsys_display *, void *, EGLSurface, EGLConfig, egl_winsys_surface **, EGLBoolean) returns 0x3000
I/libEGL (28335): [MIX Game SDI] low_latency_mode(0 pid(-1)) property(-1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/Surface (28335): Surface::connect(this=0x7041980000,api1)
D/flutterView (28335): Detaching from a FlutterEngine: io.flutter.embedding.engine.FlutterEngine@5a1le7
V/PhoneWindow(28335): DecorView setVisibility visibility = 4, Parent = android.view.ViewRootImpl@60beed, this = DecorView@46a3901[MainActivity]
D/FlutterEngine(28335): Destroying.
D/View (28335): [Warning] assignParent to null: this = DecorView@46a3901[MainActivity]
Application finished.
Exited (sigterm)

```

Figure 36: Implementation Screenshot (API Request modules (left) and navigation logic(right))

```
File Edit Selection View Go Run Terminal Help
```

```
modelPayment.dart - lib/main.dart - Visual Studio Code
```

```
EXPLORER
```

```
OPENED FOLDERS
```

```
connection.dart lib/helper  
modelTray.dart lib/scopes  
modelOrders.dart lib/scopes  
main.dart.lib
```

```
GROUP 2
```

```
! pubspec.yaml  
nav.dart lib/pages/nav  
values.dart lib/values
```

```
xFP_APP
```

```
> orders  
> payment  
> settings  
> tray  
splash.dart  
scopes  
  mainModel.dart  
  modeConnector.dart  
  modeHome.dart  
  modeLogin.dart  
  modeNavigator.dart  
  modeOrders.dart  
  modePayment.dart  
  modeSettings.dart  
  modeSplash.dart  
  modeTray.dart  
values  
  borders.dart  
  colors.dart  
  radii.dart  
  shadows.dart  
  sizes.dart  
  styles.dart  
  values.dart
```

```
main.dart
```

```
test
```

```
flutter.pubspec  
flutter.plugins-dependencies
```

```
.gitignore  
.metadata  
packages
```

```
OUTLINE
```

```
TIMELINE
```

```
DEPENDENCIES
```

```
PROBLEMS
```

```
LIBRARIES
```

```
connection.dart lib/orders > ...  
1 import 'package:fp/app/helper/connection.dart';  
2 import 'package:fp/app/models/order.dart';  
3 import 'package:fp/app/models/orderItem.dart';  
4 import 'package:fp/app/models/modelConnector.dart';  
5 import 'package:fp/app/libraries/modelConnector.dart';  
6 import 'package:fp/app/pages/orders/components/timelineOrder.dart';  
7  
8 mixin ModelOrders on ModelConnector{  
9   bool _orderLoading = false;  
10  List<Order> _orderedItems = [];  
11  List<TimelineModel> _orderTimeline = [];  
12  
13  orderTimelineModel()=>_orderTimeline;  
14  
15  orderLoading()=>_orderLoading;  
16  setOrderLoading(bool t){  
17    _orderLoading = t;  
18    notifyListeners();  
19  }  
20  
21  orderedItems()=>_orderedItems;  
22  setOrderedItems(List<Order> o){  
23    _orderedItems = o;  
24    notifyListeners();  
25  }  
26  
27  updateOrders()async{  
28    setOrderLoading(true);  
29    Connector().getOrderHistory().then((data){  
30      print(data);  
31      List<Order> retrieved = data.map<Order>(d){  
32        return Order(  
33          id: d['id'],  
34          fo: d['fo'],  
35          ordered: d['ordered'].mapOrderItem()(oi){  
36            return OrderItem(  
37              id: oi['id'],  
38              name: oi['food']['name'],  
39              price: oi['price']/1,  
40              quantity: oi['quantity'],  
41              status: oi['status'],  
42              image: oi['image']  
43            }  
44          }  
45        );  
46      }).toList();  
47      setTransactions(retrieved);  
48      updateTimeLine();  
49    });  
50  }  
51  
52  updateTimeLine(){  
53    List<TimelineModel> tm = transactions().map<TimelineElement>((Tra  
54      return TimelineElement(t);  
55    )).toList();  
56    setTM(tm);  
57    setPayLoading(false);  
58    notifyListeners();  
59  }
```

```
D/FlutterEngine(28335): Destroying.  
D/FlutterEnginePluginRegistry(28335): Destroying.  
D/View (28335): [Warning] assignParent to null: this = DecorView@46a3901[MainActivity]  
Application finished.  
Exited (sigterm)
```

```
Ln 64 Col 1 Species: 2 UTF-8 CR/LF Dart: Flutter: 1.12.13+hotfix.9 No Device
```

Figure 37: Implementation Screenshot (Orders Logic(left) and Payment logic(right))

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "FP APP".
- Code Editor:** Displays the main.dart file with the following code:

```
lib> pages> home > menu.dart ...  
1 import 'package:flutter/material.dart';  
2 import 'package:fp_app/components/linkedText.dart';  
3 import 'package:fp_app/components/txt.dart';  
4 import 'package:fp_app/libraries/helper/connection.dart';  
5 import 'package:fp_app/scopes/menuCategory.dart';  
6 import 'package:fp_app/pages/home/components/menu.dart';  
7 import 'package:fp_app/pages/home/components/recommendations.dart';  
8 import 'package:fp_app/scopes/mainModel.dart';  
9 import 'package:fp_app/scopes/modelHome.dart';  
10 import 'package:fp_app/values/colors.dart';  
11 import 'package:scoped_model/scoped_model.dart';  
12  
class Home extends StatelessWidget {  
13  
  @override  
14    Widget build(BuildContext context) {  
15      return ScopedModelDescendant<MainModel>(  
16        builder: (_, child, ModelHome model){  
17          if(!model.homeLoaded){  
18            model.updateData();  
19            List<Widget> menuTabs = [];  
20            for (MenuItem mc in model.menuItems) {  
21              menuTabs.add(Text(mc.name));  
22            }  
23            return Scaffold(  
24              backgroundColor: AppColors.primaryBackground,  
25              body: Container(  
26                padding: EdgeInsets.only(left: 20),  
27                child: Column(  
28                  crossAxisAlignment: CrossAxisAlignmentAlignment.start,  
29                  children: [  
30                    Container(  
31                      child: FlatButton(  
32                        splashColor: Colors.transparent,  
33                        highlightColor: Colors.transparent,  
34                        unpressed: () {  
35                          model.toggleRecommend();  
36                        },  
37                        child: Row(  
38                          children: [  
39                            Txt('Try our Recommendations', type: 'thin-  
40                            Space(),  
41                          ],  
42                        ],  
43                      ),  
44                    ]  
45                  ]  
46                )  
47              )  
48            );  
49          }  
50        }  
51      );  
52    }  
53  }  
54  
55  class Menu extends Container{  
56    Menu(MainModel model):super(  
57      child: Expanded(  
58        child: DefaultTabController(  
59          length: model.menu.length,  
60          child: Column(  
61            crossAxisAlignment: CrossAxisAlignmentAlignment.start,  
62            children: [  
63              Container(  
64                height: 28,  
65                margin: EdgeInsets.only(bottom: 10),  
66                child: TabBar(  
67                  isScrollable: true,  
68                  tabs: model.menu.map(Widget((m)=>  
69                    Tab(  
70                      child: Txt(m.name, type: 'small-secondary-header'),  
71                    )),).toList() // Tab  
72                  ), // TabBar  
73                ), // Container  
74                Expanded(  
75                  child: TabBarView(  
76                    controller: model.menu.map(Widget((m)=>  
77                      TabBarView(

```

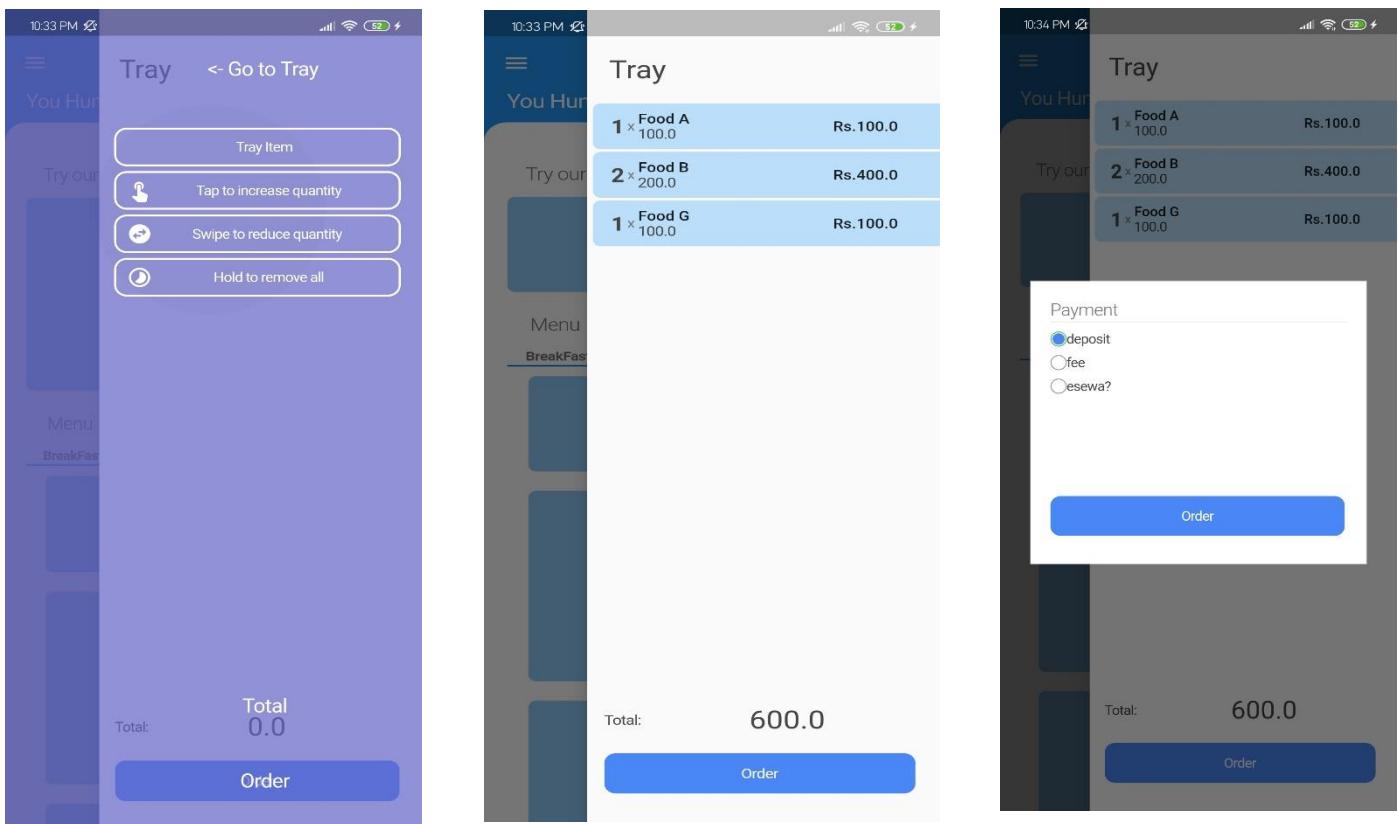
- Terminal:** Shows the output of the Flutter Engine and View destruction.
- Bottom Status Bar:** Includes icons for file operations like Open, Save, Find, and Timeline.

Figure 38: Implementation Screenshot (home widget(left) and menu component(right))

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** On the left, it lists files and folders. Opened files include `mainOrders.dart`, `modelOrders.dart`, `tray.dart`, `main.dart`, `order.dart`, `subpage.dart`, `values.dart`, `modelPayment.dart`, and `orders.dart`. Other visible files include `menuTemplate.dart`, `recommend.dart`, `home.dart`, `login.dart`, `navi.dart`, `components/homeInstruction.dart`, `components/homeOrdering.dart`, `components/instructionTrayItem.dart`, `menuDrawer.dart`, `trayDrawer.dart`, `TrayInstruction.dart`, `navi.dart`, `orders.dart`, `components/tray.dart`, `spaced.dart`, `modelMainModel.dart`, `modelConnector.dart`, `modelHomeModel.dart`, `modelLogin.dart`, `modelNavigation.dart`, `modelOrders.dart`, `modelPayment.dart`, and `modelSettings.dart`.
- Code Editor:** The main area displays the `main.dart` file. It contains code for a `Tray` stateless widget that builds a `Container` with a `Column` of `TrayItem`s. Each `TrayItem` is a `SingleChildScrollView` containing a `Row` with a `Text` and a `Spacer`. The `mainAxisAlignment` is set to `MainAxisAlignment.spaceEvenly`. The `model` variable is used throughout the code.
- Terminal:** At the bottom, the terminal shows the output of the Flutter engine destroying the application and the application finishing.

Figure 39: Implementation Screenshot (tray widget (left) and orders widget(right))



*Figure 40: Implementation Screenshot (from left: tray instruction, tray drawer, payment before order )*

Web

Figure 41: Implementation Screenshot (React routes (left) and API Routes(right))

The screenshot shows a developer's workspace in Visual Studio Code with several tabs open across different panes:

- routes.js**: A file containing route definitions for a Node.js application.
- package.json**: The project's package configuration.
- apicontrollers.php**, **actions.js**, **index.js**, and **action-types.js**: PHP files representing the API logic.
- routes**: A folder containing resources for routes.
- store**: A folder containing reducers, actions, and persistStore files.
- index.js**: The main entry point for the application.
- Settings**: The settings file.
- Stats**: The stats file.
- Transaction**: The transaction file.
- User**: The user file.
- Users**: The users file.
- action-types**: The action types file.
- actions**: The actions file.
- history**: The history file.
- index**: The index file.
- index.html**: The main HTML file.
- bootstrap.js**: The bootstrap file.
- helpers.js**: The helpers file.
- lang**: Language files.
- views**: View components.
- indexblade.php**: The main blade template.

At the bottom, a terminal window is open in PowerShell with the command `dotnet run` and the output "Compiled successfully in 1428ms".

Figure 42: Implementation Screenshot (NPM packages (left) and Reducers created for REDUX (right))

```

// actions.js
45 // Send mail to the email to reset password
46 export function resetPassword(credentials) {
47   return (dispatch) => {
48     dispatch({
49       type: ActionTypes.RESET_PASSWORD,
50       payload: true
51     });
52     authService.resetPassword(credentials).then((res) => {
53       success("Check Your Email!");
54       dispatch({
55         type: ActionTypes.RESET_PASSWORD,
56         payload: false
57       });
58     }).catch((err) => {
59       warn("Try Again");
60       dispatch({
61         type: ActionTypes.RESET_PASSWORD,
62         payload: false
63       });
64     });
65   }
66 }

// update password
67 export function updatePassword(credentials) {
68   return (dispatch) => {
69     authService.updatePassword(credentials).then(response => {
70       success(response.data.message);
71       history.push("/login");
72     }).catch((response) => {
73       dispatch({
74         type: ActionTypes.UPDATE_PASSWORD,
75         payload: response.data.errors
76       });
77       warn("Invalid Credentials");
78     });
79   }
80 }

81 export function logout() {
82   return {
83     type: ActionTypes.AUTH_LOGOUT,
84   }
85 }

86 
```

```

// api.php
1 import { Http } from './Http';
2 import { storeOrders } from '../store/actions';
3
4 export function getStats() {
5   return http.get('/api/v1/stats');
6 }
7
8 export function getCategories() {
9   return http.get('/api/v1/categories/');
10 }
11
12 export function insertCategories(details) {
13   return http.post('/api/v1/category/insert', details);
14 }
15
16 export function getFood() {
17   return http.get('/api/v1/food/');
18 }
19
20 export function setFood(details) {
21   return http.post('/api/v1/food/create', details);
22 }
23
24 export function updateFood(id, status) {
25   return http.post('/api/v1/food/update', { id:id, status:status });
26 }
27
28 export function getOrders() {
29   return http.get('/api/v1/orders/');
30 }
31
32 export function getUserOrders(id) {
33   return http.get('/api/v1/orders/' + id);
34 }
35
36 export function getOrderHistory() {
37   return http.get('/api/v1/orders/all');
38 }
39
40
41 export function storeOrder(order) {
42   return http.post('/api/v1/orders/store', { orders:order });
43 }


```

PS E:\Workspace\Cart> [ ]

Figure 43: Implementation Screenshot (actions (left) and API requesting services(right))

The screenshot shows the Chrome browser with the URL "Not secure | cant.test". The main content area displays a dashboard with two charts: "Today's Top Sellers" and "Weekly Sales Report". Below the charts is a sidebar with a grid of small icons. At the bottom of the browser window, the DevTools interface is visible, specifically the "Redux" tab of the "Inspector" panel. The state tree shows the following structure:

```

filter...
HOME_LOADING
FOOD_LOADING
FOOD_LOADING
STATS_TOP
STATS_WEEKLY
STATS_LOADING
HOME_ORDERS
FOOD_MENU

```

The state tree pane shows the current state of the application, including:

- Auth**: { user: ..., isAuthenticated: true, loading: false, ... }
- Home**: { state: { orders: [], search: "", showMenu: false, ... } }
- Stats**: { top: { state: { ... } }, weekly: { state: { ... } }, loading: false, errors: [] }
- Food**: { state: { food: [], menu: [], categories: [] }, ... }

Figure 44: Implementation Screenshot – Home Page statistics with REDUX states

The screenshot shows a web application interface for managing transactions. At the top, there's a navigation bar with links for Home, Menu, Orders, Transactions, and Users. A dropdown menu indicates the user is 'admin'. Below the navigation is a table titled 'Transactions' with columns: ID, Amount, User, Method, Description, and Date. The table contains three rows of transaction data.

ID	Amount	User	Method	Description	Date
41	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 91	2020-04-19 16:50:39
40	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 90	2020-04-19 16:49:36
39	Rs. 120			cash payment at counter	2020-04-19 16:45:51

At the bottom of the screen, a Redux DevTools sidebar is visible, showing the current state of the application. The 'State' tab is selected, displaying a tree view of the Redux store with various actions like HOME\_LOADING, FOOD\_LOADING, and TRANSACTION\_TRANSACTIONS, along with their corresponding payload data.

Figure 45: Implementation Screenshot – Transactions with REDUX states

This screenshot shows a user detail page. The top navigation bar includes links for Home, Menu, Orders, Transactions, and Users, with 'admin' selected. The main content area displays a user profile for 'b@b.b' with a balance of 'Rs.0'. Below this is a table titled 'Transaction' with columns: ID, Amount, Description, Date, and Orders. The table lists two transactions: one for Rs. 100 (order id 91) and another for Rs. 100 (order id 90). The 'Orders' column for each transaction shows the details of the food items ordered.

ID	Amount	Description	Date	Orders
41	Rs. 100	food ordered via app; order id 91 - order id 91	2020-04-19 16:50:39	order ID: 91 1* Food G 2* Food B @ Rs.100
40	Rs. 100	food ordered via app; order id 90 - order id 90	2020-04-19 16:49:36	order ID: 90 1* Food A @ Rs.100

Similar to Figure 45, a Redux DevTools sidebar is present at the bottom, showing the state of the application, specifically focusing on the 'User' and 'transaction' slices.

Figure 46: Implementation Screenshot – User Detail with REDUX states

## CHAPTER 4: TESTING AND ANALYSIS

### 4.1 TEST PLAN

#### 4.1.1 MANUAL INCREMENT TEST PLAN

Increment test is a type of unit testing where the new modules are tested when integrating with the main project. Each Increment is tested after adding to the project. Some of the test cases and their results are then documented and recorded in the titles below.

#### 4.1.2 MANUAL SYSTEM TEST PLAN

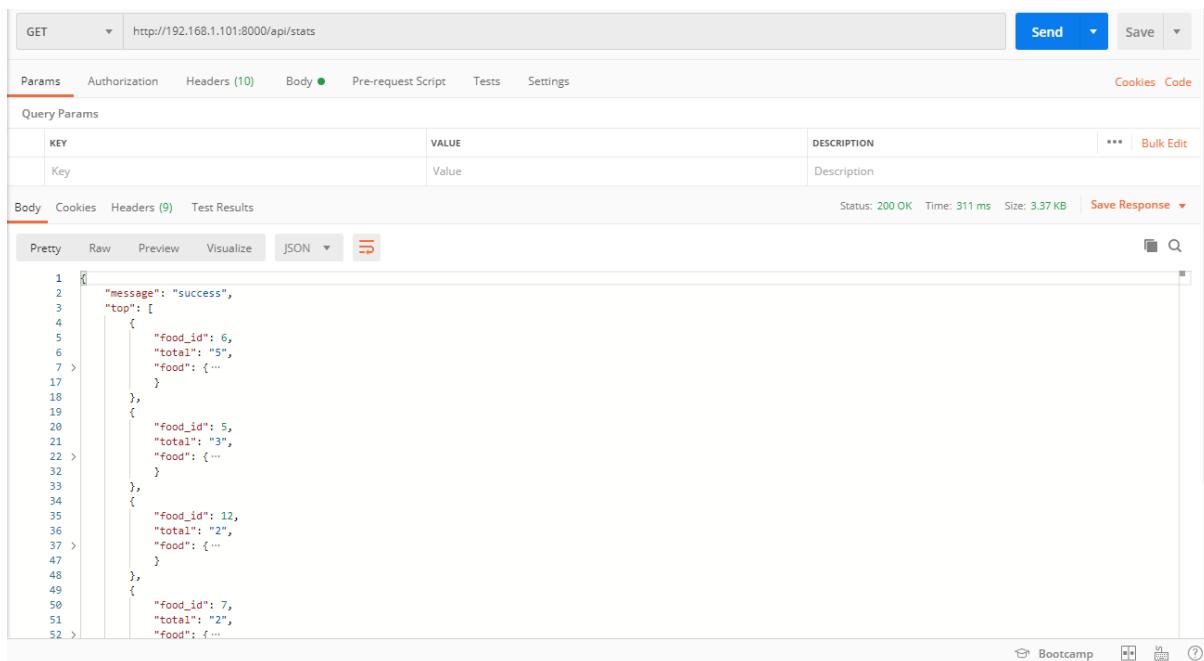
System Testing is a test to check the entire system functions as intended after completion. The system testing can be categorized into two groups, black box testing and white box testing. White box testing refers to the tests conducted with the code of the project by someone familiar with it. In contrast, black box testing is checks the validity of functions usually by someone without the knowledge of the inner workings of the project.

For this project the system will be tested in a white box scenario and the results will be documented in the title below.

## 4.2 MANUAL INCREMENT TESTING

### 4.2.1 Statistics API

- Platform : Web App  
Added Increment: Statistics API  
Action: the API request will be sent via post man app  
Expected outcome: the server should respond with appropriate data.  
Actual outcome: server returned list of top and weekly sales  
Result: Success



The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET', a dropdown for 'Authorization', and a URL input field containing 'http://192.168.1.101:8000/api/stats'. To the right of the URL are 'Send' and 'Save' buttons. Below the header is a navigation bar with tabs: 'Params', 'Authorization', 'Headers (10)', 'Body' (which is currently selected and highlighted in green), 'Pre-request Script', 'Tests', and 'Settings'. On the far right of the navigation bar are 'Cookies' and 'Code' buttons. Under the 'Body' tab, there is a table titled 'Query Params' with columns 'KEY', 'VALUE', and 'DESCRIPTION'. A single row is present with 'Key' in the KEY column and 'Value' in the VALUE column. Below the table, there are tabs for 'Body', 'Cookies', 'Headers (9)', and 'Test Results'. The 'Body' tab is active. To the right of these tabs, status information is displayed: 'Status: 200 OK', 'Time: 311 ms', 'Size: 3.37 KB', and 'Save Response'. At the bottom of the interface, there are several small icons: a mail icon, 'Bootcamp', a square icon, a 'ln' icon, and a help icon.

```
1 {
2   "message": "success",
3   "top": [
4     {
5       "food_id": 6,
6       "total": "5",
7       "food": { ...
8     },
9     {
10       "food_id": 5,
11       "total": "3",
12       "food": { ...
13     },
14     {
15       "food_id": 12,
16       "total": "2",
17       "food": { ...
18     },
19     {
20       "food_id": 7,
21       "total": "2",
22       "food": { ...
23     }
24   ]
25 }
```

Figure 47: Test – Statistics API test

#### 4.2.2 Test 2: Fire base Implementation

Platform : Mobile App  
Added Increment: Firebase  
Action: firebase json file was added to the project and build initiated  
Expected outcome: the project should build without error  
Actual outcome: error occurred

```
Launching lib\main.dart on Android SDK built for x86 in debug mode...
[!] Your app isn't using AndroidX.
    To avoid potential build failures, you can quickly migrate your app by following the steps on https://goo.gl/CP92wY.
```

Figure 48: Test – error case

Possible solution: Android migration and use of multidex

```
4
5     subprojects {
6         project.configurations.all {
7             resolutionStrategy.eachDependency { details -
8                 if (details.requested.group == 'com.android.support'
9                     && !details.requested.name.contains('multidex') ) {
10                     details.useVersion "27.1.1"
11                 }
12                 if (details.requested.group == 'androidx.core'
13                     && !details.requested.name.contains('androidx') ) {
14                     details.useVersion "1.0.1"
15                 }
16             }
17         }
18     }
19 }
```

Figure 49: Test – fix

Expected outcome: the project should build without error

Actual outcome: project built without errors

Result: Success

```
37     shared_preferences: ^0.5.4+5
38
39     firebase_messaging: ^6.0.13
40     onesignal_flutter: ^2.4.1
41
42     image_picker: ^0.6.7
43
44 
```

Figure 50: Test – FCM package for flutter

```
PROBLEMS 33 OUTPUT DEBUG CONSOLE TERMINAL
Launching lib\main.dart on Redmi Note 8 Pro in debug mode...
✓ Built build\app\outputs\apk\debug\app-debug.apk.
Connecting to VM Service at ws://127.0.0.1:1580/Yih_0lMaSro=/ws
```

Figure 51: Test – Build Success

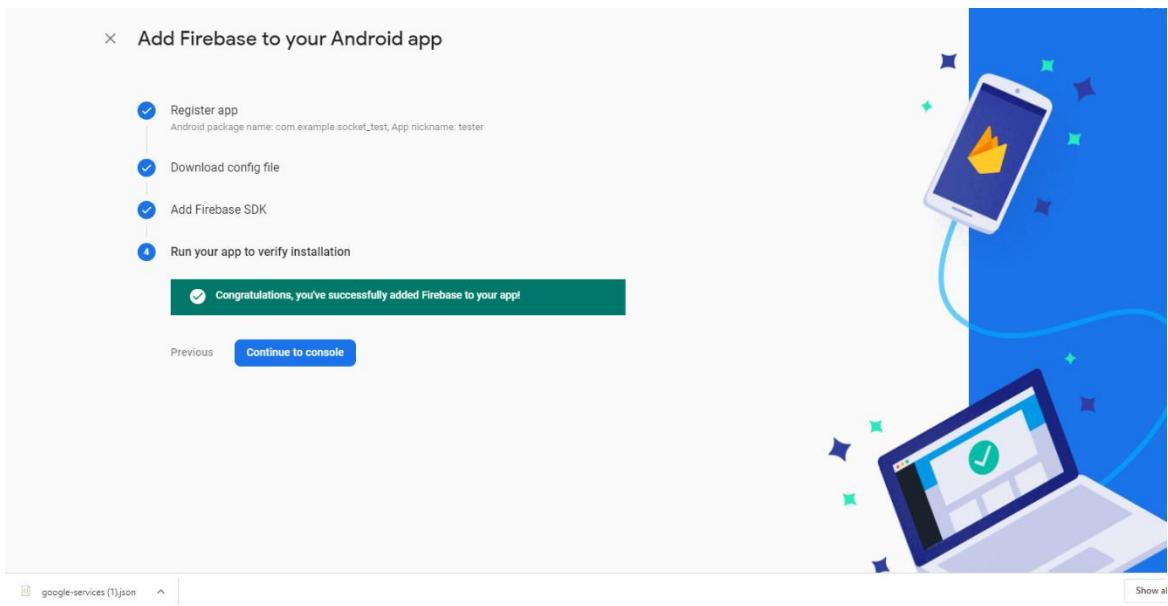


Figure 52: Test – FCM implementation

#### 4.2.3 Test 3: Notification

Platform : Web and Mobile  
 Added Increment: Notification  
 Action: the button to respond to orders will be clicked  
 Expected outcome: the web should send notification to mobile app.  
 Actual outcome: Mobile app received notification  
 Result: Success

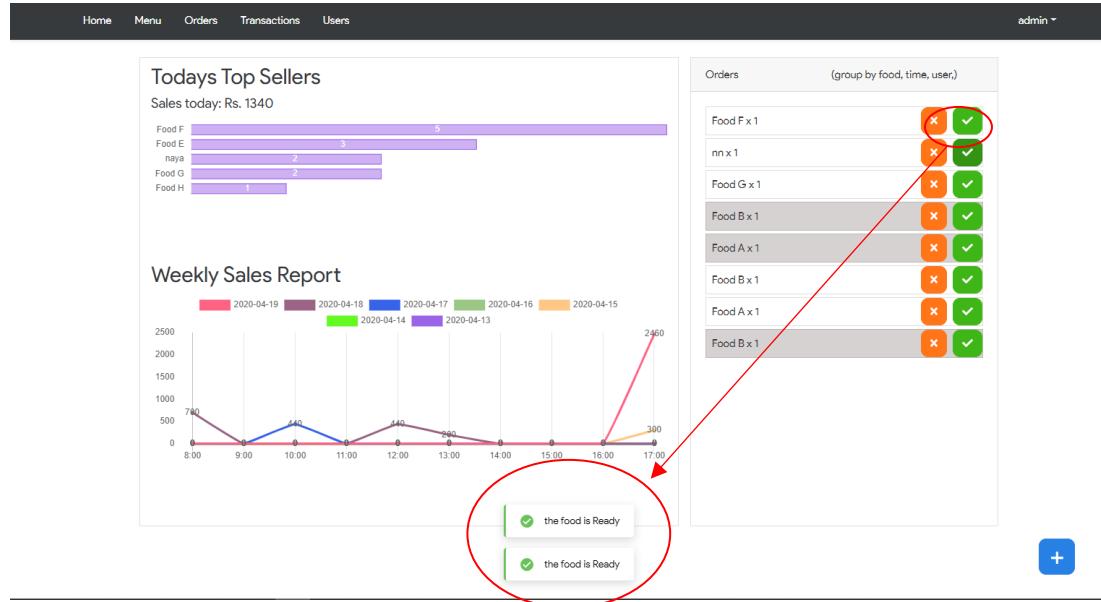


Figure 53: Test – changing order status

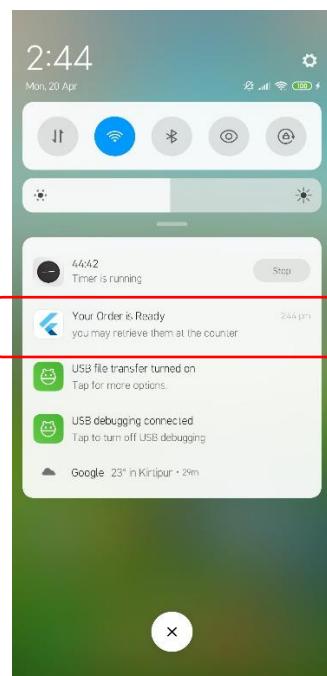


Figure 54: Test - mobile notification

#### 4.2.3 Test 3: Instruction Overlays

Platform : mobile App  
Added Increment: user tutorials  
Action: mobile Application is executed without any prior cache files  
Expected outcome: the tutorial overlays should display at appropriate pages.  
Actual outcome: the overlays appeared as they should  
Result: Success

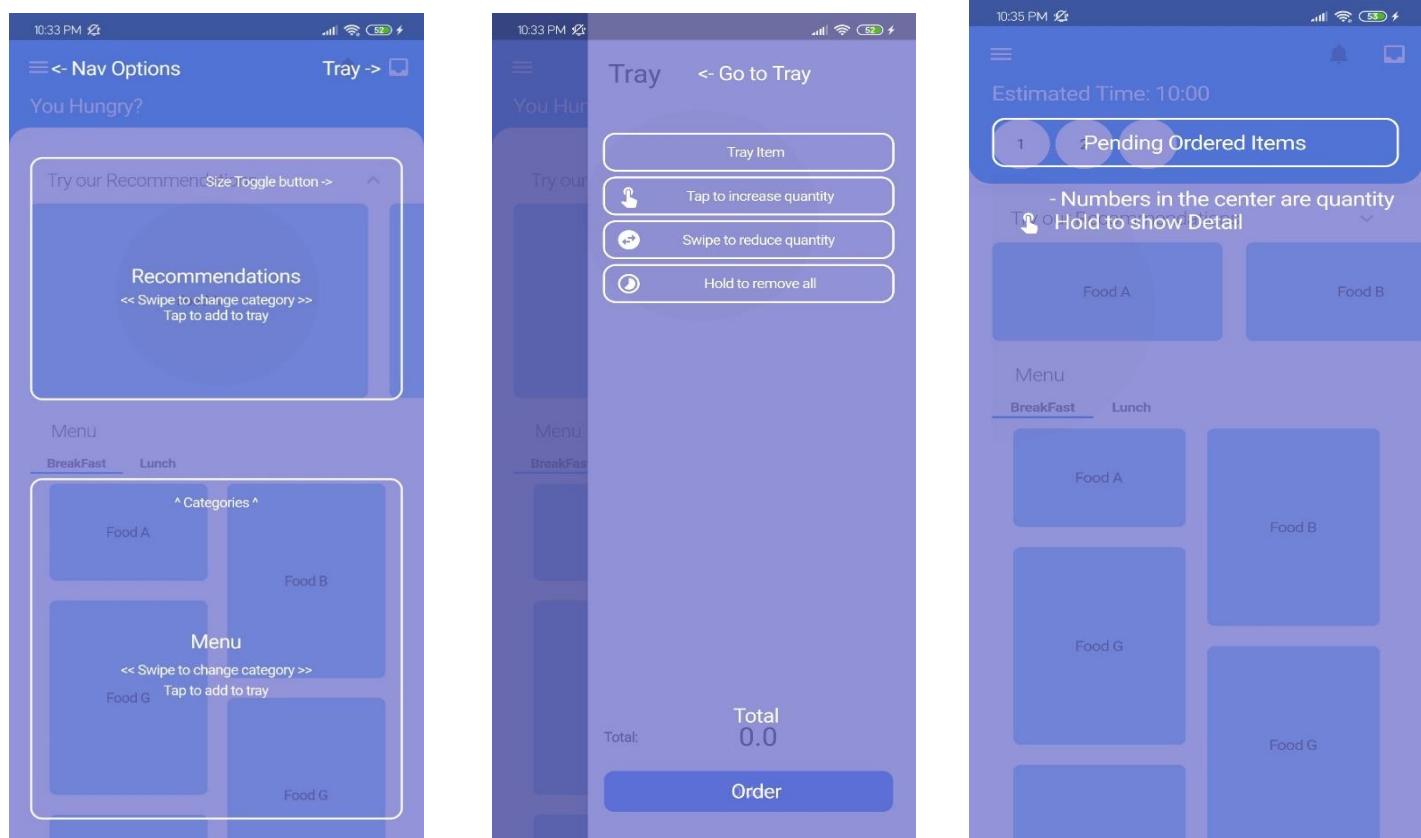


Figure 55: Test - mobile Instruction Overlay

#### 4.2.4 System Test 4: Tutorial Repetition

Test: Tutorial Repetition  
Added Increment: user tutorials  
Action: Mobile application will be restarted in succession  
Expected outcome: the tutorials should not display more than once  
Actual outcome: the tutorials appeared only once  
Result: Success

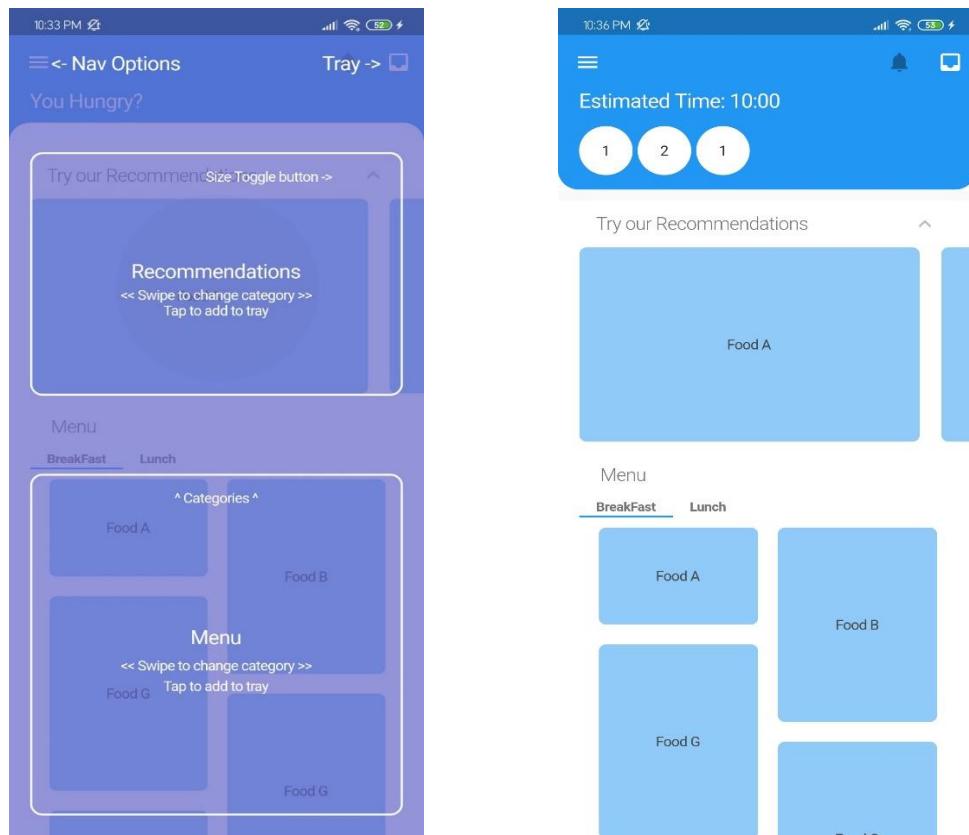


Figure 56: Test – Tutorial repetition

## 4.3 SYSTEM TESTING

### 4.3.1 System Test 1: Disconnection handling

Test: disconnection handling  
Platform : Mobile and Web  
Action: server will be disconnected during user interacting with app  
Expected outcome: the app should not crash.  
Actual outcome: the app did not crash and showed appropriate message  
Result: Success

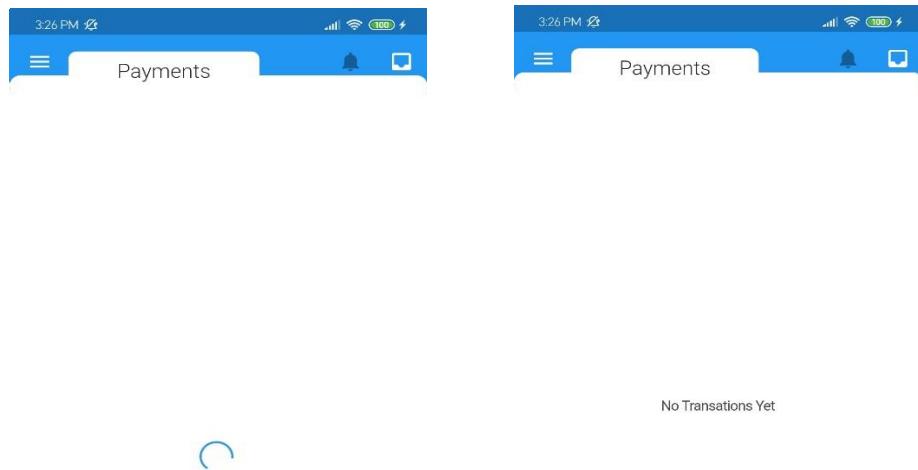


Figure 57: Test - disconnection

#### 4.3.2 System Test 2: Menu Update

Test: Tutorial Repetition  
Platform : Web and Mobile App  
Action: the check box button to update menu is clicked  
Expected outcome: the menu should be updated in web and mobile  
Actual outcome: the menu was updated  
Result: Success

The image consists of two screenshots of a web-based menu management application. Both screenshots show a navigation bar at the top with links: Home, Menu, Orders, Transactions, and Users. On the far right of the top bar, it says "admin".

**Screenshot 1 (Top):** This screenshot shows the "Menu" section. It has two main sections: "BreakFast" and "Lunch". Under "BreakFast", there are several items with checkboxes: Food A 100 (checked), Food B 200 (checked), Food G 100 (checked), Food G 100 (unchecked), nn 100 (checked), and naya 120 (checked). Under "Lunch", there are several items with checkboxes: Food C 100 (checked), Food D 100 (checked), Food E 100 (checked), Food F 100 (checked), Food H 100 (unchecked), and Food H 100 (checked). To the right of the menu list is a "Create Food" form. It includes fields for Name (empty), Category (a dropdown menu showing "Select..."), Price (empty), and Description (empty). Below these fields is a blue "Create" button.

**Screenshot 2 (Bottom):** This screenshot shows the same "Menu" section as the first one. The "BreakFast" section remains the same. In the "Lunch" section, the checkbox for "Food G 100" is now checked. To the right of the menu list is a "Create Food" form. It includes fields for Name (empty), Category (a dropdown menu showing "Select..."), Price (empty), and Description (empty). Below these fields is a blue "Create" button. At the bottom of the page, there is a green success message box containing the text "Food G is on the menu".

Figure 58: Test – Menu Update

#### 4.4 CRITICAL ANALYSIS

The system described in the proposal and the actual project while bearing many similarities still differ in some areas. Throughout the development process, the design and schema can be visibly seen growing however in some cases, features had to be laid off due to practicality, feasibility or inaccessibility reasons. The system is made dynamic as much as possible with appropriate use of state management techniques and data retrievers.

Looking back, during the project, especially at the early phases, a lot of mistakes were made based on wrong assumptions and not much time was allotted for mishaps and fallback while planning the project. The project was a medium to demonstrate the knowledge as well as to learn new ideas. Therefore, I was also researching and practically testing different state managers like Redux, pragmatic and scoped models on each iteration of the progress. On the hindsight, it was a naïve mistake on my part. Not only did implementing untested methods make the project unstable, it also cost a lot of time. However, having used the iterative incremental methodology, accommodating the issues into the development lifecycle was somewhat viable. Nonetheless, slightly more research could have avoided the issue.

While the system is completely in working condition, the visual aspect is still questionable. It was designed to be intuitive with user experience in mind, the finer details to visually appeal to the users still needs some work. That said the project encompasses all the major functionalities and then some.

## CHAPTER 5: CONCLUSION

### 5.2 ADVANTAGES

The application has plenty of advantages over other similar applications ranging from implementations never seen before to slight nuances that helps the user experience.

**Multi-platform** – The foremost, the mobile application is developed in flutter, a multiplatform programming framework for dart programming language. Therefore, in theory the application should work for IOS as well as android.

**Multiuser support** - On the web portal, the users can place new orders without obstructing the list of the pending orders. This should allow the chefs to view the items to be prepared directly from the web application without having to wait when a new order is placed.

**Statistics** – When the order panel is not in use on the web portal, which will be very often once the use of mobile application takes effect, the user can view the overall statistics of the sales. Based on these statistics, the system generates recommendations for the mobile and web application.

**Active request listeners** - The web applications have active listeners for new requests and immediately updates when new requests are received.

**Notification** – When the items are ready, notification can be sent to the user letting them know of the status of the food.

**Menu Handling** – the menu is divided into two groups which the users do not see, the all food items list and the list that are available on the menu. Staff can easily create or edit the menu item from the web portal. In addition, adding or removing the existing item to available menu is as easy as clicking the checkbox.

**User logs** – Each user history, order and transaction, are recorded into the system. The staff can view all records or specific logs based on the user.

**Mobile Action** – The mobile and web combination removes most of the restriction prevailing in systems with only web application. The users can view food menu and order without having to physically visit the counter. When the food is ready or cancelled, the user can be notified even when they are not in the vicinity.

**Dynamically Accessible Tray** – on the mobile side, the users will have a dynamic easily accessible tray from any of the pages. This allows the users to view their orders without having to navigate to the specific page for orders.

**Intuitive UI** – The user interface is very intuitive and easy to use. In case the user still has difficulties, the application provides on screen User instructions.

## 5.4 LIMITATIONS

**Spamming Concern** – Since users can order through their phones, there is a possibility that they will do so continuously or from off-site locations with no intention of actually receiving the order. For instance, a user might place an order of ten coffee while they are nowhere near the cafeteria and cancel one the food is prepared. Hopefully, these scenarios are expected to be limited once the payment system and possibly geofencing is implemented.

**Bill of purchase** – While each order details can be viewed in the web as well as mobile application, as of now there is no implementation to print out the physical paper bill of purchase. However, it is one of the top priorities in the upcoming future works.

**Order States** – Currently the mobile application, upon restarting, can load the pending orders from the local memory and checks them against the database servers. But since the statuses of the order items are updated via the notifications the status themselves are not stored locally. So, if some of the items in an order are already received it is updated much later in the application.

**Security** is always an issue when it comes to online application, especially when transactions are concerned. While the system has CSRF token security provided by Laravel framework, much detailed security is needed if the application is to be implemented

## 5.3 FUTURE WORK

**Grouping** - One of the original intentions since the initiation that never came to fruition, the list of pending order could have an option to group the same items and just display the name and total quantity. This would allow the chefs to see the amount of particular dishes to be prepared and plan accordingly

**Meal plans** – Currently the menu only offers items individually. Items that are often ordered together can be placed in meal plans to make it easier for the user to order.

**Validation on pickup** – In future the system will need a verification method to identify the user as the one who ordered during food pickup. I was hoping to implement something similar to QR scans, but further research is necessary.

**Custom order** – In the real world, sometimes people make custom orders that are not available on the menu. This can be replicated in the application with text fields maybe and someone to acknowledge the feasibility of the request.

**Better recommendation algorithm** – the algorithm currently present recommends the most ordered within the week. This can be improved if time of the day and year are taken into factor while generating recommendations.

**User interface** – I still lament not having enough time to implement a better visual user interface. Had there been more time, I was planning on custom designing the icons for the mobile application. As for the web application front. A post application survey can be issued to smooth out the kinks in the system and a new design can be implemented.

**Food Delegation** – Consider a scenario where a user has ordered an item but cancelled once the food is prepared. Currently the case can be handled in one of two ways. First, not to allow the user to cancel but that would be poor customer service. Second, allow to user to cancel, wait for someone else to order the item and send it their way. The second option is more viable but if another user does not order the said item, it goes to waste.

As a solution, the item can be placed on the recommendations, increasing the chance of it being ordered. When the item is ordered, it will be automatically delegated notifying the staff and the customer.

**Payment method** – the current payment method only has only two working option, the customers can either pay directly at the counter or run a debit/credit account with the cafeteria itself. In the future the options can be increased with different online payment options or

## CHAPTER 6: REFERENCES

Bhoj, 2019. *Bhoj App*

Available at:

<https://play.google.com/store/apps/details?id=com.app.bhojdeals&hl=ne>.

Bhoj, 2019. *Bhoj deals*. [Online]

Available at: <https://www.bhojdeals.com/>

[Accessed 2019].

FoodMandu pvt Ltd, 2018. *FoodMandu*. [Online]

Available at: <https://foodmandu.com/>

[Accessed Oct 15 2019].

FoodMario, 2019. *FoodMario Customer App*. Kathmandu: Food Mario.

MealNepal, 2019. *Online food delivery in Nepal: Top 5 services | Meal Nepal*. [Online]

[Accessed 15 Oct 2019].

Press, C. U., 2009. *Evaluation in information | Stanford Edu*. [Online]

Available at: <https://nlp.stanford.edu/IR-book/pdf/08eval.pdf>

[Accessed 2019].

React Chart, 2019. *git hub*. [Online]

Available at: <https://github.com/jerairrest/react-chartjs-2/blob/master/example/src/components/line.js>

[Accessed 2019].

## CHAPTER 7: BIBLIOGRAPHY

Chapple, M., 2020. *Database Normalization Basics*. [Online]

Available at: [About.com](#)

[Accessed 2020].

Code Academy, 2019. *Code Academy*. [Online]

Available at: <https://www.codecademy.com/courses/learn-vue-js/lessons/vue-introduction/exercises/front-end-frameworks>

[Accessed 2019].

Editors of guru, 2020. *Database Normalization*. [Online]

Available at: <https://www.guru99.com/database-normalization.html>

[Accessed 2020].

Flutter Community, 2019. *Flutter Documentation*. [Online]

Available at: <https://flutter.dev/docs>

[Accessed 12 October 2019].

Google, 2019. *Firebase Documemntation*. [Online]

Available at: <https://firebase.google.com/docs>

[Accessed 12 October 2019].

Gore, A., 2018. *Vue.js Developers - The Ultimate Vue.js & Laravel CRUD Tutorial*.

[Online]

Available at: <https://vuejsdevelopers.com/2018/02/05/vue-laravel-crud/>

[Accessed 28 December 2019].

Kent, W., 1983. *A Simple Guide to Five Normal Forms in Relational Database Theory*. s.l.:s.n.

Laravel LLC, 2019. *Laravel Documentation*. [Online]

Available at: <https://laravel.com/docs/6.x>

[Accessed 12 October 2019].

Press, C. U., 2009. *Evaluation in information | Stanford Edu*. [Online]

Available at: <https://nlp.stanford.edu/IR-book/pdf/08eval.pdf>

[Accessed 2019].

React Chart, 2019. *git hub*. [Online]

Available at: <https://github.com/jerairrest/react-charts-2/blob/master/example/src/components/line.js>

[Accessed 2019].

redux.js.org., 2019. *Motivation · Redux*. [Online]

Available at: <https://redux.js.org/>

[Accessed september 2019].

w3school, 2019. *AJAX Introduction | w3school*. [Online]

Available at: [https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp)

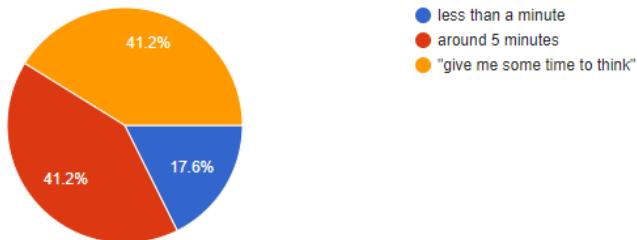
[Accessed 27 12 2019].

## CHAPTER 8: APPENDIX

### 8.1 APPENDIX A: SURVEY

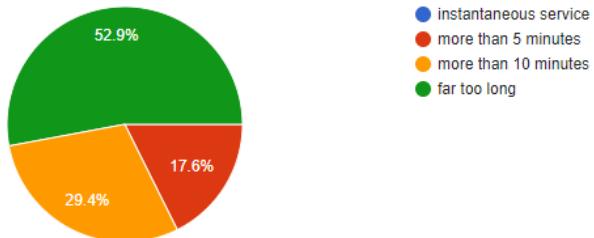
How long do you usually spend choosing meals in-front of the counter?

17 responses



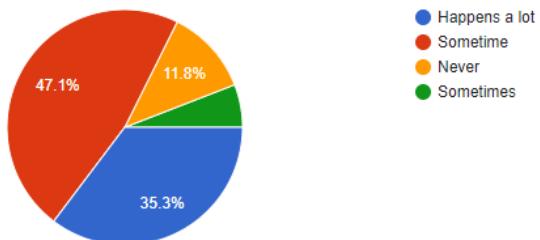
How long do you usually wait for meals once ordered in the college cafeteria?

17 responses



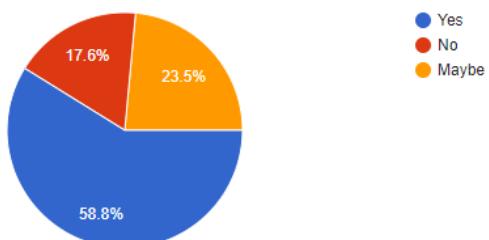
Have you had to move elsewhere because the cafeteria was too crowded?

17 responses



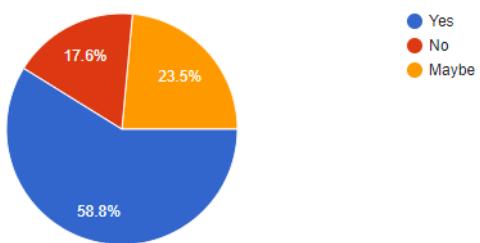
Do you have trouble deciding what to order often?

17 responses



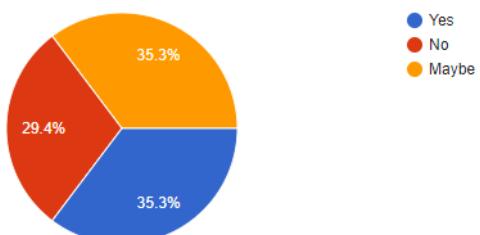
Does what other people order change your mind?

17 responses



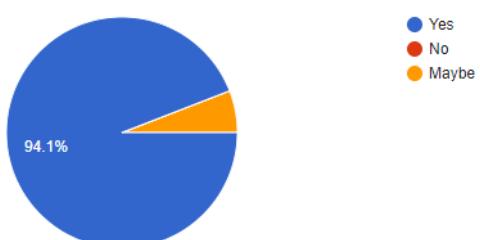
do you tend to avoid interaction with strangers?

17 responses



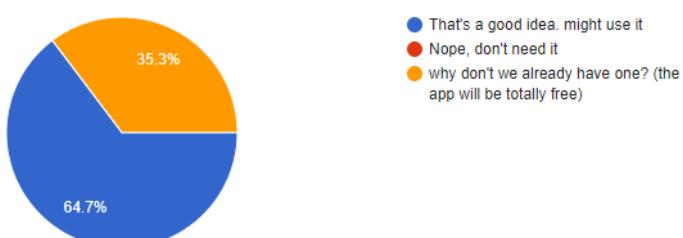
Would you want someone to place your orders and notify you when its ready?

17 responses



would it help if the college cafeteria had a mobile app?

17 responses



## 8.2 APPENDIX C: SAMPLE CODES

### 8.2.1 SAMPLE CODE OF THE WEB APP

The following is a section of action.js file used for redux state management method

```
export function login(credentials) {
    return (dispatch) => {
        dispatch({
            type: ActionTypes.AUTH_LOADING,
            payload: true
        });
        authService.login(credentials)
            .then((res) => {
                success(res.data.message);
                dispatch(setAuthUser(res.data));
                history.push('/');
            })
            .catch((err) => {
                warn("Invalid Credentials");
                dispatch({
                    type: ActionTypes.AUTH_INVALID,
                    payload: true
                });
            });
    };
}

// Send mail to the email to reset password
export function resetPassword(credentials) {
    return (dispatch) => {
        dispatch({
            type: ActionTypes.RESET_PASSWORD,
            payload: true
        });
        authService.resetPassword(credentials).then((res) => {
            success("Check Your Email");
            dispatch({
                type: ActionTypes.RESET_PASSWORD,
                payload: false
            });
        }).catch((err) => {
            warn("Try Again");
            dispatch({
                type: ActionTypes.RESET_PASSWORD,
                payload: false
            });
        });
    };
}
```

```

// update password
export function updatePassword(credentials) {
    return (dispatch) => {
        authService.updatePassword(credentials).then(response => {
            success(response.data.message);
            history.push("/login");
        }).catch(({response}) => {
            dispatch({
                type: ActionTypes.UPDATE_PASSWORD,
                payload: response.data.errors
            });
            warn("Invalid Credentials");
        });
    };
}

export function logout() {
    return {
        type: ActionTypes.AUTH_LOGOUT,
    };
}

export function authCheck() {
    try {
        let access_token = localStorage.getItem('access_token');
        if (access_token) {
            return auth GetUser(access_token);
        }
    } catch (err) {
    }
    return logout();
}

export function auth GetUser(token) {
    return (dispatch) => {
        Http.defaults.headers.common.Authorization = `Bearer ${token}`;
        authService.getUser().then(response => {
            dispatch(setAuthUser({
                access_token: token,
                user: response.data
            }));
            dispatch(setSetting({
                ...response.data.settings
            }));
        }).catch(e => authLogout());
    };
}

```

```

export function resetUserForm() {
    return {
        type: ActionTypes.RESET_USER
    };
}

export function storeUser(payload) {
    return (dispatch) => {
        dispatch(setUserLoading(true));
        userService.store(payload).then(response => {
            success(response.data.message);
            history.push('/user');
        }).catch(({response}) => {
            warn(response.data.message);
            dispatch({
                type: ActionTypes.USER_ERROR,
                payload: response.data.errors
            });
        });
    };
}

export function setUser(user) {
    return {
        type: ActionTypes.SET_USER,
        payload: user
    };
}

export function getUser(slug) {
    return (dispatch) => {
        dispatch(setUserLoading(true));
        userService.getUser(slug).then(response => {
            dispatch(setUser(response.data.data));
            dispatch(setUserLoading(false));
        }).catch(({response}) => {
            warn(response.data.message);
        });
    };
}

export function updateUser(slug, data) {
    return (dispatch) => {
        dispatch(setUserLoading(true));
        userService.update(slug, data).then(response => {
            dispatch(setUser(response.data.data));
            success(response.data.message);
            history.push('/user');
        });
    };
}

```

```

        }).catch(({response}) => {
            warn(response.data.message);
            dispatch({
                type: ActionTypes.USER_ERROR,
                payload: response.data.errors
            });
        });
    );
}
}

export function deleteUser(slug) {
    return (dispatch) => {
        dispatch(setUserLoading(true));
        userService.remove(slug).then(response => {
            success(response.data.message);
            history.push('/user');
        }).catch(({response}) => {
            warn(response.data.message);
            dispatch(setUserLoading(false));
        });
    };
}

export function setUserLoading/loading) {
    return {
        type: ActionTypes.USER_LOADING,
        payload: loading
    };
}

//Orders
export function setOrders(orders){
    return {
        type: ActionTypes.HOME_ORDERS,
        payload: orders
    }
}

export function storeOrders(orders){
    return dispatch=>{
        foodServices.storeOrder(orders).then((response)=>{
            if(response.status==201){
                success("Orders added successfully");
                dispatch(setSelected([]));
            }else{
                warn("Trouble adding Orders. check orders and add again");
            }
            dispatch(getOrders());
        })
    }
}

```

```

        });
    }
}

export function updateOrders(id,status){
    return dispatch=>{
        foodServices.updateOrder(id,status).then((response)=>{
            if(response.status==200){
                success(response.data.message);
            }else{
                warn(response.data.message);
            }
            dispatch(getOrders());
        }).catch(e=>{
            console.log("error", e)
        });
    }
}

export function showMenu(data){
    return {
        type: ActionTypes.HOME_SHOW_MENU,
        payload: data
    }
}

export function setHomeSearch(text){
    return {
        type: ActionTypes.HOME_SEARCH,
        payload: text
    }
}

export function setSelected(data){
    return {
        type: ActionTypes.HOME_SELECTED,
        payload: data
    }
}

export function setHomeLoading/loading){
    return {
        type: ActionTypes.HOME_LOADING,
        payload: loading
    }
}

```

```

export function getOrders(){
    return dispatch => {
        dispatch(setHomeLoading(true));
        foodServices.getOrders().then(
            response=>{
                dispatch(setOrders(response.data))
            }
        );
        dispatch(setHomeLoading(false));
    }
}

//HOME
export function setStatsLoading(payload){
    return {
        type: ActionTypes.STATS_LOADING,
        payload: payload
    }
}

export function setTopStats(payload){
    return {
        type: ActionTypes.STATS_TOP,
        payload: payload
    }
}

export function setWeeklyStats(payload){
    return {
        type: ActionTypes.STATS_WEEKLY,
        payload: payload
    }
}

export function setFood(payload){
    return {
        type: ActionTypes.FOOD_FOOD,
        payload: payload
    }
}

export function setFoodSelected(payload){
    return {
        type: ActionTypes.FOOD_SELECTED,
        payload: payload
    }
}

```

```

export function getStats(){
    return dispatch=>{
        dispatch(setStatsLoading(true));
        foodServices.getStats().then(
            response=>{
                dispatch(setTopStats(response.data.top));
                dispatch(setWeeklyStats(response.data.weekly));
                dispatch(setStatsLoading(false));
            }
        )
    }
}

export function getCategories(){
    return dispatch=>{
        dispatch(setFoodLoading(true));
        var categories = [];
        foodServices.getCategories().then(response => {
            categories=
                response.data.map(c => {
                    return {value: c.id, label: c.name};
                })
            dispatch(setCategories(categories));
        });
        dispatch(setFoodLoading(false));
    }
}

export function insertCategory(data){
    return dispatch=>{
        foodServices.insertCategories(data).then(response=>{
            if(response.status==200){
                dispatch(getCategories());
                success("new category "+data.name+' created');
            }else{
                }
        });
    }
}

```

## 8.2.2 SAMPLE CODE OF THE MOBILE APP

Following is the flutter code form Home.dart file of the mobile app

```
class Home extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    Connector().checkConnection();

    return ScopedModelDescendant<MainModel>(
      builder: (_, child, ModelHome model){
        if(!model.homeLoaded()){
          model.updateData();
        }
        List<Widget> menutabheads = [];
        for (MenuCategory mc in model.menu) {
          menutabheads.add(Text(mc.name));
        }
        return Scaffold(
          backgroundColor: AppColors.primaryBackground,
          body: Container(
            padding: EdgeInsets.only(left: 20),
            child: Column(
              mainAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Container(
                  child: FlatButton(
                    splashColor: Colors.transparent,
                    highlightColor: Colors.transparent,
                    onPressed: (){
                      model.toggleRecomHeight();
                    },
                  ),
                  child: Row(
                    children: <Widget>[
                      Txt( 'Try our Recommendations', type: 'thin-
header',),
                      Spacer(),
                      Icon(
                        model.recomHeight==200?
                        Icons.expand_less:
                        Icons.expand_more,
                        color: Colors.grey[400],
                      ),
                      SizedBox(width:20),
                    ],
                  ),
                ),
              ],
            ),
          ),
        );
      }
    );
  }
}
```

```

        Recommends(model),
        LinkedText(
            text: 'Menu',
            left: 20,
            top: 10,
            width: double.infinity,
            type: 'thin-header', onTap: (){
                model.setRecomHeight(100);
            }, ),
        Menu(model),
    ],
),
);
);
);
}
}
}

```

Following is the business logic for home page separated with Scoped Model State Management method.

```

mixin ModelHome on ModelConnector{
    bool _homeLoading =false;
    bool _loaded = false;
    List<MenuItem> _recommended = [];
    double recomHeight = 200;
    double maxRecomHeight = 200;
    double minRecomHeight = 200;

    recommended()=>_recommended;
    setRecommended(List<MenuItem> r){
        _recommended = r;
    }

    // recomHeight()=>_recomHeight;
    setRecomHeight(double h){
        recomHeight=h;
        notifyListeners();
    }
    toogleRecomHeight(){
        recomHeight=recomHeight==200?100:200;
        notifyListeners();
    }
}

```

```

homeLoaded()=>_loaded;
setHomeLoaded(bool l){
    _loaded = l;
    notifyListeners();
}

homeloading()=>_homeLoading;
 setLoading(bool l){
    _homeLoading=l;
    notifyListeners();
}

updateData(){
    setMenu();
    setRecommend();
}

dragged(double moved){
    if(recomHeight>minRecomHeight && moved>0 || recomHeight<maxRecomHeight &
& moved<0){
        recomHeight = recomHeight - moved;
    }
    notifyListeners();
}

setMenu()async{
    setHomeLoaded(true);
    Connector().getMenu().then((res){
        if(res.statusCode==200){
            var data = json.decode(res.body);
            if(data.length!=0){
                for (var item in data) {
                    List<MenuItem> foods =[];
                    for (var i in item['foods']) {
                        MenuItem mi = MenuItem(
                            id: i['id'],
                            name: i['name'],
                            price: i['price']/1,
                            served: i['served_today'],
                            details: i['details'],
                        );
                        foods.add(mi);
                    }
                    MenuCategory mc = MenuCategory(
                        id: item['id'],
                        name: item['name'],

```

```

        foods: foods,
    );
    menu.add(mc);
}
}
notifyListeners();
});
});
}

setRecommend()async{
print('here');
Connector().getRecommend().then((res){
if(res.statusCode==200){
var data = json.decode(res.body);
if(data[ 'data']!=null){
for (var item in data[ 'data'] ) {
MenuItem m = MenuItem(
id: item[ 'id'],
name: item[ 'name'],
details: item[ 'details'],
price: item[ 'price']/1,
image: item[ 'image'],
);
_recommended.add(m);
}
notifyListeners();
}
}
})
);
}

appendMenu(MenuCategory mc){
menu.add(mc);
notifyListeners();
}
}

```

## Orders.dart

```
class Orders extends StatelessWidget {

  @override

  Widget build(BuildContext context) {
    bool _updated = false;

    return Container(
      color: Colors.white,
      child: ScopedModelDescendant<MainModel>(
        builder: (context, child, ModelOrders model) {
          if (!_updated) {
            model.updateOrders();
            _updated = true;
          }
          return Container(
            color: Colors.white,
            child: Column(
              children: <Widget>[
                Container(
                  height: MediaQuery.of(context).size.height - 100,
                  child: model.orderTimeLineModel().length > 0
                    ? Timeline(
                      position: TimelinePosition.Left,
                      children: model.orderTimeLineModel(),)
                    : model.orderLoading()
                    ? Center(
                      child: LoadingIndicator(
                        size: 30,
                        width: 2,))

                    : Center(
                      child: Txt(
                        'No Orders Yet',))),
                Spacer(),
              ],
            );
          }
        }));
  }
}
```

## modelOrders.dart

```
mixin ModelOrders on ModelConnector{
  bool _orderLoading = false;
  List<Order> _ordereditems = [];
  List<TimelineModel> _ordertimeline = [];

  orderTimeLineModel()=>_ordertimeline;

  orderLoading()=>_orderLoading;
  setOrderLoading(bool l){
    _orderLoading = l;
    notifyListeners();}

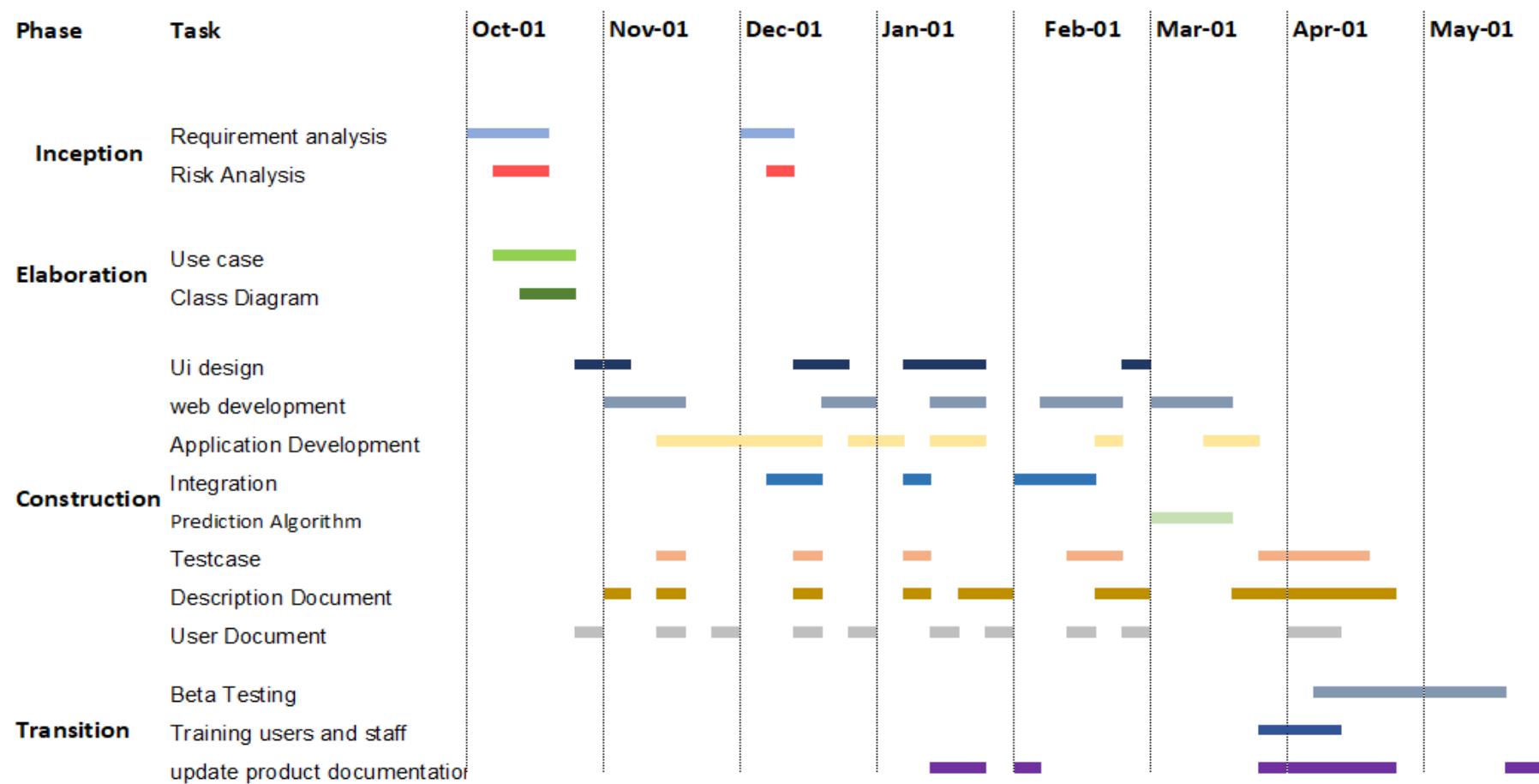
  ordereditems()=>_ordereditems;
  setOrdereditems(List<Order> o){
    _ordereditems = o;
    notifyListeners();}

  updateOrders()async{
    setOrderLoading(true);
    Connector().getOrderHistory().then((data){
      print(data);
      List<Order> retrieved = data.map<Order>((d){
        return Order(
          id: d['id'],
          fo: d['for'],
          ordered: d['ordered'].map<OrderItem>((oi){
            return OrderItem(
              id: oi['id'],
              name: oi['food']['name'],
              price: oi['price']/1,
              quantity: oi['quantity'],
              status: oi['status'],
              image: oi['image']); }).toList(),
        );});).toList();
      setOrdereditems(retrieved);
      updateOrderTimeLine();
    });
  }

  updateOrderTimeLine(){
    _ordertimeline = ordereditems().map<TimeLineOrder>((Order o){
      return TimeLineOrder(o);
    }).toList();
    setOrderLoading(false);
    notifyListeners();
  }
}
```

## 8.3 APPENDIX D: DESIGNS

### 8.3.1 GANTT CHART



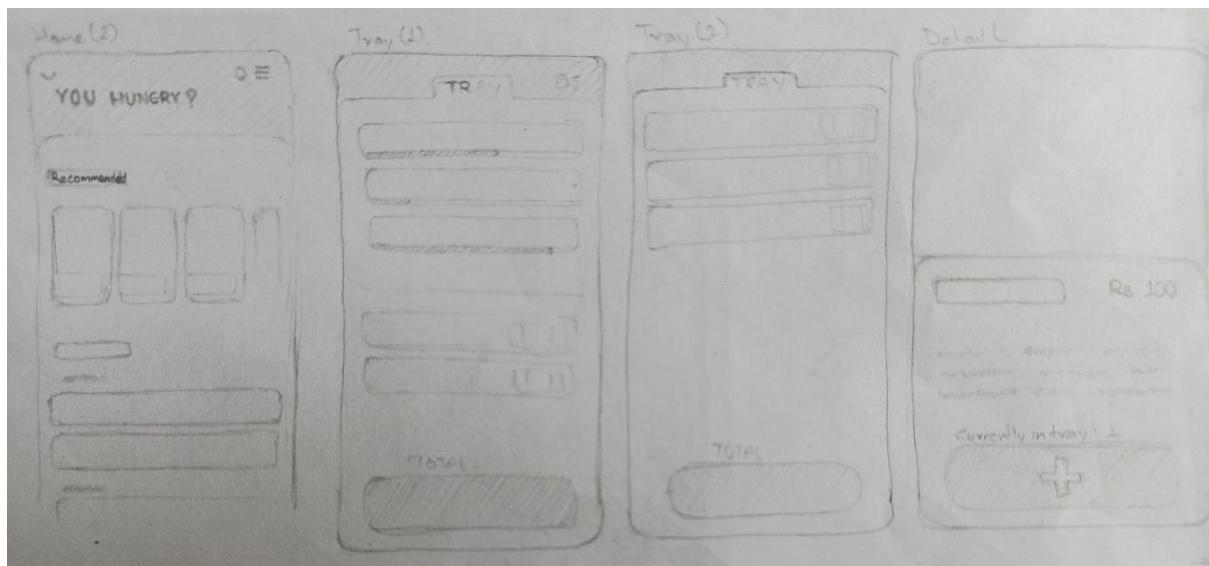
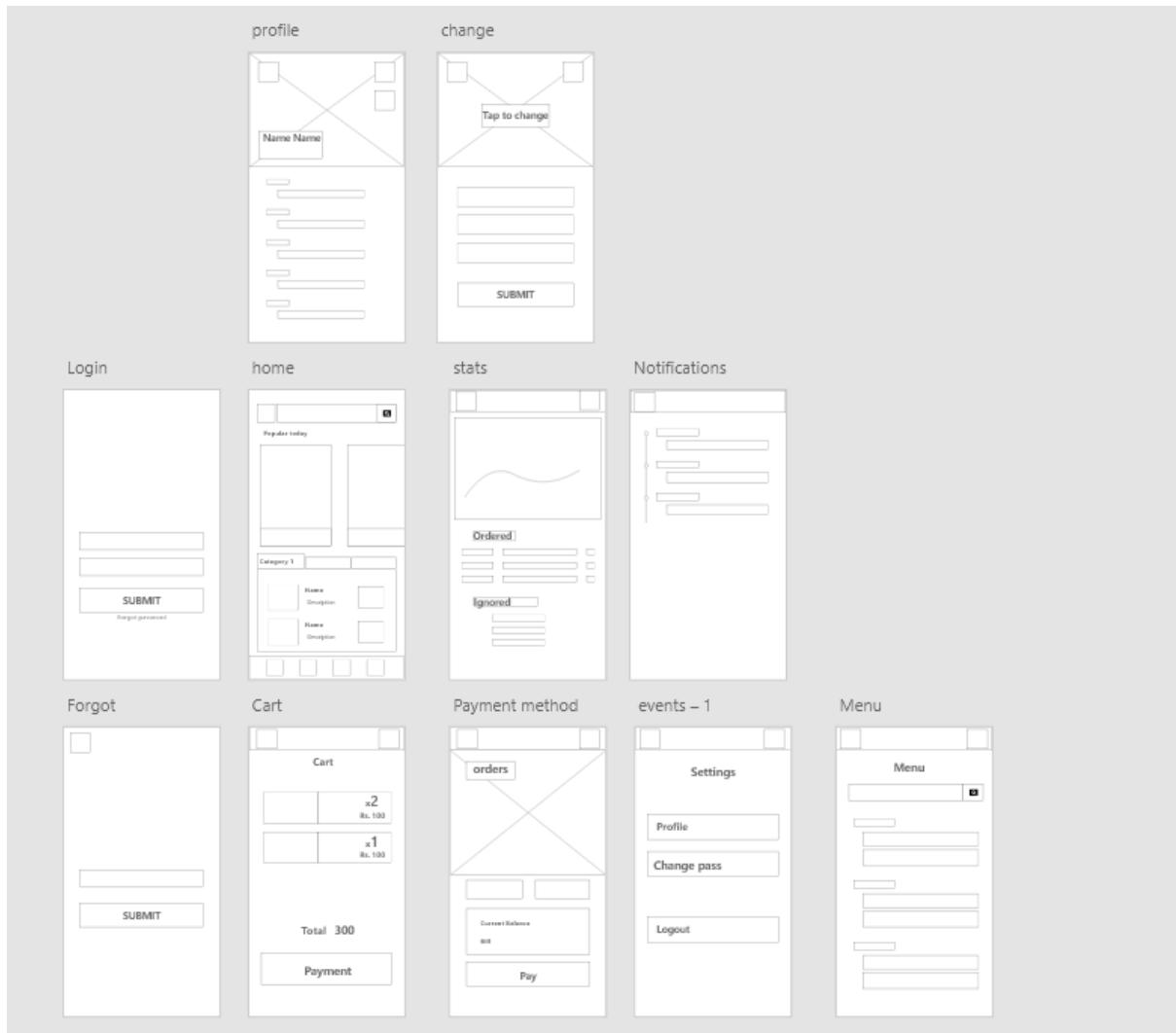
### 8.3.2 WORK LOG

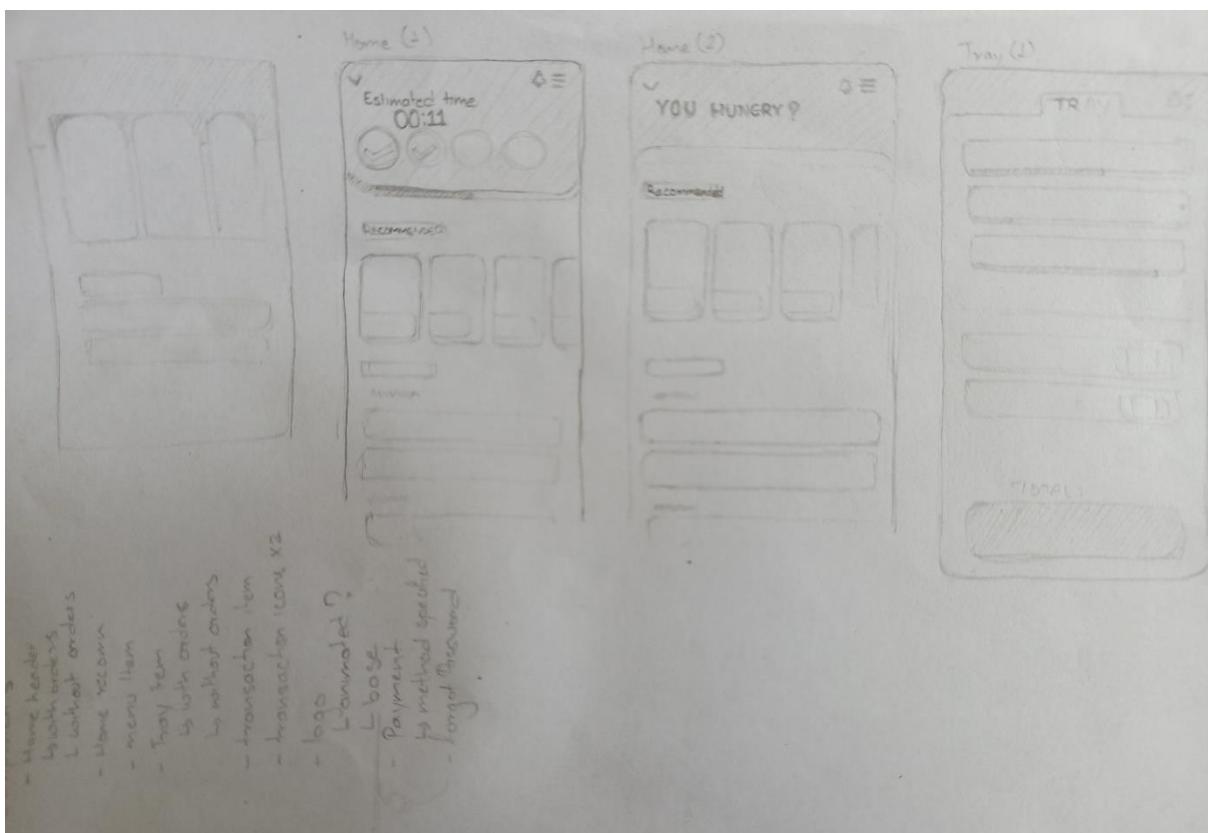
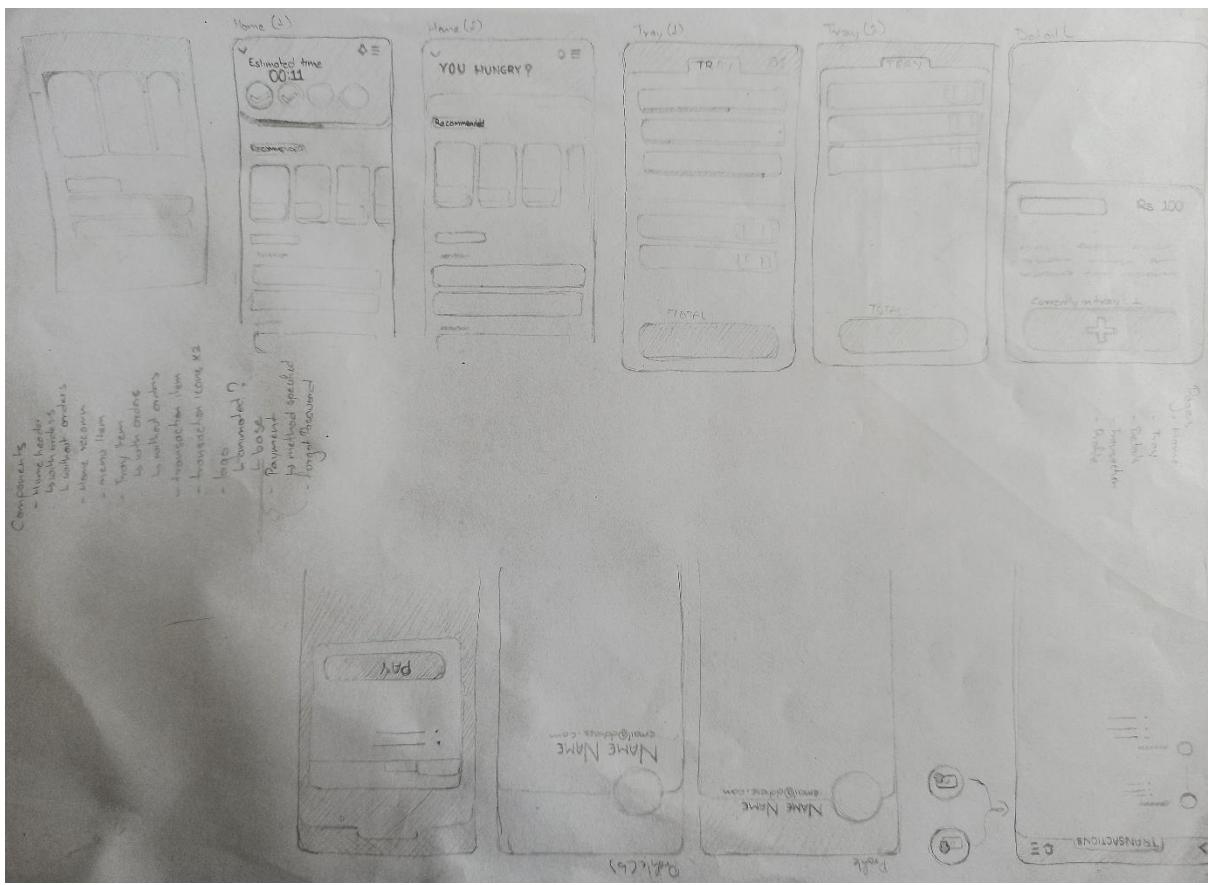
	A	B	C	D	E	F	G
1	<b>Name</b>	Deepan Singh					
2	<b>Project Name</b>	Canteen app					
3	<b>Student ID</b>	id - 17031147					
4	<b>Internal Supervisor</b>	Subeksha Shrestha					
5	<b>External Supervisor</b>	Ishwor Shrestha					
6							
7							
8	Section	Feature	Task	Completed	Completed On		
9					first iteration	second Iteration	Third Iteration
10	Web	Login	Create login api	<input checked="" type="checkbox"/>	24/11/2019		5/3/2020
11			Add form validation	<input checked="" type="checkbox"/>	24/11/2019		5/3/2020
12			implement jwt auth for future api	<input checked="" type="checkbox"/>			5/3/2020
13			Implement in front end	<input checked="" type="checkbox"/>	24/11/2019		5/3/2020
14							
15		Navigation	create page routes	<input checked="" type="checkbox"/>			5/3/2020
16			implement in front end	<input checked="" type="checkbox"/>	9/12/2019		5/3/2020
17							
18		Statistics component	create api	<input checked="" type="checkbox"/>			14/4/2020
19			create algorithm	<input checked="" type="checkbox"/>			14/4/2020
20			implement in front end	<input checked="" type="checkbox"/>			14/4/2020
21							
22		Show Menu component	create menu api	<input checked="" type="checkbox"/>	10/12/2019		6/3/2020
23			Implement in front end	<input checked="" type="checkbox"/>	10/12/2019		6/3/2020
24							
25		Reccomendation system	create recommendation api	<input checked="" type="checkbox"/>	11/12/2019		6/3/2020
26			create recommendation algorithm	<input checked="" type="checkbox"/>			14/4/2020
27			implement in front end	<input checked="" type="checkbox"/>			8/3/2020
28							
29		tray component	create backend function	<input checked="" type="checkbox"/>			16/3/2020
30			implement in front end	<input checked="" type="checkbox"/>			16/3/2020
31							
32		Order function	create order api	<input checked="" type="checkbox"/>	11/12/2019		6/3/2020
33			implement in front end	<input checked="" type="checkbox"/>	11/12/2019		7/3/2020
34							
35		User Orders List	create pending order api	<input checked="" type="checkbox"/>	8/1/2020	22/1/2020	6/3/2020
36			implement in front end	<input checked="" type="checkbox"/>	8/1/2020	22/1/2020	9/3/2020
37							
38		Order Response funciton	create respond to order api	<input checked="" type="checkbox"/>			6/3/2020
39			add notification feature	<input checked="" type="checkbox"/>			12/4/2020
40			implement on front end	<input checked="" type="checkbox"/>			7/3/2020
41							
42		Food and menu CMS	create api	<input checked="" type="checkbox"/>	14/1/2020		6/3/2020
43			add validation	<input checked="" type="checkbox"/>			17/4/2020
44			create backend function	<input checked="" type="checkbox"/>			17/3/2020
45			implement in front end	<input checked="" type="checkbox"/>	14/1/2020		17/3/2020
46							
47		Order History	create orders retrieval api	<input checked="" type="checkbox"/>	16/1/2020		6/3/2020
48			implement in front end	<input checked="" type="checkbox"/>	16/1/2020		16/3/2020
49							
50		Transactions	create transaction retrieval api	<input checked="" type="checkbox"/>			1/4/2020
51			implement in front end	<input checked="" type="checkbox"/>			1/4/2020
52							
53		Users details page	create user list api	<input checked="" type="checkbox"/>			1/4/2020
54			implement in front end	<input checked="" type="checkbox"/>			15/4/2020
55							
56		Account Balance update	create api to update balance	<input checked="" type="checkbox"/>			17/4/2020
57			implement in front end	<input type="checkbox"/>			
58							
59		Testing	create test cases	<input checked="" type="checkbox"/>			17/4/2020
60			test application	<input checked="" type="checkbox"/>			17/4/2020
61							

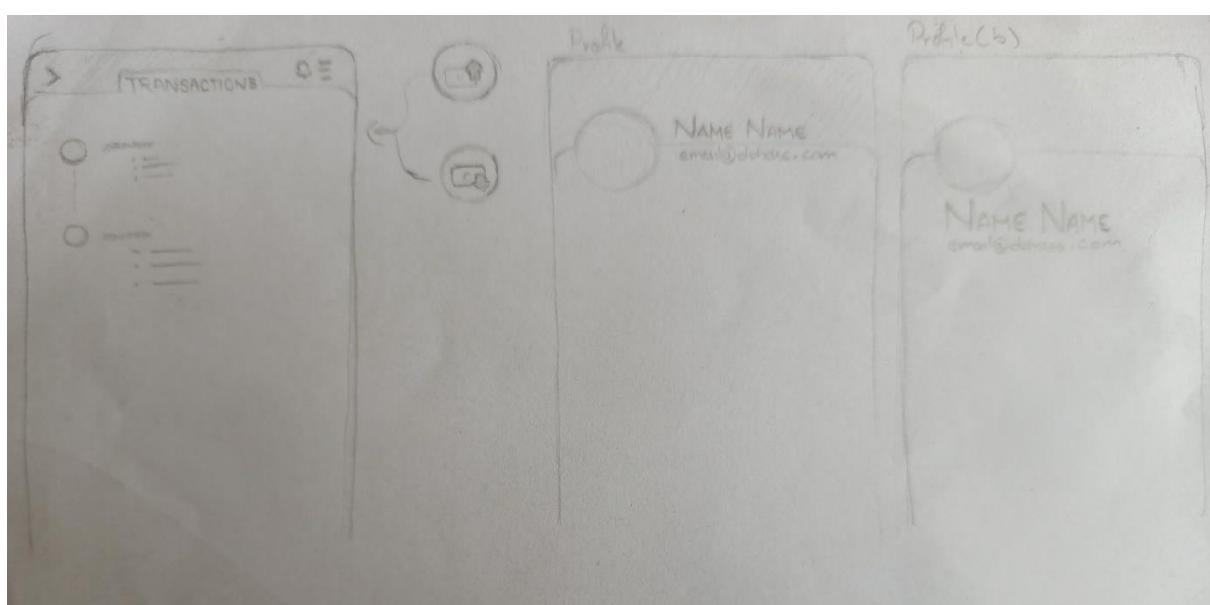
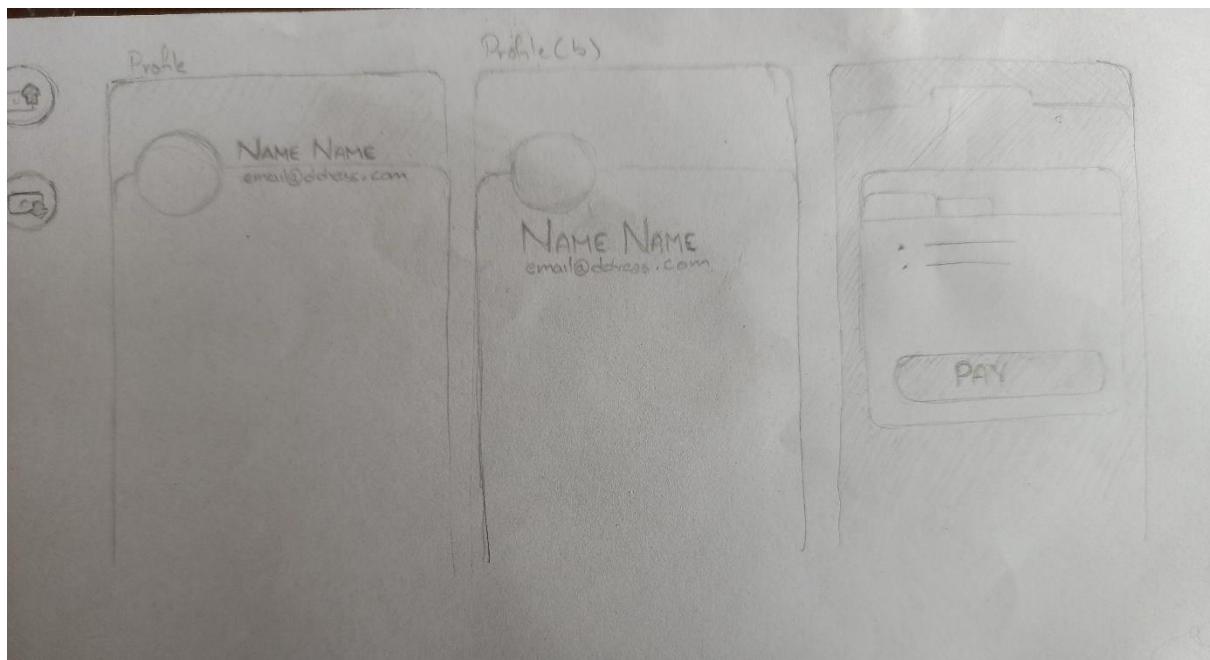
A	B	C	D	E	F	G
55	Name	Deepan Singh				
56	Project Name	Canteen app				
57	Student ID	id - 17031147				
58	Internal Supervisor	Subeksha Shrestha				
59	External Supervisor	Ishwor Shrestha				
60						
61						
62	Mobile App					
63	Login	Create login api	✓	11/12/2019	11/12/2019	6/3/2020
64		Add form validation	✓	29/12/2019	12/1/2020	30/3/2020
65		Implement in front end	✓	29/12/2019	12/1/2020	30/3/2020
66						
67	Navigation	create page routes	✓	29/12/2019	12/1/2020	30/3/2020
68		implement in front end	✓	29/12/2019	12/1/2020	30/3/2020
69						
70	Show Menu component	create menu api	✓	11/12/2019	12/1/2020	6/3/2020
71		Implement in front end	✓	3/1/2020	12/1/2020	30/3/2020
72						
73	tray component	create backend function	✓	3/1/2020	12/1/2020	6/3/2020
74		implement in front end	✓	3/1/2020	12/1/2020	1/4/2020
75						
76	Order function	create order api	✓	11/12/2019	13/1/2020	6/3/2020
77		implement in front end	✓	4/1/2020	13/1/2020	1/4/2020
78						
79	Notification	implement firebase	✓		21/1/2020	13/4/2020
80		integrate with one signal	✓		21/1/2020	13/4/2020
81		on notification received function	✓			13/4/2020
82						
83	Check food progress	create check status api	✓			3/4/2020
84		implement in front end	✓			17/4/2020
85						
86	Tray Page	backend processing function	✓	4/1/2020		6/4/2020
87		implement in front end	✓	4/1/2020		6/4/2020
88						
89	Order History	create orders history api of given user	✓	5/1/2020	13/1/2020	4/4/2020
90		implement in front end	✓	5/1/2020	13/1/2020	4/4/2020
91						
92	Payment	create api	✓			13/4/2020
93		validation	□			
94		implement in UI	□			
95						
96	Transactions	create transaction retrieval api of given user	✓	5/1/2020	13/1/2020	4/4/2020
97		implement in front end	✓	6/1/2020	13/1/2020	4/4/2020
98						
99	Settings page	create each user data api	✓	9/1/2020		6/3/2020
100		implement in front end	✓	9/1/2020		12/4/2020
101						
102	User Instructions	create on app instructions	✓			12/4/2020
103		create user manual	✓			16/4/2020
104						
105	Testing	create test cases	✓			17/4/2020
106		test application	✓			17/4/2020
	Add	1000	more rows at bottom.			

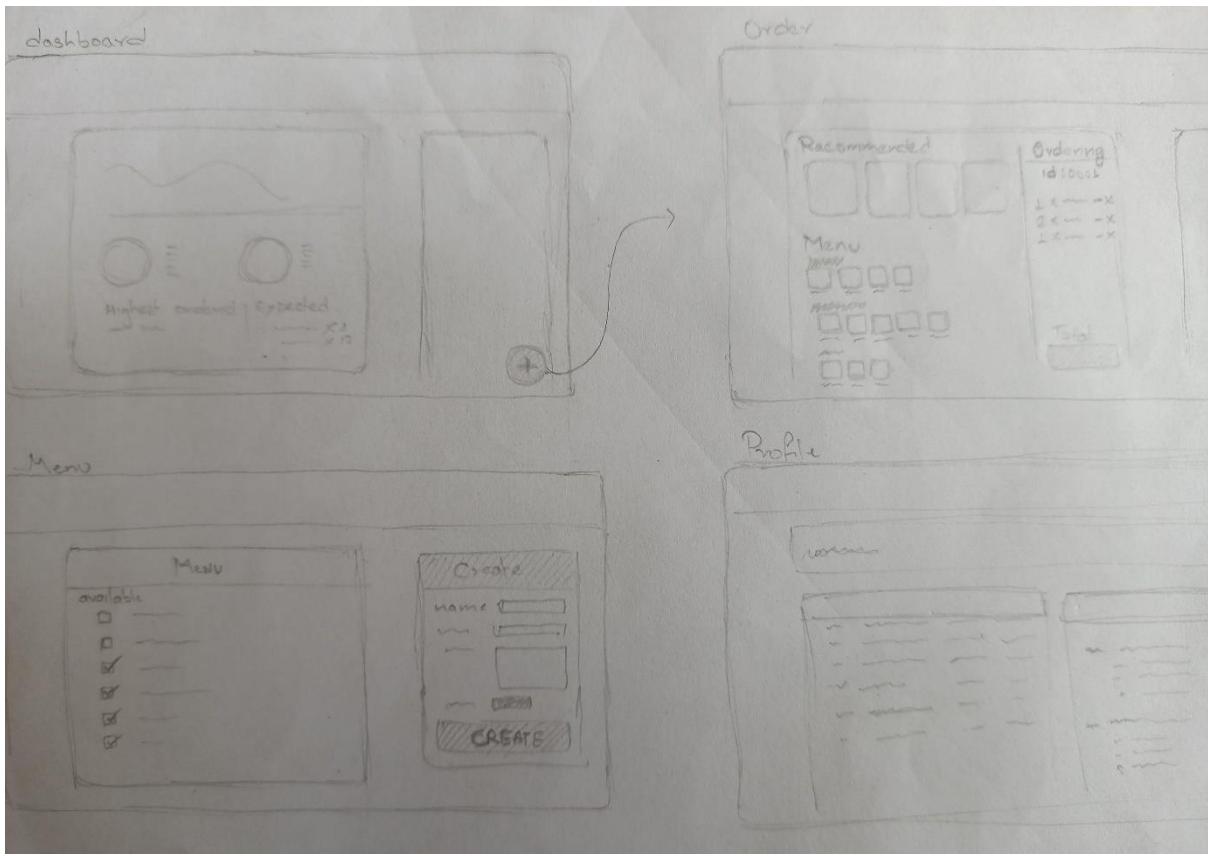
### 8.3.3 WIREFRAME

The following wireframe was designed in Adobe XD software for windows



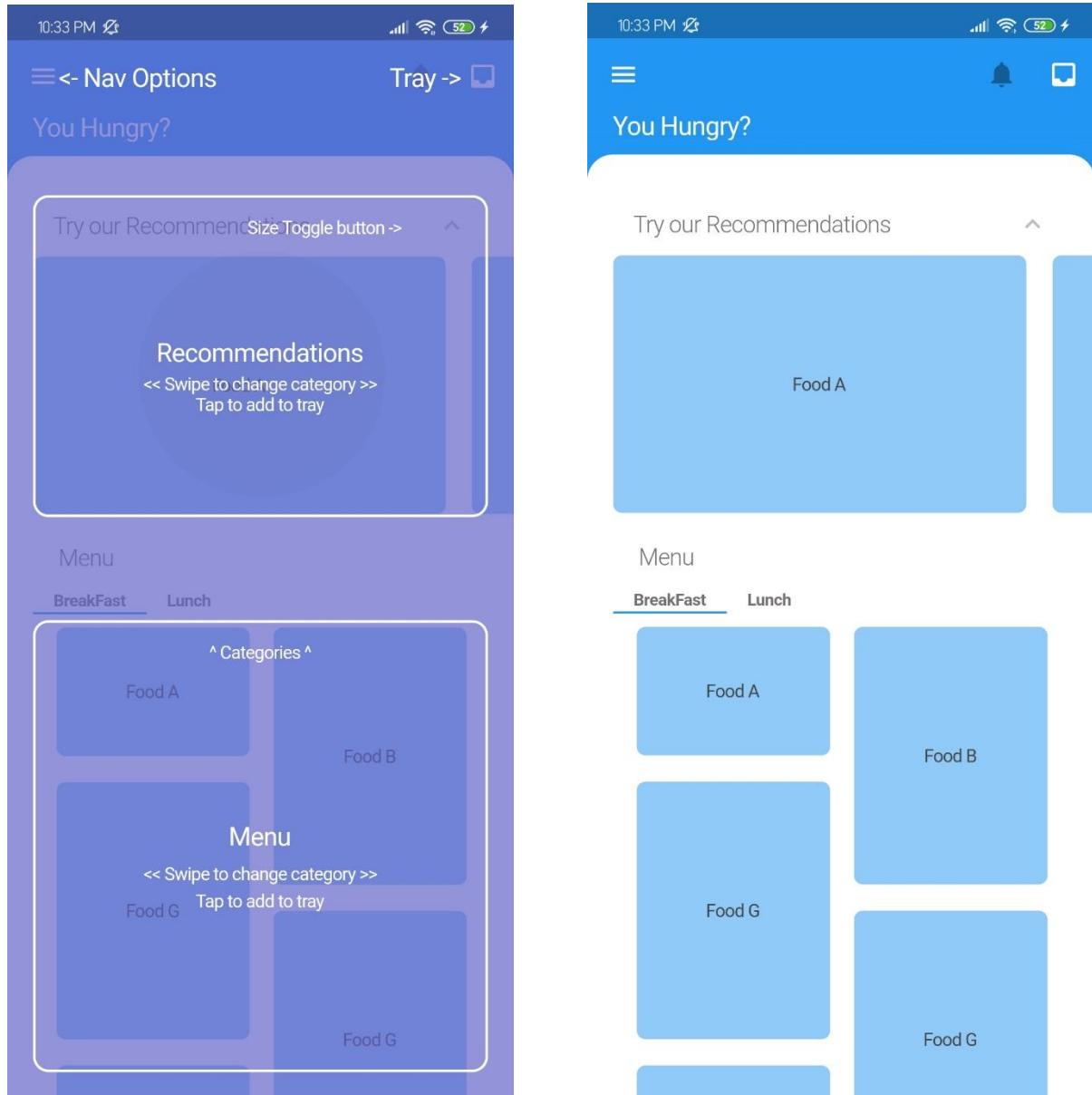


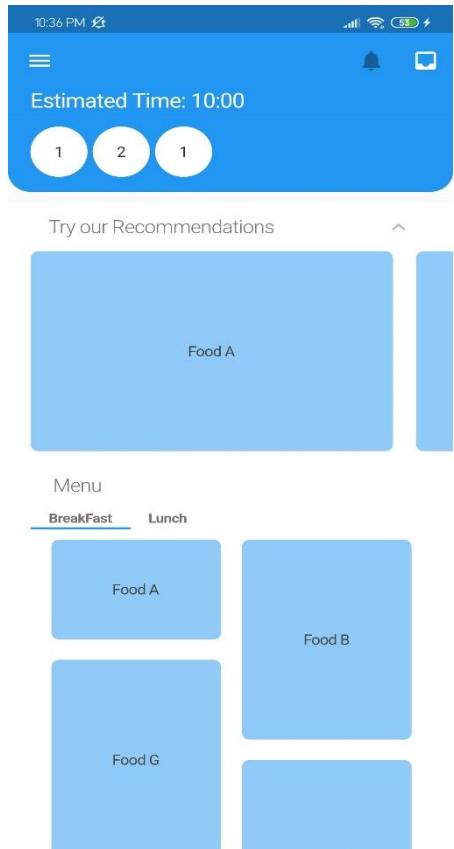
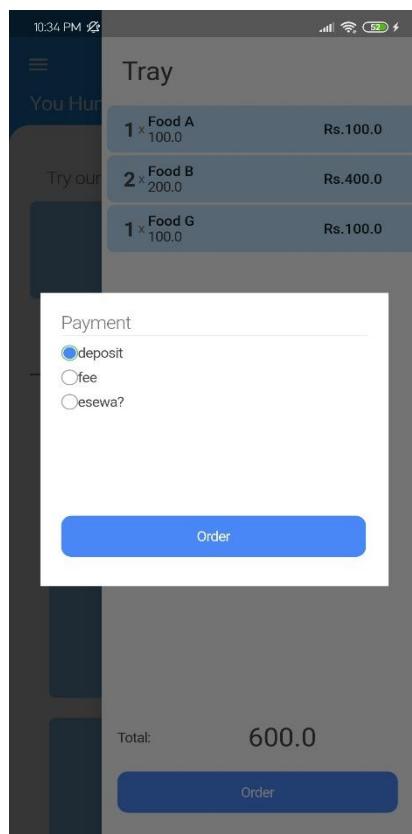
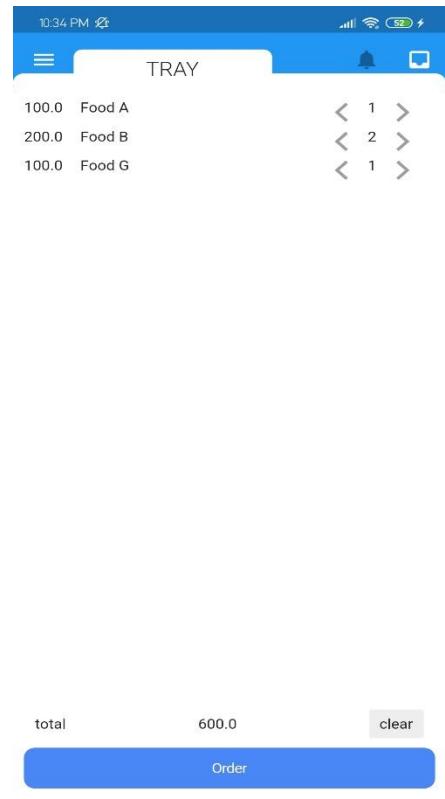
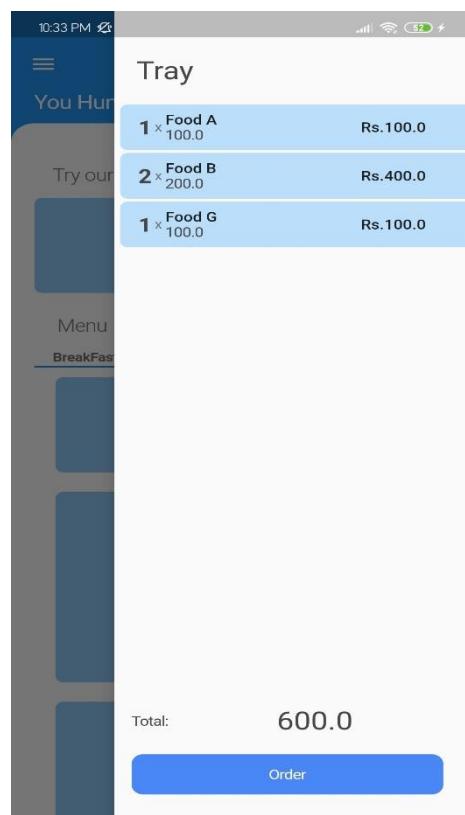
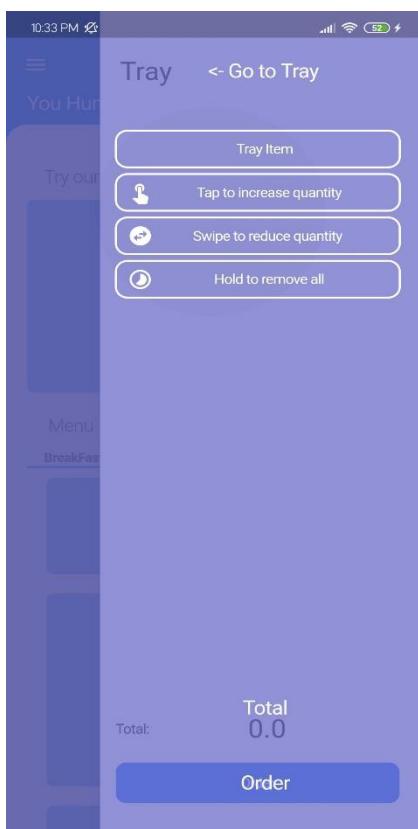




## 8.4 APPENDIX E: SCREENSHOTS OF THE SYSTEM

### 8.4.1 MOBILE APP





The image displays three screenshots of a mobile application interface, likely for a food delivery or ordering service.

**Home Screen:**

- Top bar: 10:34 PM, signal strength, battery level.
- User info: nameemail
- Balances: Rs. 500
- Navigation menu:
  - Home
  - Tray
  - Orders
  - Payments

**Payments Screen:**

Shows a list of payment transactions:

- 2020-04-11 14:32:36: Rs. 100.0 paid from deposit account food ordered via app; order id 54
- 2020-04-11 14:41:38: Rs. 100.0 paid from deposit account food ordered via app; order id 55
- 2020-04-11 17:12:08: Rs. 200.0 paid from deposit account food ordered via app; order id 57
- 2020-04-11 17:15:02: Rs. 200.0 paid from deposit account food ordered via app; order id 58
- 2020-04-12 07:36:25: Rs. 200.0 paid from deposit account food ordered via app; order id 59
- 2020-04-12 07:42:07: Rs. 200.0 paid from deposit account food ordered via app; order id 60

**Orders Screen:**

Shows a list of orders:

- Order id - 89: Total - Rs. 120.0 2020-04-19 00:00:00 1 \* naya @ 120.0
- Order id - 88: Total - Rs. 200.0 2020-04-19 00:00:00 2 \* Food F @ 100.0
- Order id - 87: Total - Rs. 500.0 2020-04-19 00:00:00 2 \* Food E @ 100.0 1 \* Food B @ 200.0 1 \* Food D @ 100.0
- Order id - 86: Total - Rs. 440.0 2020-04-19 00:00:00 2 \* naya 2 \* Food F @ 120.0 @ 100.0
- Order id - 85: Total - Rs. 200.0 2020-04-18 00:00:00 1 \* Food A @ 100.0 1 \* Food H @ 100.0
- Order id - 84: Total - Rs. 240.0 2020-04-18 00:00:00 2 \* naya @ 120.0
- Order id - 83: Total - Rs. 200.0 2020-04-18 00:00:00

## 8.4.2 WEB APPLICATION

Screenshot of the Canteen Counter web application login page:

**Canteen Counter**  
Ready to start the business?  
Enter email  
Password  
 Remember me  
Don't Remember your Password?  
**Sign in**

---

Screenshot of the Canteen Counter web application dashboard:

**Todays Top Sellers**  
Sales today: Rs. 1260

Food Item	Count
Food F	4
naya	3
Food E	2
Food B	1
Food D	1

**Weekly Sales Report**

2020-04-18 2020-04-17 2020-04-16 2020-04-15 2020-04-14  
2020-04-13 2020-04-12

Date	Time	Sales
2020-04-18	8:00	400
2020-04-18	9:00	700
2020-04-18	10:00	980
2020-04-18	11:00	100
2020-04-18	12:00	440
2020-04-18	13:00	240
2020-04-18	14:00	0
2020-04-18	15:00	0
2020-04-18	16:00	0
2020-04-18	17:00	0
2020-04-17	8:00	0
2020-04-17	9:00	0
2020-04-17	10:00	540
2020-04-17	11:00	0
2020-04-17	12:00	0
2020-04-17	13:00	0
2020-04-17	14:00	0
2020-04-17	15:00	0
2020-04-17	16:00	0
2020-04-17	17:00	0
2020-04-16	8:00	0
2020-04-16	9:00	0
2020-04-16	10:00	0
2020-04-16	11:00	0
2020-04-16	12:00	0
2020-04-16	13:00	0
2020-04-16	14:00	0
2020-04-16	15:00	0
2020-04-16	16:00	0
2020-04-16	17:00	0
2020-04-15	8:00	0
2020-04-15	9:00	0
2020-04-15	10:00	0
2020-04-15	11:00	0
2020-04-15	12:00	0
2020-04-15	13:00	0
2020-04-15	14:00	0
2020-04-15	15:00	0
2020-04-15	16:00	0
2020-04-15	17:00	0
2020-04-14	8:00	0
2020-04-14	9:00	0
2020-04-14	10:00	0
2020-04-14	11:00	0
2020-04-14	12:00	0
2020-04-14	13:00	0
2020-04-14	14:00	0
2020-04-14	15:00	0
2020-04-14	16:00	0
2020-04-14	17:00	0

**Orders** (group by food, time, user.)

User	Food	Count	Action
naya	x 1	4	X ✓
Food G	x 1	3	X ✓
Food B	x 1	2	X ✓
Food A	x 1	1	X ✓
Food B	x 1	1	X ✓
Food A	x 1	1	X ✓
Food B	x 1	1	X ✓

**admin**

Screenshot of the Cant\_App application interface showing the Home screen.

**Tray**  
Rs. 400

Food G 1 x 100 Rs. 100	Food F 2 x 100 Rs. 200	Food E 1 x 100 Rs. 100
- X	- X	- X

**Best Sellers**

Food F Rs.100	naya Rs.120	Food E Rs.100	Food B Rs.200	Food D Rs.100
------------------	----------------	------------------	------------------	------------------

**Menu**

**BreakFast**

Food A Rs.100	Food B Rs.200	Food G Rs.100	nn Rs.100	naya Rs.120
------------------	------------------	------------------	--------------	----------------

**Lunch**

Food C Rs.100	Food D Rs.100	Food E Rs.100	Food F Rs.100	Food H Rs.100
------------------	------------------	------------------	------------------	------------------

**Orders** (group by food, time, user)

naya x 1	X	V
Food G x 1	X	V
Food B x 1	X	V
Food A x 1	X	V
Food B x 1	X	V
Food A x 1	X	V
Food B x 1	X	V



Screenshot of the Cant\_App application interface showing the Menu screen.

**Menu**

**BreakFast**

- Food A 100
- Food B 200
- Food G 100
- Food G 100
- nn 100
- naya 120

**Lunch**

- Food C 100
- Food D 100
- Food E 100
- Food F 100
- Food H 100
- Food H 100

**Create Food**

Name:

Category:

Price:

Description:

**Create category**

Name:

**Create** button

Cant\_App

Not secure | cant.test/all-orders

Home Menu Orders Transactions Users admin

### All Orders

Order id.	Date	User
Order id. 89	2020-04-19 00:00:00	a@islingtoncollege.edu.np
1 * naya	@ Rs.120	
Order id. 88	2020-04-19 00:00:00	a@islingtoncollege.edu.np
2 * Food F	@ Rs.100	
Order id. 87	2020-04-19 00:00:00	a@islingtoncollege.edu.np
2 * Food E	@ Rs.100	
1 * Food B	@ Rs.200	
1 * Food D	@ Rs.100	
Order id. 86	2020-04-19 00:00:00	a@islingtoncollege.edu.np
2 * naya	@ Rs.120	
2 * Food F	@ Rs.100	
Order id. 85	2020-04-18 00:00:00	a@islingtoncollege.edu.np
1 * Food A	@ Rs.100	
1 * Food H	@ Rs.100	
Order id. 84	2020-04-18 00:00:00	a@islingtoncollege.edu.np
2 * naya	@ Rs.120	
Order id. 83	2020-04-18 00:00:00	a@islingtoncollege.edu.np
2 * Food H	@ Rs.100	
Order id. 82	2020-04-18 00:00:00	a@islingtoncollege.edu.np

Cant\_App

Not secure | cant.test/Transaction

Home Menu Orders Transactions Users admin

### Transactions

ID	Amount	User	Method	Description	Date
41	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 91	2020-04-19 16:50:39
40	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 90	2020-04-19 16:49:36
39	Rs. 120			cash payment at counter	2020-04-19 16:45:51
38	Rs. 200			cash payment at counter	2020-04-19 16:45:39
37	Rs. 100			cash payment at counter	2020-04-19 16:45:21
36	Rs. 200			cash payment at counter	2020-04-19 16:45:17
35	Rs. 100			cash payment at counter	2020-04-18 12:12:27
34	Rs. 240			cash payment at counter	2020-04-18 11:35:14
33	Rs. 200			cash payment at counter	2020-04-18 11:35:05
32	Rs. 200			cash payment at counter	2020-04-18 07:33:44
31	Rs. 100			cash payment at counter	2020-04-18 07:33:36
30	Rs. 200			cash payment at counter	2020-04-18 07:08:57
29	Rs. 300			cash payment at counter	2020-04-18 06:12:47
28	Rs. 120			cash payment at counter	2020-04-18 06:12:40
27	Rs. 200			cash payment at counter	2020-04-18 06:12:14
26	Rs. 100			cash payment at counter	2020-04-17 09:37:03
25	Rs. 100			cash payment at counter	2020-04-17 09:36:41
24	Rs. 120			cash payment at counter	2020-04-17 09:34:44
23	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 73	2020-04-15 17:18:27
22	Rs. 100	a@islingtoncollege.edu.np		food ordered via app; order id 72	2020-04-15 17:11:45

Name	Email	Balance
a@islingtoncollege.edu.np		0
b@b.b		0
CanteenAdmin@islingtoncollege.edu.np		0

Showing 1 - 3 of 3 entries

Previous Next

Transaction				Orders	
ID	Amount	Description	Date	order ID:	Date
41	Rs. 100	food ordered via app; order id 91 - order id 91	2020-04-19 16:50:39	91	2020-04-19 16:50:39 1* Food G @ Rs.100 2* Food B @ Rs.200 1* Food A @ Rs.100
40	Rs. 100	food ordered via app; order id 90 - order id 90	2020-04-19 16:49:36	90	2020-04-19 16:49:36 1* Food G @ Rs.100 2* Food B @ Rs.200 1* Food A @ Rs.100
39	Rs. 120	cash payment at counter - order id 89	2020-04-19 16:45:51	89	2020-04-19 16:45:51 1* Food G @ Rs.100 2* Food B @ Rs.200 1* Food A @ Rs.100
38	Rs. 200	cash payment at counter - order id 88	2020-04-19 16:45:39	88	2020-04-19 16:45:39 1* Food G @ Rs.100 2* Food B @ Rs.200 1* Food A @ Rs.100
37	Rs. 100	cash payment at counter - order id 87	2020-04-19 16:45:21	87	2020-04-19 16:45:21 1* naya @ Rs.120
36	Rs. 200	cash payment at counter - order id 86	2020-04-19 16:45:17	86	2020-04-19 16:45:17 order ID: 24 2020-04-05 07:18:22 1* Food G @ Rs.100
35	Rs. 100	cash payment at counter - order id 85	2020-04-18 12:12:27	85	2020-04-18 12:12:27 order ID: 14 2020-03-26 21:13:05 1* Food B @ Rs.200
34	Rs. 240	cash payment at counter - order id 84	2020-04-18 11:35:14	84	2020-04-18 11:35:14 order ID: 7 2020-03-21 05:15:24 1* Food A @ Rs.100
33	Rs. 200	cash payment at counter - order id 83	2020-04-18 11:35:05	83	2020-04-18 11:35:05 order ID: 2 2020-03-21 05:15:24 1* Food A @ Rs.100
32	Rs. 200	cash payment at counter - order id 82	2020-04-18 07:33:44	82	2020-04-18 07:33:44 order ID: 24 2020-04-05 07:18:22 1* Food B @ Rs.200
31	Rs. 100	cash payment at counter - order id 81	2020-04-18 07:33:36	81	2020-04-18 07:33:36 order ID: 14 2020-03-26 21:13:05 1* Food B @ Rs.200
30	Rs. 200	cash payment at counter - order id 80	2020-04-18 07:08:57	80	2020-04-18 07:08:57 order ID: 14 2020-03-26 21:13:05 1* Food B @ Rs.200

The image displays four screenshots of a software application interface, likely a food ordering system, showing code in the background and a user interface in the foreground.

**Screenshot 1:** Shows a complex code editor with multiple tabs open, displaying various files like `index.html`, `script.js`, and `style.css`. The code includes logic for generating random numbers and handling user input. The user interface shows a chart titled "Todays Top Sellers" with a bar chart and a line graph, and a table of "Orders (grouped by Food, time, user)".

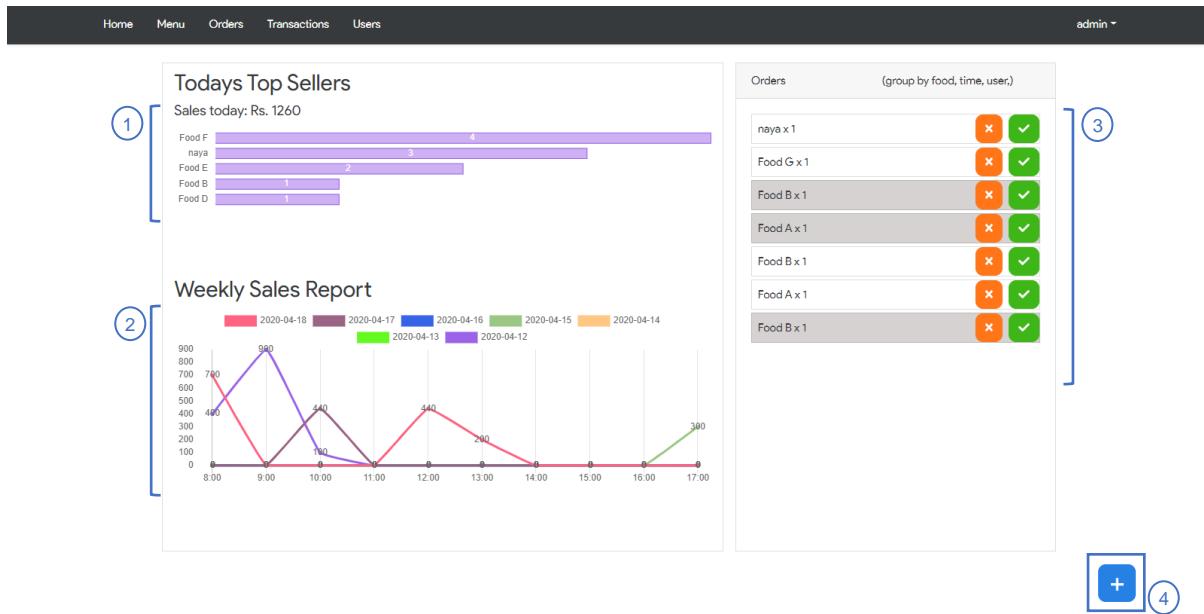
**Screenshot 2:** Shows the same code editor and user interface. The chart shows "Weekly Sales Report" with a line graph and a bar chart. The "Orders" table lists items like Food A, Food B, Food C, etc., grouped by user.

**Screenshot 3:** Shows the code editor and user interface. The chart shows "Weekly Sales Report" with a line graph and a bar chart. The "Orders" table lists items like Food A, Food B, Food C, etc., grouped by user.

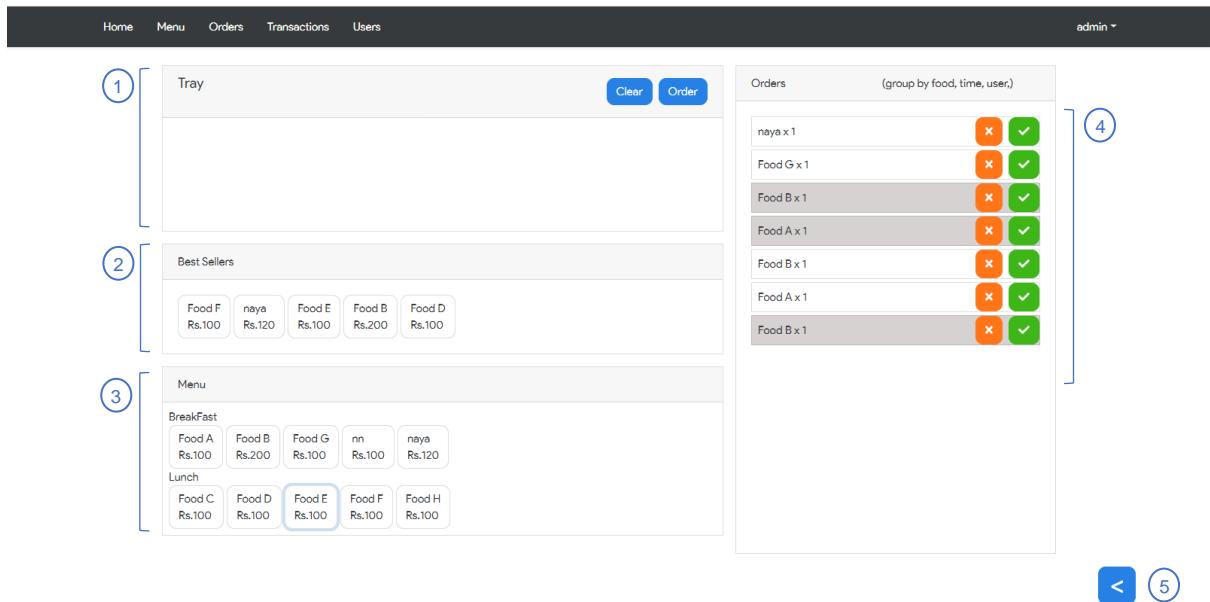
**Screenshot 4:** Shows the code editor and user interface. The chart shows "Weekly Sales Report" with a line graph and a bar chart. The "Orders" table lists items like Food A, Food B, Food C, etc., grouped by user.

## 8.6: USER MANUAL

### Home Page



The Home page can be accessed from the navigation bar of each page. It shows the over sales report per day (1) as well as hourly incomes over the past week (2). On the left side, a panel list the pending orders (3) which is automatically updated when a new order is received. New orders can be placed by clicking on the button (4) found on the bottom right.



Clicking the bottom right button swaps the stats page with the order page. The user can revert to the stats panel by clicking same button (5). The top section (1) is the tray which contains the selected items. Items can be added from the recommendation (2) or the menu (3) by clicking on the item itself. The orders panel (4) on the right remains unchanged.

The screenshot shows the application's main interface. At the top, there is a navigation bar with links for Home, Menu, Orders, Transactions, and Users, and a user dropdown labeled 'admin'. Below the navigation bar are three main panels:

- Tray:** A panel titled 'Tray' with a total value of 'Rs. 400'. It contains three items: Food G (1x100, Rs.100), Food F (2x100, Rs.200), and Food E (1x100, Rs.100). Each item has a quantity input field with a minus sign and an 'x' button.
- Best Sellers:** A panel listing five items: Food F (Rs.100), naya (Rs.120), Food E (Rs.100), Food B (Rs.200), and Food D (Rs.100).
- Menu:** A panel divided into Breakfast and Lunch sections. Breakfast includes Food A (Rs.100), Food B (Rs.200), Food G (Rs.100), nn (Rs.100), and naya (Rs.120). Lunch includes Food C (Rs.100), Food D (Rs.100), Food E (Rs.100), Food F (Rs.100), and Food H (Rs.100). The 'Food E' button in the menu is highlighted with a blue border.

At the bottom right of the interface is a blue back arrow button.

This screenshot is similar to the one above, but with numbered annotations:

- Annotation 1 points to the 'Tray' panel.
- Annotation 2 points to the 'Food E' button in the 'Menu' panel.
- Annotation 3 points to the 'Clear' and 'Order' buttons at the top right of the 'Tray' panel.
- Annotation 4 points to the 'Orders' panel.
- Annotation 5 points to the status buttons ('x' and '✓') next to the order items in the 'Orders' panel.

Clicking the items on the recommendation or the menu adds the item to the tray (1). The same item can be added either by clicking the same button or the items on the tray themselves (2). The items in the tray have more option. Clicking on the item increases the quantity, clicking on the ‘ - ’ button reduces it while clicking on ‘ x ’ removes the item altogether. The total is displayed at the top left whereas the top right panel (3) has options to either clear the tray or place the selected orders.

Once ordered, the items are listed on the right panel (4) which can be used to update the status and notify the customers. Clicking on the item itself sets the item as “cooking” and are marked grey. Clicking the buttons (5) ‘ x ’ and ‘ ✓ ’ will cancel or complete the order and notify the users accordingly.

## Menu Page

The screenshot shows a web-based application interface for managing a menu. At the top, there is a navigation bar with links for Home, Menu, Orders, Transactions, and Users. On the right side of the header is a user account dropdown labeled "admin". The main content area is divided into three sections:

- Central Column (1):** This section is titled "Menu" and contains two lists: "BreakFast" and "Lunch". Each list contains several items, each with a checkbox. For example, under "BreakFast", there are entries like "Food A: 100" and "Food B: 200". Under "Lunch", there are entries like "Food C: 100" and "Food D: 100".
- Right Panel (2):** This panel is titled "Create Food" and contains fields for Name, Category (with a dropdown menu), Price, and Description. There is also a "Create" button.
- Panel Below (3):** This panel is titled "Create category" and contains a single field for Name and a "Create" button.

Blue circles and numbers (1, 2, 3) are overlaid on the interface to point to these specific components.

The menu creation page can be accessed from the menu tab in the navigation. The page lists all created food items in the central component (1). The items that are available on the menu are checked (). User can change the availability of the item by simply checking or unchecking the boxes for each item. Clicking on the item leads to a page containing more detailed information of the item where it can also be edited if necessary. The panel on side (2) can be used to create new food items while the panel below (3) can be used to create new category

## Transaction Page

Transactions					
ID	Amount	User	Method	Description	Date
46	Rs. 100		cash payment at counter	2020-04-20 09:27:34	
48	Rs. 100		cash payment at counter	2020-04-20 09:27:24	
44	Rs. 100		cash payment at counter	2020-04-20 09:27:14	
43	Rs. 100		cash payment at counter	2020-04-20 09:27:07	
42	Rs. 100		cash payment at counter	2020-04-20 09:27:03	
41	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 91	2020-04-19 16:50:39	
40	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 90	2020-04-19 16:49:36	
39	Rs. 120		cash payment at counter	2020-04-19 16:45:51	
38	Rs. 200		cash payment at counter	2020-04-19 16:45:39	
37	Rs. 100		cash payment at counter	2020-04-19 16:45:21	
36	Rs. 200		cash payment at counter	2020-04-19 16:45:17	
35	Rs. 100		cash payment at counter	2020-04-18 12:22:27	
34	Rs. 240		cash payment at counter	2020-04-18 11:35:14	
33	Rs. 200		cash payment at counter	2020-04-18 11:35:05	
32	Rs. 200		cash payment at counter	2020-04-18 07:33:44	
31	Rs. 100		cash payment at counter	2020-04-18 07:33:36	
30	Rs. 200		cash payment at counter	2020-04-18 07:08:57	
29	Rs. 300		cash payment at counter	2020-04-18 06:12:47	
28	Rs. 120		cash payment at counter	2020-04-18 06:12:40	
27	Rs. 200		cash payment at counter	2020-04-18 06:12:14	
26	Rs. 100		cash payment at counter	2020-04-17 09:37:03	
25	Rs. 100		cash payment at counter	2020-04-17 09:36:41	
24	Rs. 120		cash payment at counter	2020-04-17 09:34:44	
23	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 73	2020-04-15 17:18:27	
22	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 72	2020-04-15 17:11:45	
21	Rs. 100		cash payment at counter	2020-04-15 16:58:16	
20	Rs. 100	a@islingtoncollege.edu.np	Food ordered via app; order id 70	2020-04-12 09:04:06	
19	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 69	2020-04-12 08:25:53	
18	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 68	2020-04-12 08:25:22	
17	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 67	2020-04-12 08:24:23	
16	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 66	2020-04-12 08:22:33	
15	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 65	2020-04-12 08:22:12	
14	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 64	2020-04-12 08:20:10	
13	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 63	2020-04-12 08:19:00	
12	Rs. 100	a@islingtoncollege.edu.np	food ordered via app; order id 62	2020-04-12 08:17:06	

The transaction page accessible from the navigation bar list all the transactions till date. More specific data can be seen on individual user pages.

## Orders Page

All Orders					
Order id: 96	2020-04-20 00:00:00	a@islingtoncollege.edu.np			
2 * Food F @ Rs.100					
1 * Food E @ Rs.100					
1 * Food D @ Rs.100					
Order id: 95	2020-04-20 00:00:00	a@islingtoncollege.edu.np			
1 * Food F @ Rs.100					
1 * Food E @ Rs.100					
Order id: 94	2020-04-20 00:00:00	a@islingtoncollege.edu.np			
1 * naya @ Rs.120					
1 * Food F @ Rs.100					
1 * Food G @ Rs.100					
Order id: 93	2020-04-20 00:00:00	a@islingtoncollege.edu.np			
1 * Food H @ Rs.100					
1 * m @ Rs.100					
1 * Food E @ Rs.100					
Order id: 92	2020-04-20 00:00:00	a@islingtoncollege.edu.np			
1 * naya @ Rs.120					
1 * Food F @ Rs.100					
1 * Food G @ Rs.100					
Order id: 91	2020-04-19 00:00:00	a@islingtoncollege.edu.np			
1 * Food A @ Rs.100					
2 * Food B @ Rs.200					
1 * Food G @ Rs.100					

The Orders page, similar to transaction, list all the orders sorted according to time. Also, more specific data can be seen on individual user pages.

## Users Page

The screenshot shows a table of users with columns for Name, Email, and Balance. Three annotations are present: a blue circle labeled 1 points to the search input field; a blue bracket labeled 2 points to the second user row; and a blue bracket labeled 3 points to the 'Previous' and 'Next' buttons at the bottom.

Name	Email	Balance
a@lslingtoncollege.edu.np		0
b@b.b		0
CanteenAdmin@canteenadmin@lslingtoncollege.edu.np		0

Showing 1 - 3 of 3 entries

Previous Next

Users page lists all the user registered to the system (2). The user can search through them using the search field (1) at the top and each individual data can be viewed by clicking directly on the user row. If the users on the list exceeds a certain limit they are automatically paginated then user can move across pages with the buttons (3) at the bottom right.

## Individual User Page

The screenshot shows the details for user 'b@b.b'. A blue circle labeled 1 highlights the user's email and balance. A large blue bracket labeled 2 covers the 'Transactions' section, which lists numerous cash payments. A blue bracket labeled 3 covers the 'Orders' section, which lists food items ordered by the user.

**Transaction**

ID	Amount	Description	Date
46	Rs. 100	cash payment at counter -order id 96	2020-04-20 09:27:34
45	Rs. 100	cash payment at counter -order id 95	2020-04-20 09:27:24
44	Rs. 100	cash payment at counter -order id 94	2020-04-20 09:27:14
43	Rs. 100	cash payment at counter -order id 93	2020-04-20 09:27:07
42	Rs. 100	cash payment at counter -order id 92	2020-04-20 09:27:03
41	Rs. 100	food ordered via app; order id 91 -order id 91	2020-04-19 16:50:39
40	Rs. 100	food ordered via app; order id 90 -order id 90	2020-04-19 16:49:36
39	Rs. 120	cash payment at counter -order id 89	2020-04-19 16:45:51
38	Rs. 200	cash payment at counter -order id 88	2020-04-19 16:45:39
37	Rs. 100	cash payment at counter -order id 87	2020-04-19 16:45:21
36	Rs. 200	cash payment at counter -order id 86	2020-04-19 16:45:17
35	Rs. 100	cash payment at counter -order id 85	2020-04-19 12:12:27
34	Rs. 240	cash payment at counter -order id 84	2020-04-19 11:35:14
33	Rs. 200	cash payment at counter -order id 83	2020-04-19 11:35:05
32	Rs. 200	cash payment at counter -order id 82	2020-04-19 07:33:44
31	Rs. 100	cash payment at counter -order id 81	2020-04-19 07:33:36
30	Rs. 200	cash payment at counter -order id 80	2020-04-19 07:08:57
29	Rs. 300	cash payment at counter -order id 79	2020-04-19 06:12:47
28	Rs. 120	cash payment at counter -order id 78	2020-04-19 06:12:40
27	Rs. 200	cash payment at counter -order id 77	2020-04-19 06:12:14
26	Rs. 100	cash payment at counter -order id 76	2020-04-17 09:37:03
25	Rs. 100	cash payment at counter -order id 75	2020-04-17 09:36:41
24	Rs. 120	cash payment at counter -order id 74	2020-04-17 09:34:44
23	Rs. 100	food ordered via app; order id 73 -order id 73	2020-04-19 17:18:27
22	Rs. 100	food ordered via app; order id 72 -order id 72	2020-04-19 17:11:45
21	Rs. 100	cash payment at counter -order id 71	2020-04-19 16:58:16
20	Rs. 100	food ordered via app; order id 70 -order id 70	2020-04-12 09:04:06

**Orders**

order ID	Food Item	Date
94	Food F	2020-04-20 09:27:14 @ Rs.100
93	nn	2020-04-20 09:27:07 @ Rs.100
nn	G	2020-04-05 07:18:22 @ Rs.100
14	B	2020-03-26 21:13:05 @ Rs.200
7	A	2020-03-21 05:15:24 @ Rs.100
8	B	2020-04-19 11:35:14 @ Rs.200
9	A	2020-04-19 11:35:05 @ Rs.100
10	B	2020-04-19 07:33:44 @ Rs.200

The Individual page contains the details of the user (1). Below the details the total transactions (2) and Orders (3) made by the selected user are listed.