LONDON METROPOLITAN UNIVERSITY

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: 5th June 2020

Assignment Submission Date: 4th 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

Figures

Tables

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 recepts that won’t have to be printed, it might as well be saving the environment while reducing paper costs.

1.4AIM 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 as system through API calls and receives a response, usually, in JSON format.

State management is as 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 method 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 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.

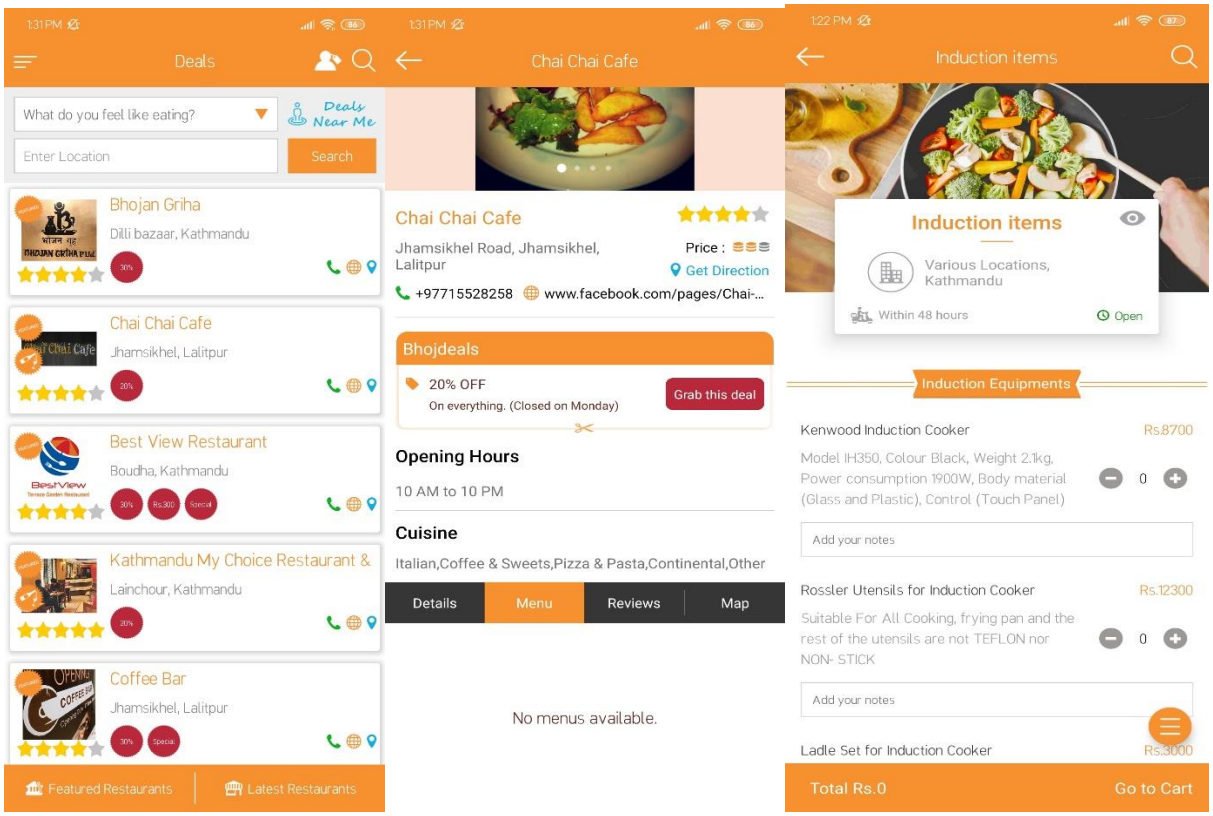


Figure 1: Bhoj app

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 out 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

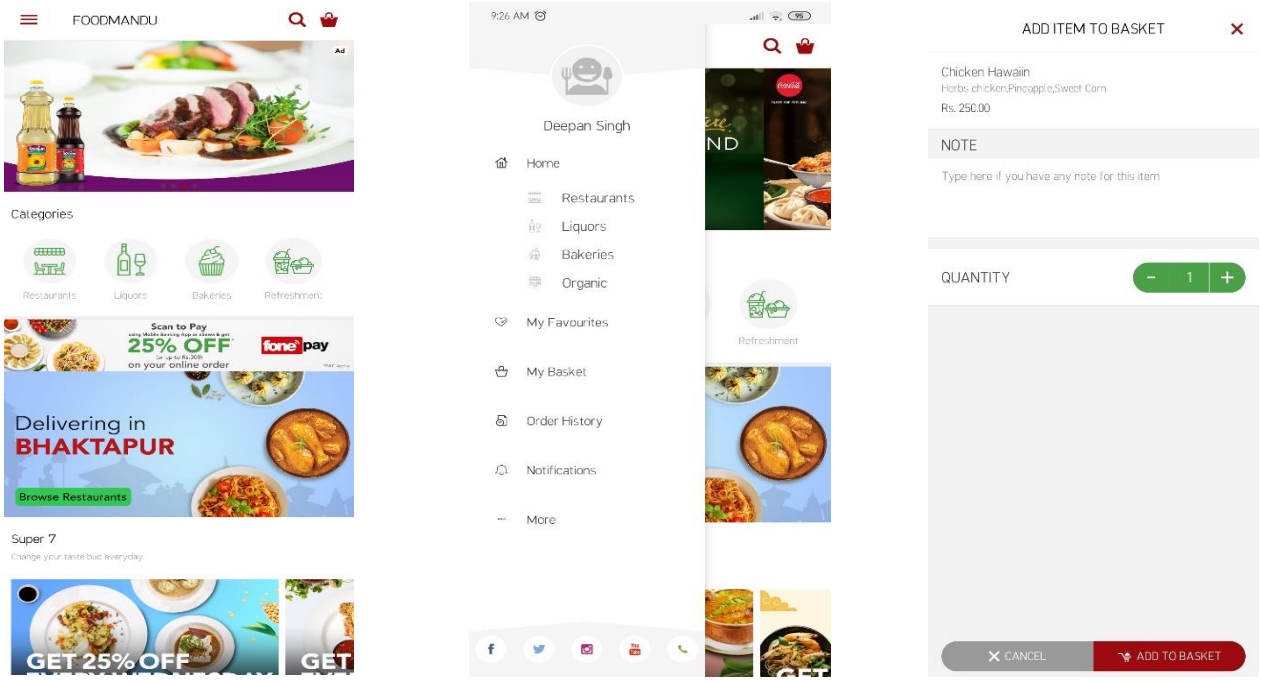


Figure 2: Foodmandu app

(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. Tor 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 or 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 me 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 USDP 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 METHODOLGY

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.

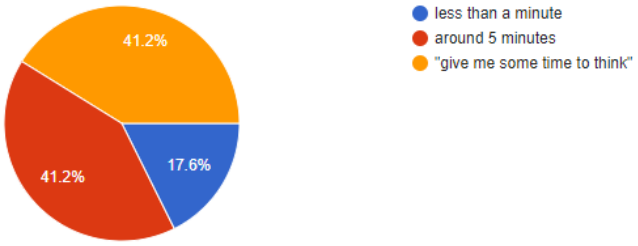
The final Transition phase delivers the iteration into the production environment.

3.4 INCEPTION

3.4.1 PRE-DEVELOPMENT SURVEY RESULTS

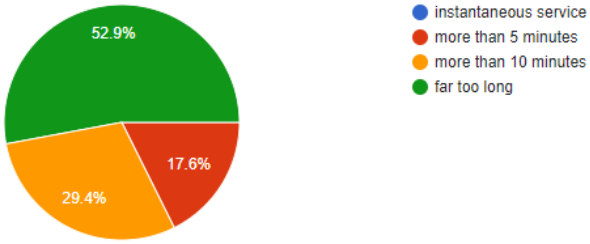
How long do you usually spend choosing meals in-front of the counter?

17 responses



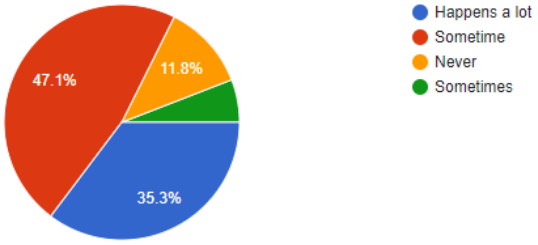
How long do you usually wait for meals once ordered in the college cafeteria?

17 responses



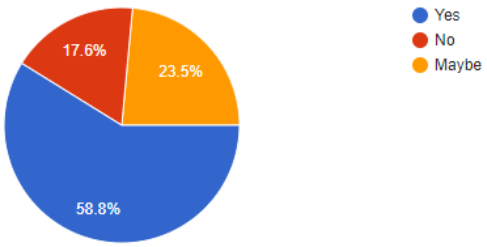
Have you had to move elsewhere because the cafeteria was too crowded?

17 responses



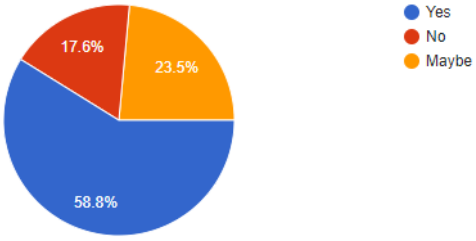
Do you have trouble deciding what to order often?

17 responses



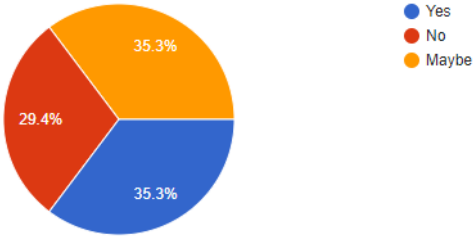
Does what other people order change your mind?

17 responses



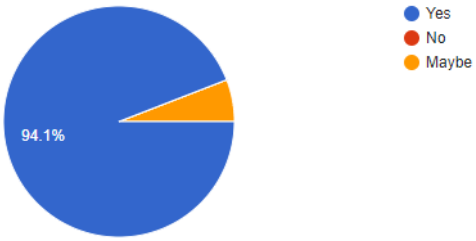
do you tend to avoid interaction with strangers?

17 responses



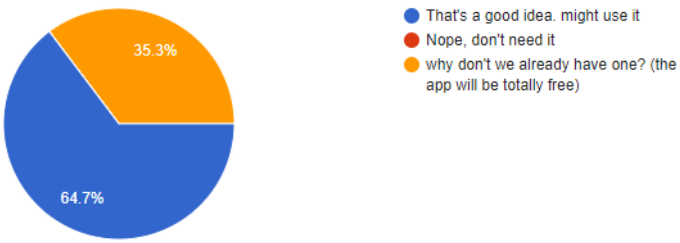
Would you want someone to place your orders and notify you when its ready?

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
UC-1 **Customer (Mobile)**	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
UC-2 **Staff (web)**	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	**User will enter authentication details and log into their account**	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	**User can view list of available Menu Item**	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	**The application recommends items to the user**	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	**User will be able to view al order history**	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	**User will be able to view al Payment history**	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	**Staff will enter authentication details and log into their account**	High	High
SR-01	when logging in only valid admin accounts must be authorized		
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	**Staff can view list of Menu Item**	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		
	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	**The system generates and displays the daily and weekly statistics**	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	**Based on the Statistics the application recommends bestselling items**	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	**Staff 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		
	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	**Staff can view all pending orders**	High	Low
	Staff can change the status of pending items		
	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	**Staff can view order history of all users**	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	**Staff will be able to view al Payment history**	Low	High

Table 18: PFW-3 Payment

PFM-9 User

Req. ID	Requirement Description	Priority	Complexity
FR-05	**Staff can view list of all users**	High	High
	The list of user can be searched through.		
	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

Tasks	Details	Description	Approximate Duration
Initiation	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
Database	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
Web app	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
Mobile App	initial mobile application	Create a UI using flutter	4 weeks
	integration	Integration between database, Web and mobile app	4 weeks
Prediction Algorithms	Research and Implement	Research various algorithms and create an implementable	4 weeks
Final Testings	Unit Test	Test functionality of each unit.	2 weeks
	Integration Test	Test functionality of integration with each other.	2 weeks
Documentation	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
Entity diagrams	visualization depicting relation between necessary table	Oct-20
Databases	Creation of table models which can be translated to another desktop if necessary	Nov-01
API	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
App	initial completion of mobile app	Dec-20
Web	add remaining features to web app	Jan-01
Notification	Add notification services to notify when meals are ready	Jan-10
UI	finalize UI	Jan-20
Eloquence	Optimize the code	Feb-20
Prediction algorithm	implement prediction algorithms	Mar-01
Beta Test	Initial release of the app	Mar-25
Documentation	Fully revised and completed documentation	Apr-20

Table 21: Milestones

3.5.3 Gantt Chart

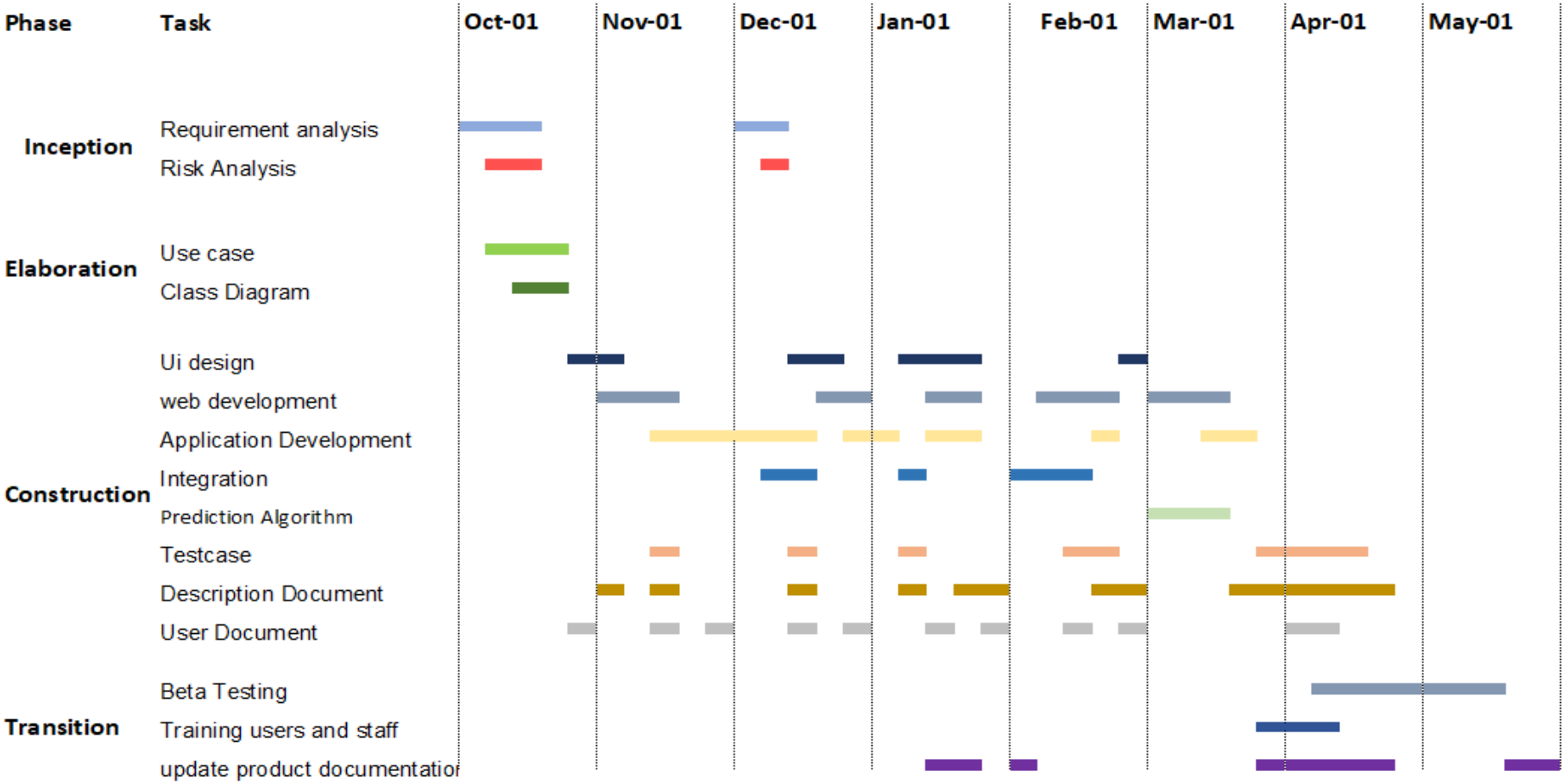


Figure 3: Gantt Chart

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 mot 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

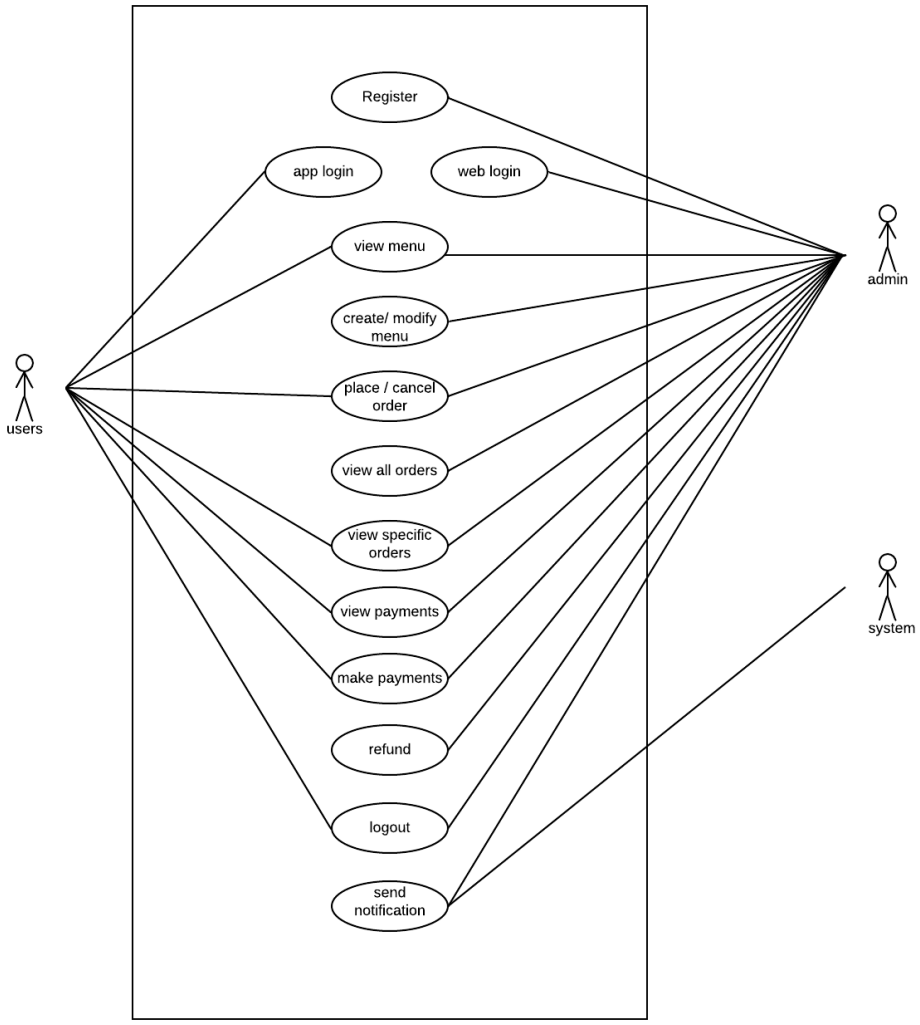


Figure 4: Use case diagram - First Iteration

Entity Relation Diagram

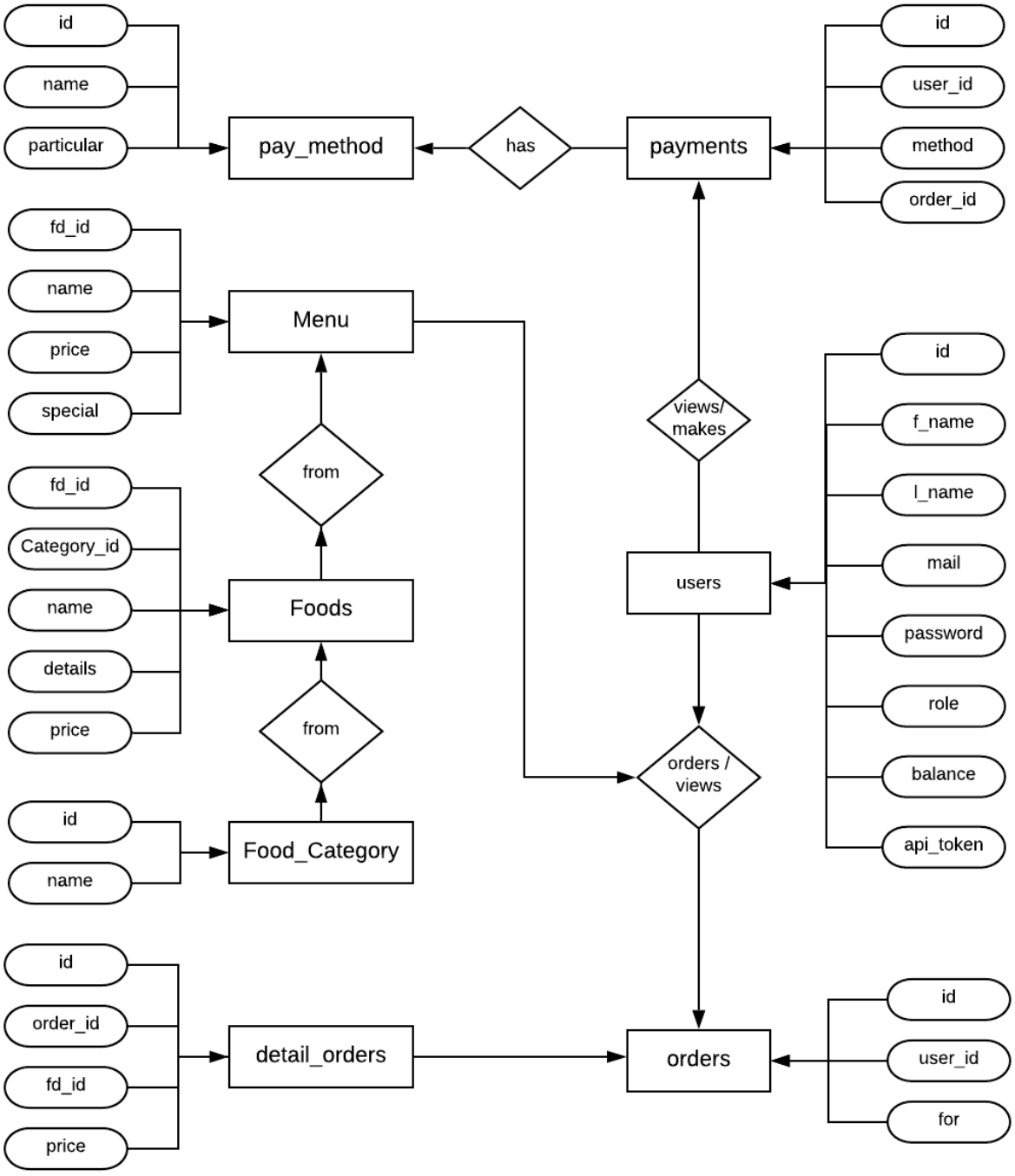


Figure 5: ERD - first iteration

Data Dictionary

users					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
id	Primary	Account identifier	Integer	8	Yes
name	unique	Name to Category,	String	50	Yes

Table 25: Food categories

orders					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
id	Unique	Order identifier	Int	11	Yes
user_id		Uer 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. IMPLEMENATION

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.

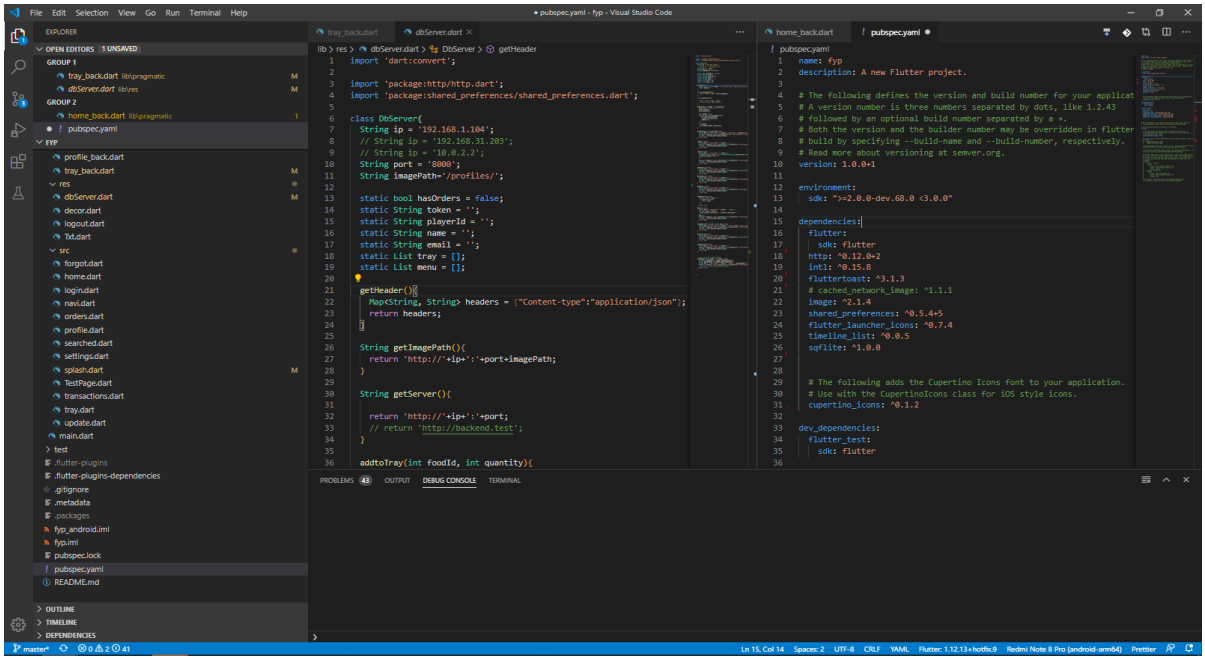


Figure 6: Implementation Screenshot (API connectors(left) and packages used(right))

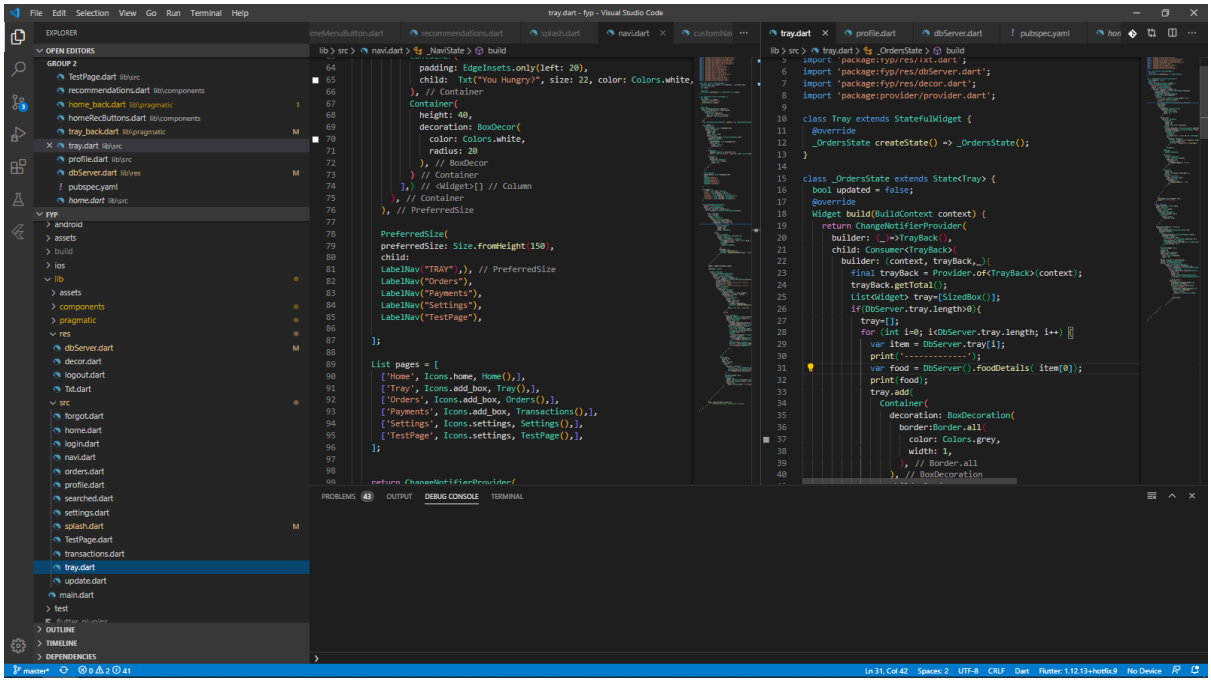


Figure 7: Implementation Screenshot (Navigation page(left) and tray page code(right))

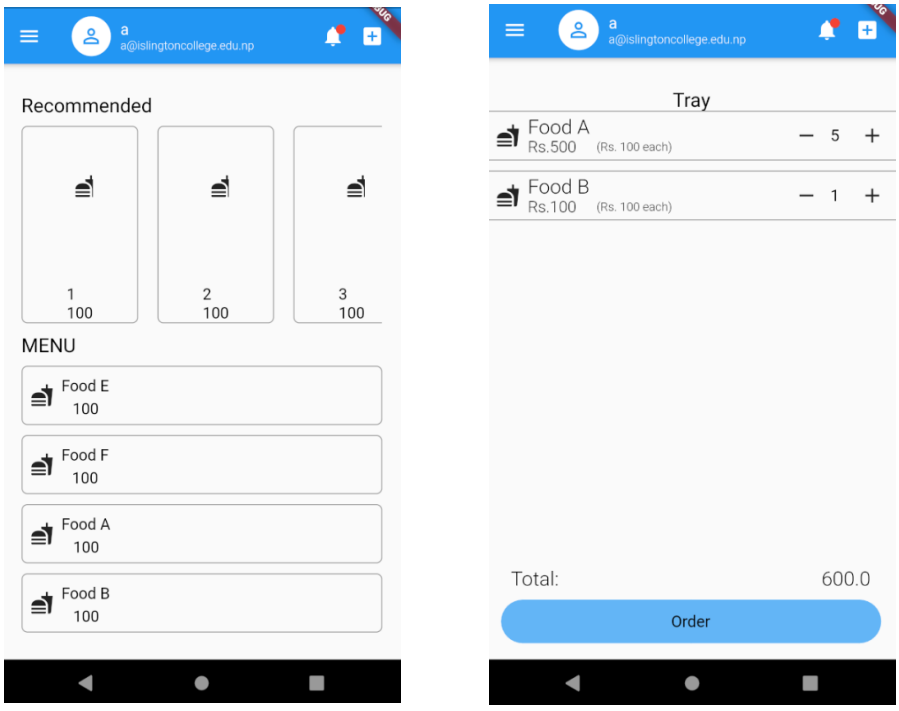


Figure 8: Implementation Screenshot (Home page(left) and Tray(right))

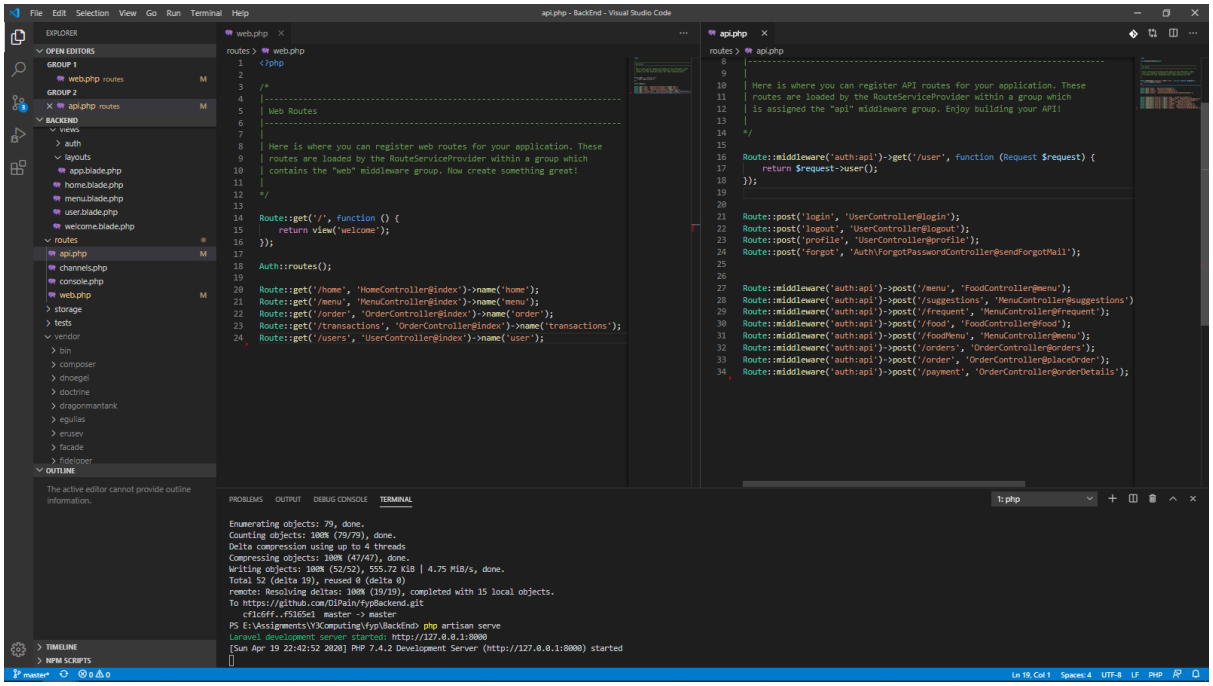


Figure 9: Implementation Screenshot (web routes(left) and API (right))

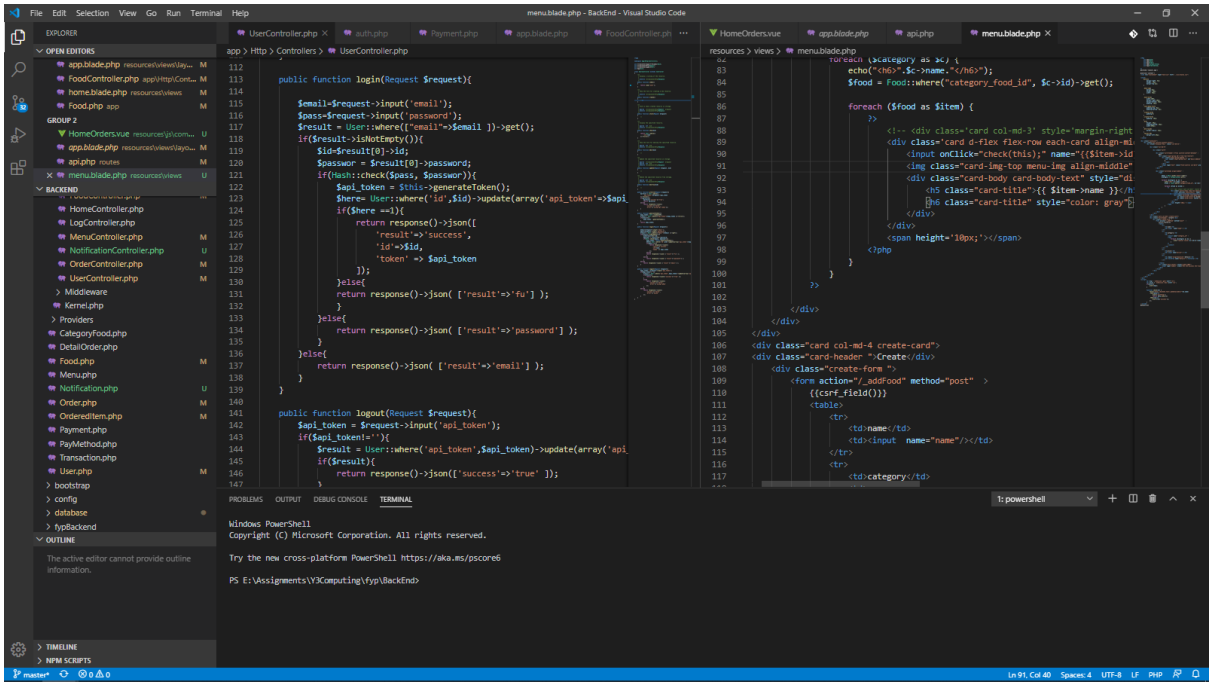


Figure 10: Implementation Screenshot (User Controller(left) and web menu code(right))

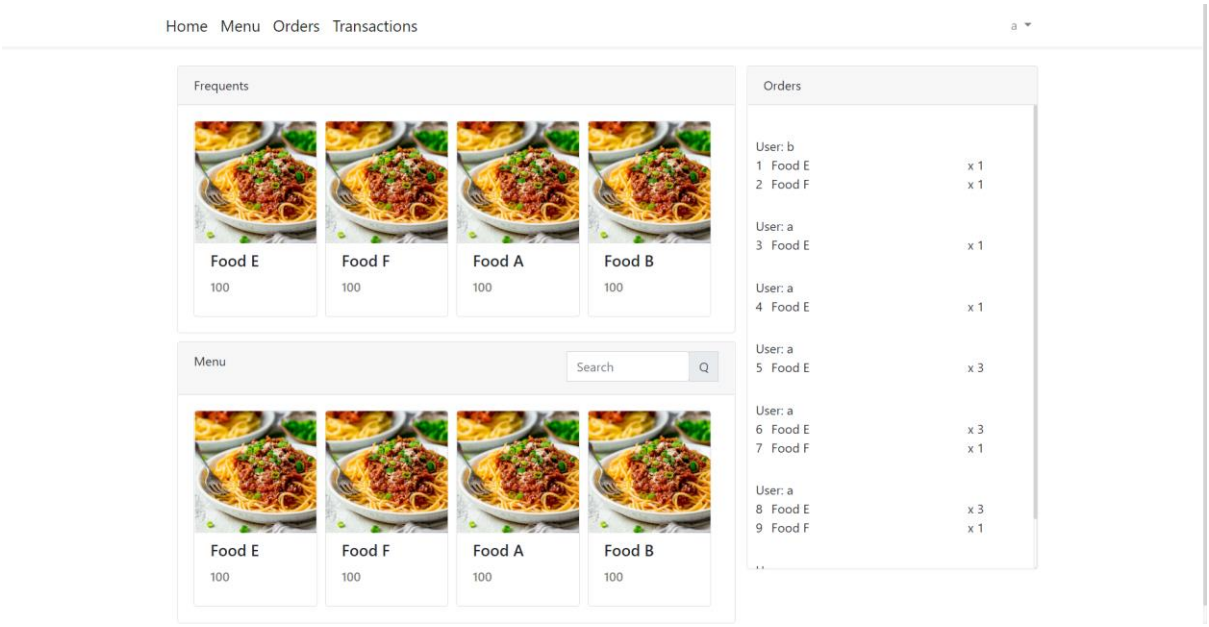


Figure 11: Implementation Screenshot (web home page)

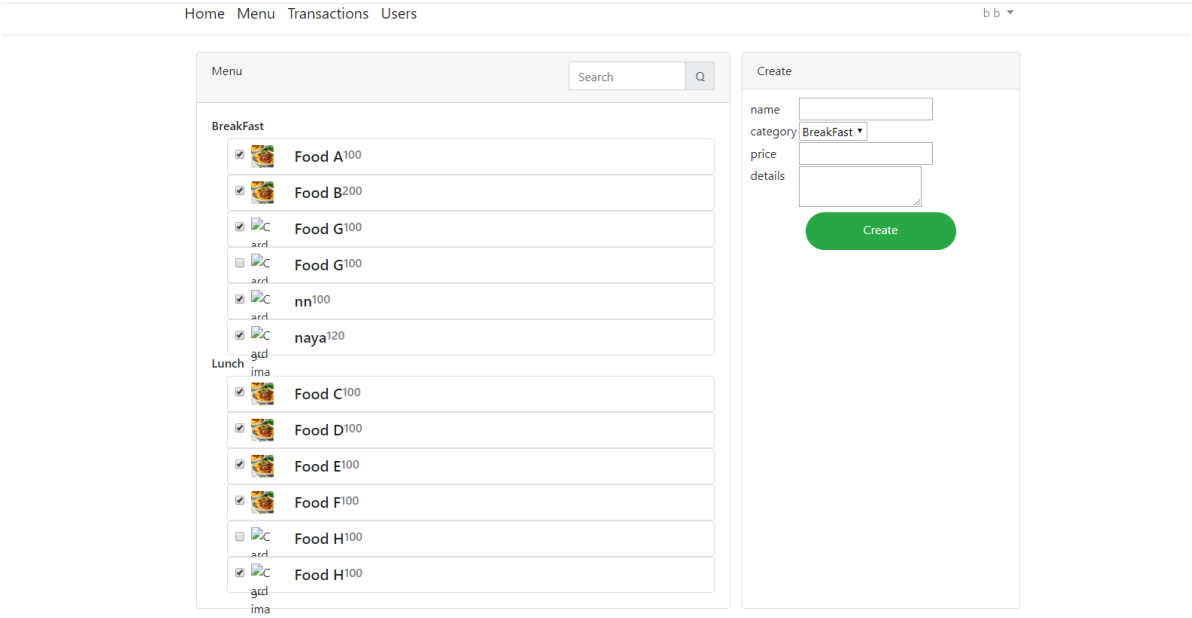


Figure 12: Implementation Screenshot (Menu creation and edit page)

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

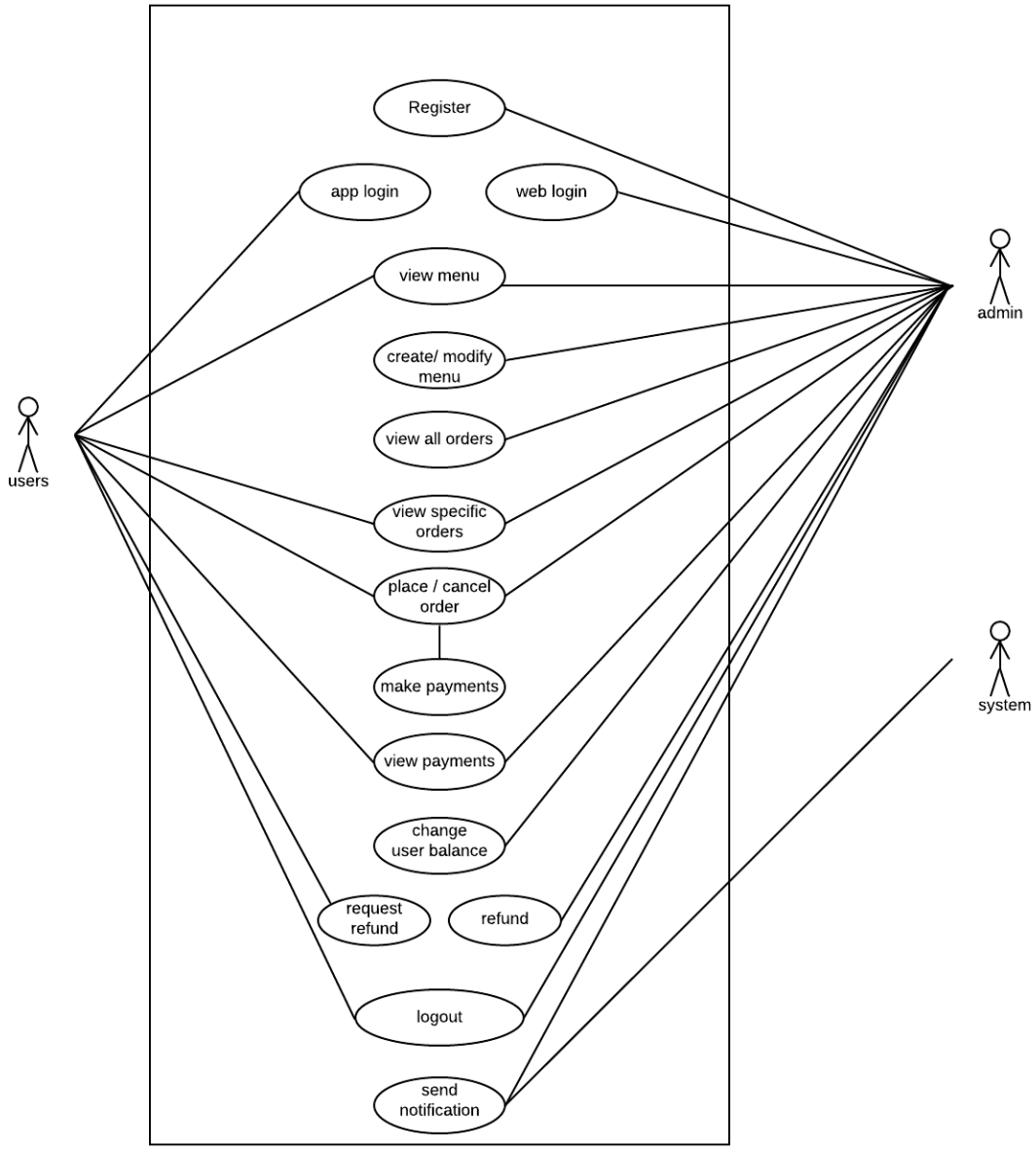


Figure 13: Use Case - Second Iteration

Entity Relational Diagram

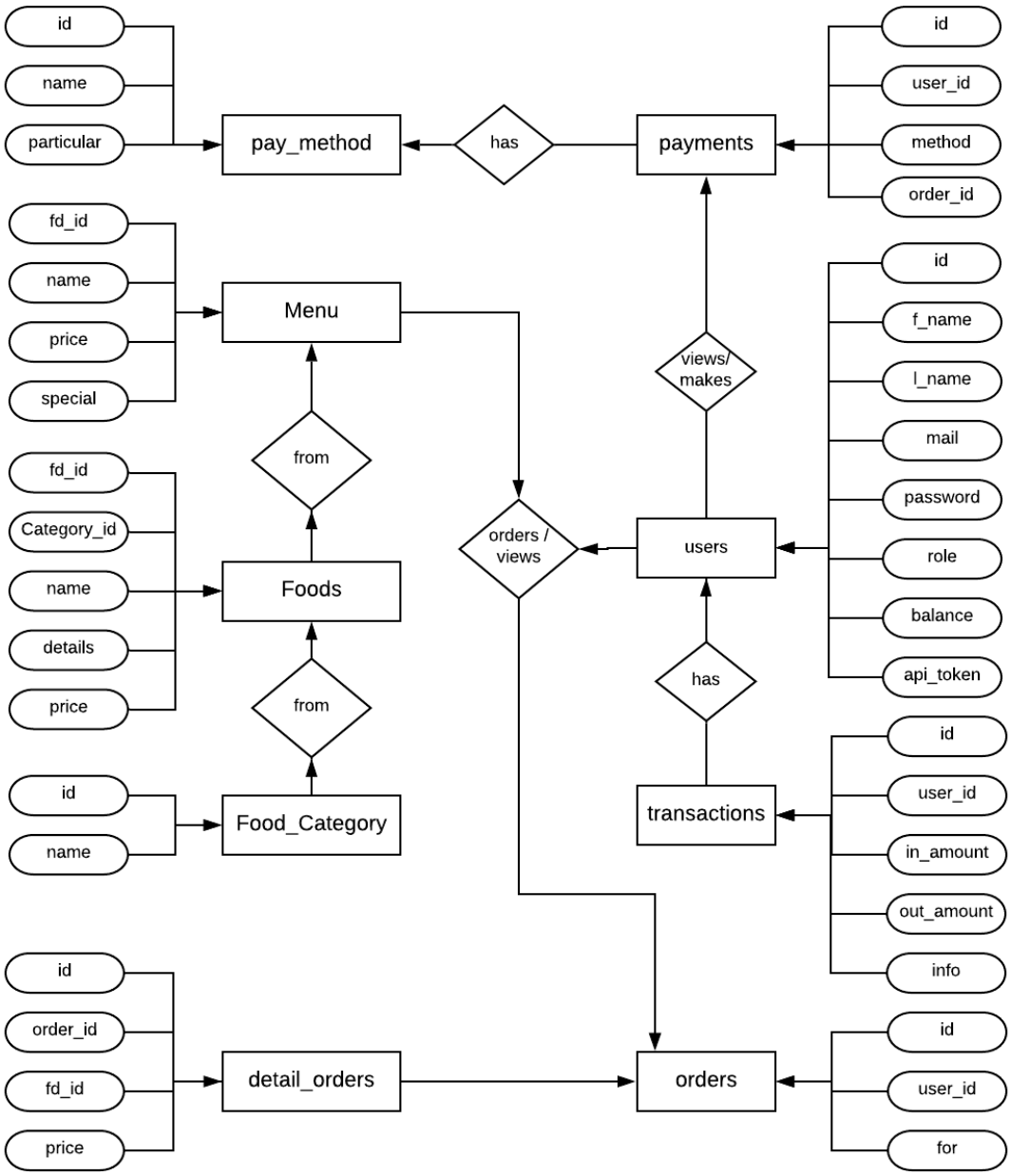
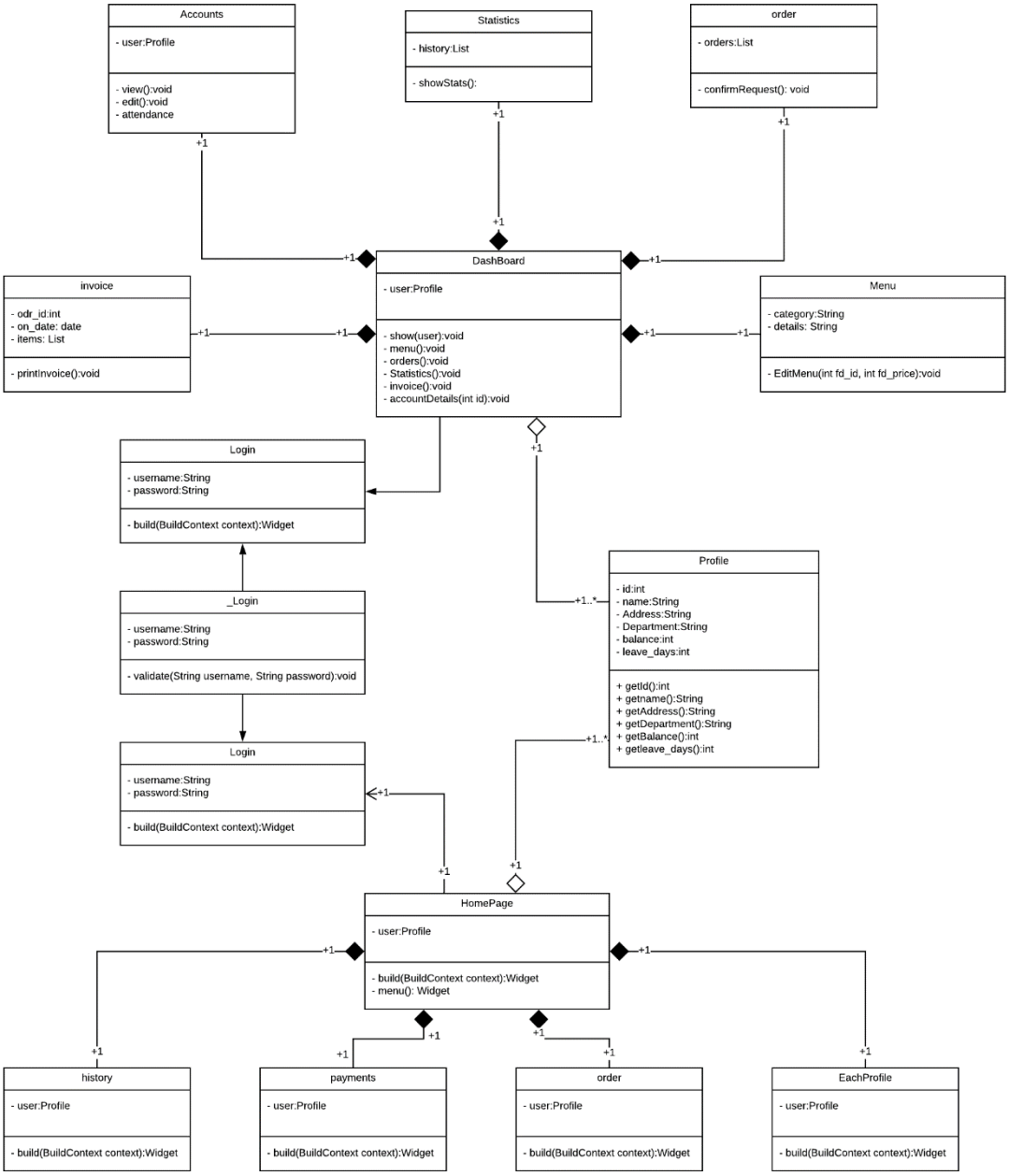


Figure 14: ERD – Second Iteration


Accounts
- user:Profile
- view():void
- edit():void
- attendance
Statistics
- history:List
- showStats():
order
- orders:List
- confirmRequest(): void
DashBoard
- user:Profile
- show(user):void
- menu():void
- orders():void
- Statistics():void
- invoice():void
- accountDetails(int id):void
invoice
- odr_id:int
- on_date: date
- items: List
- printInvoice():void
Menu
- category:String
- details: String
- EditMenu(int fd_id, int fd_price):void
Login
- username:String
- password:String
- build(BuildContext context):Widget
_Login
- username:String
- password:String
- validate(String username, String password):void
Profile
- id:int
- name:String
- Address:String
- Department:String
- balance:int
- leave_days:int
+ getId():int
+ getname():String
+ getAddress():String
+ getDepartment():String
+ getBalance():int
+ getleave_days():int
Login
- username:String
- password:String
- build(BuildContext context):Widget
HomePage
- user:Profile
- build(BuildContext context):Widget
- menu(): Widget
history
- user:Profile
- build(BuildContext context):Widget
payments
- user:Profile
- build(BuildContext context):Widget
order
- user:Profile
- build(BuildContext context):Widget
EachProfile
- user:Profile
- build(BuildContext context):Widget

Data Dictionary

users					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
id	Primary	Account identifier	Integer	8	Yes
name	unique	Name to Category,	String	50	Yes

Table 33: Food categories

orders					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
id	Unique	Order identifier	Int	11	Yes
user_id		Uer id	Int	11	Yes
method_id		Food id	Int	11	Yes
Order_id		Price when ordered	int	11	Yes

Table 37: Payments

Notifications					
name	**key**	**Description**	**Datatype**	**Length**	**Required**
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. IMPLEMENATION

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

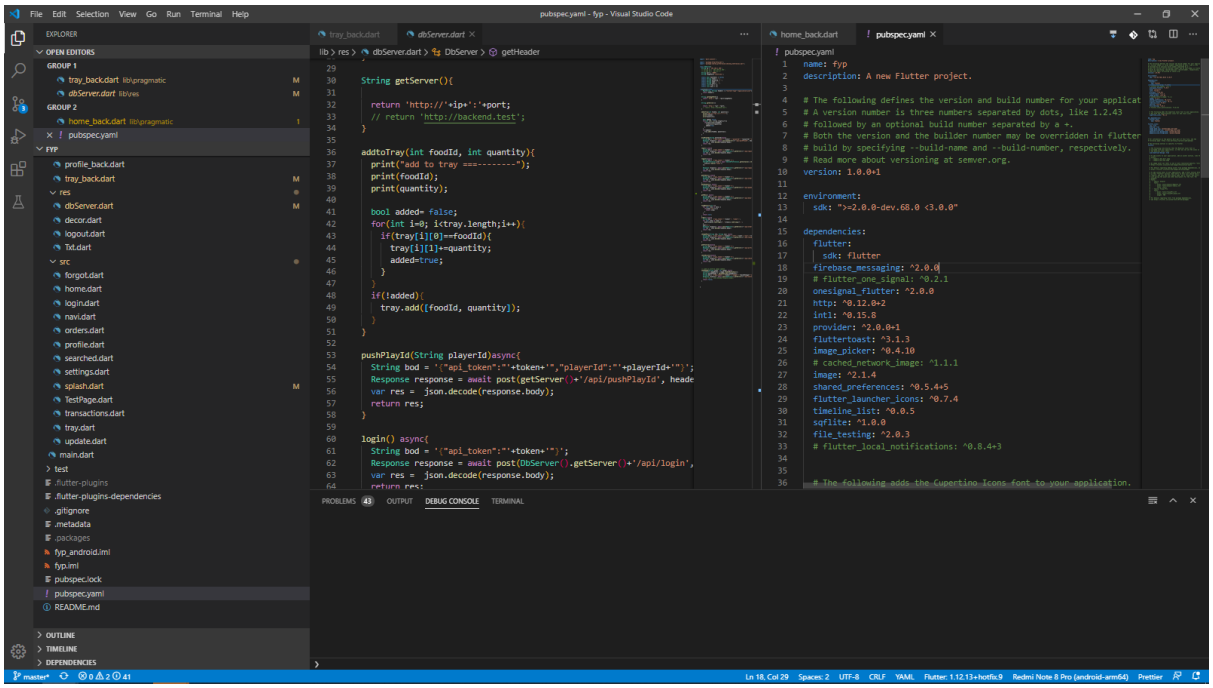


Figure 15: Implementation Screenshot (API Connector (left) and flutter packages user(right))

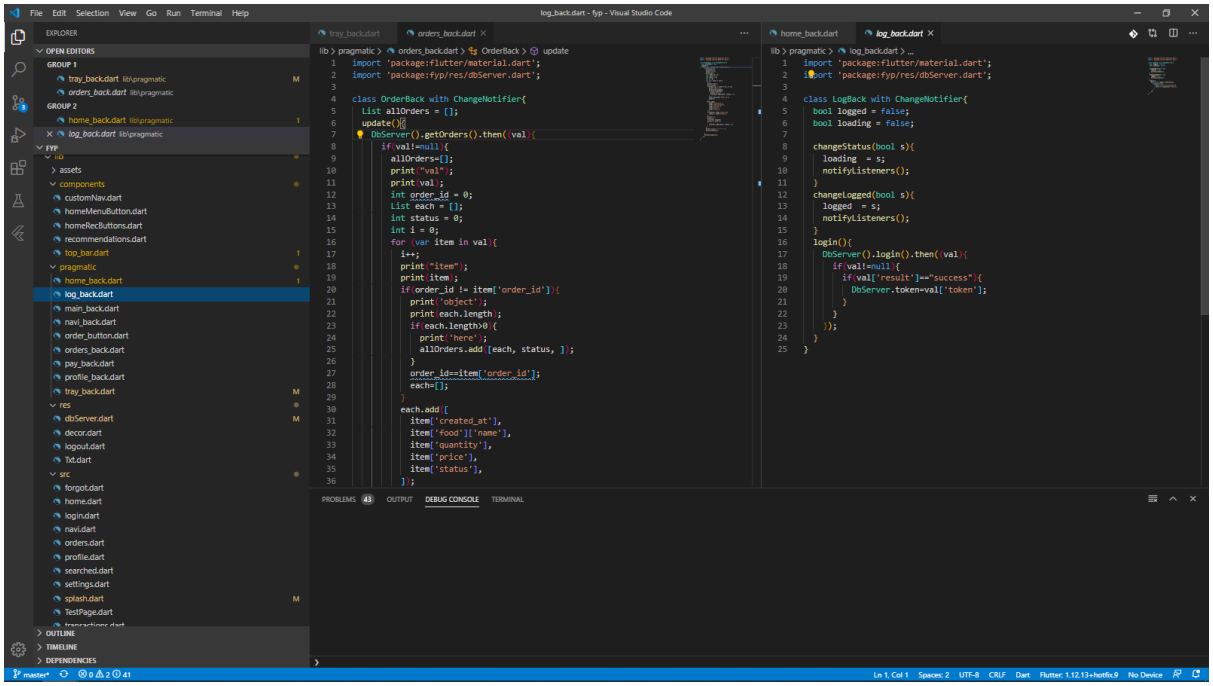


Figure 16: Implementation Screenshot (Orders logic code(left) and login logic code(right))

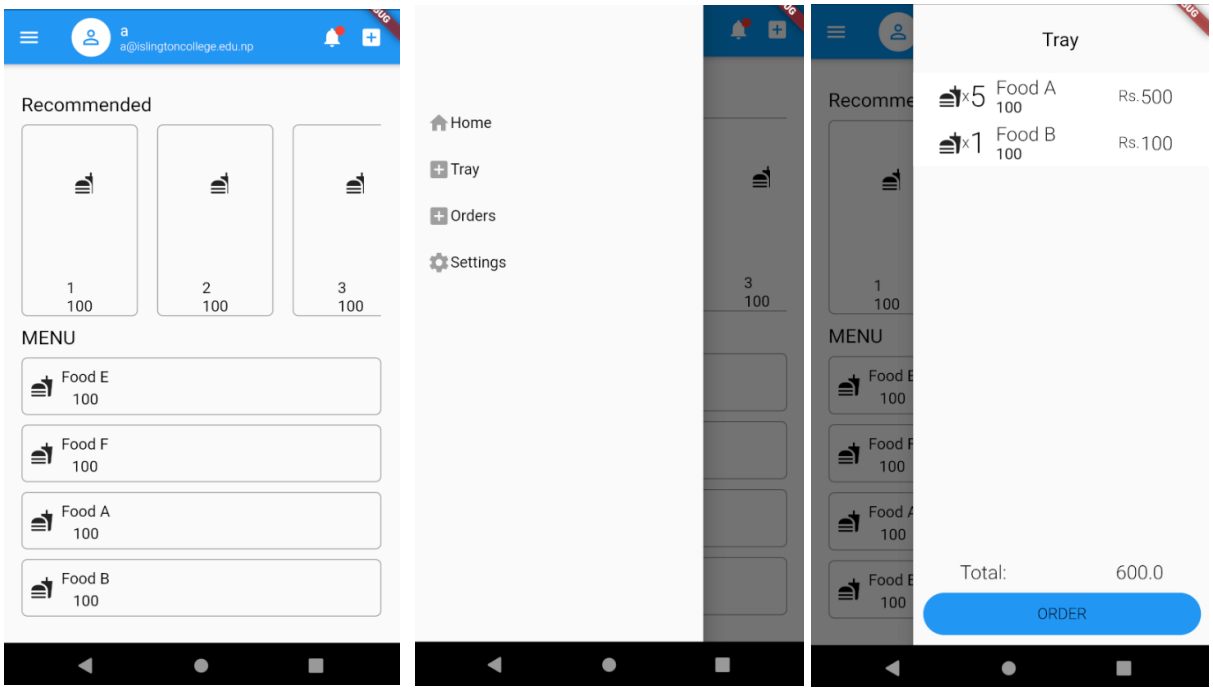


Figure 17: App in development(from left: home, options drawer, tray drawer)

Web App

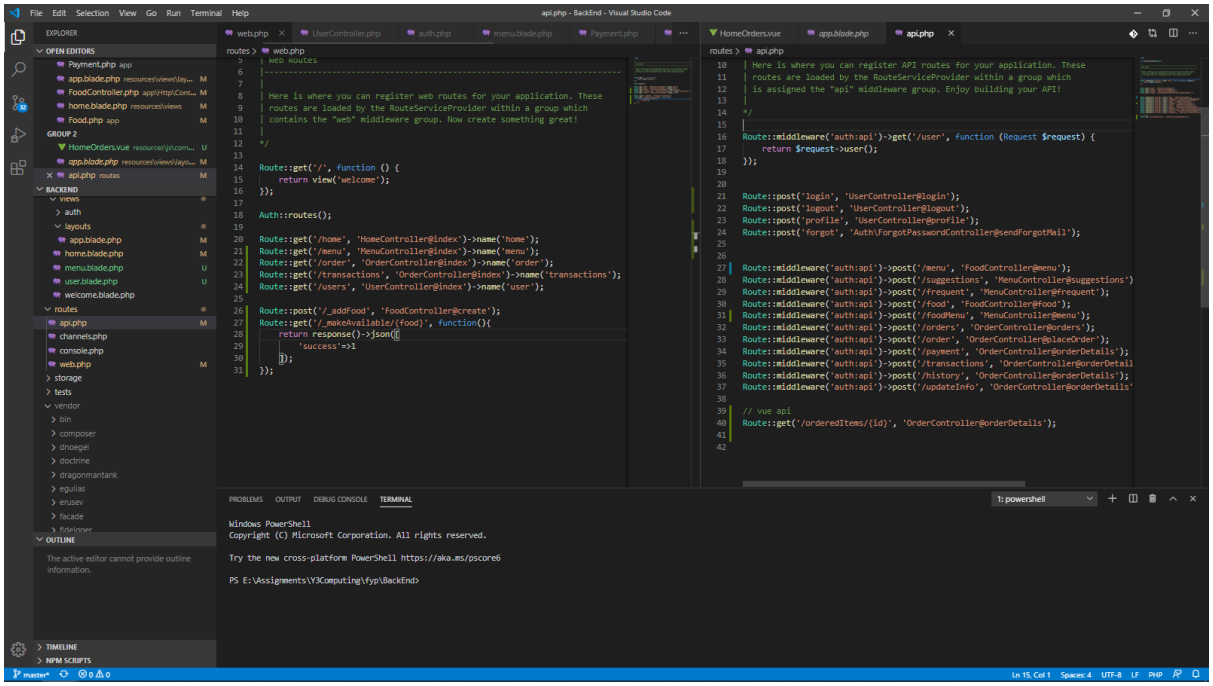


Figure 18: Implementation Screenshot (Web routes (left) API routes(right))

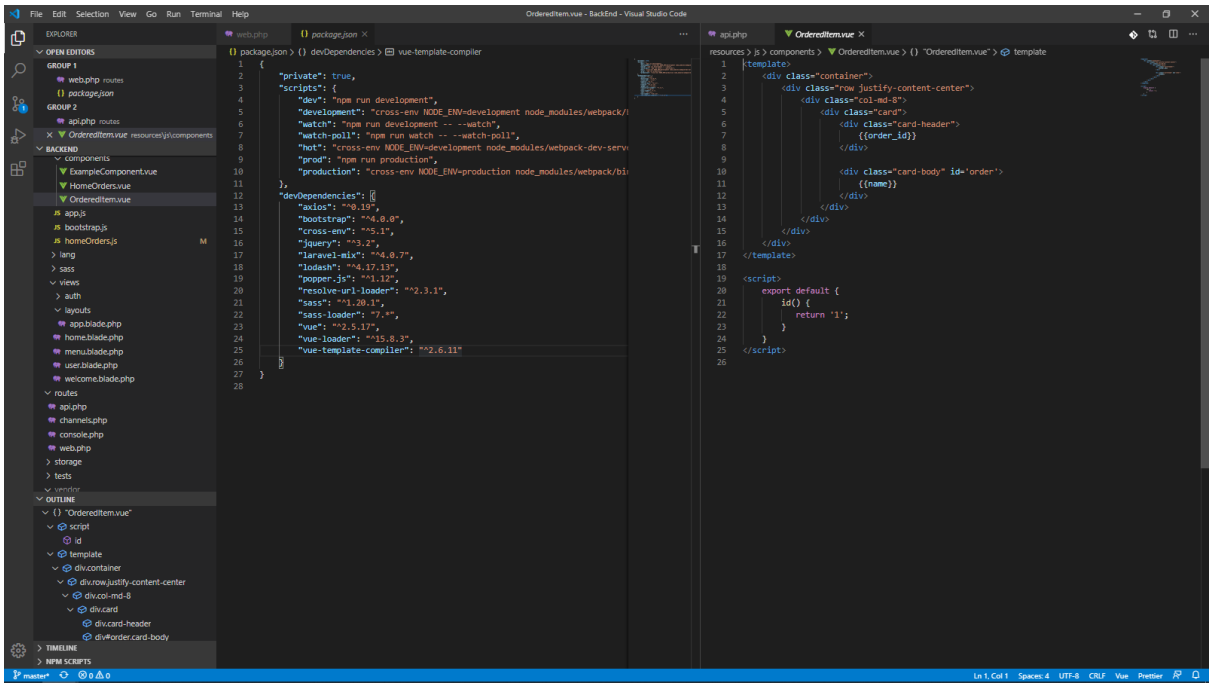


Figure 19: Implementation Screenshot (node packages used (left) and order VUE template(right))

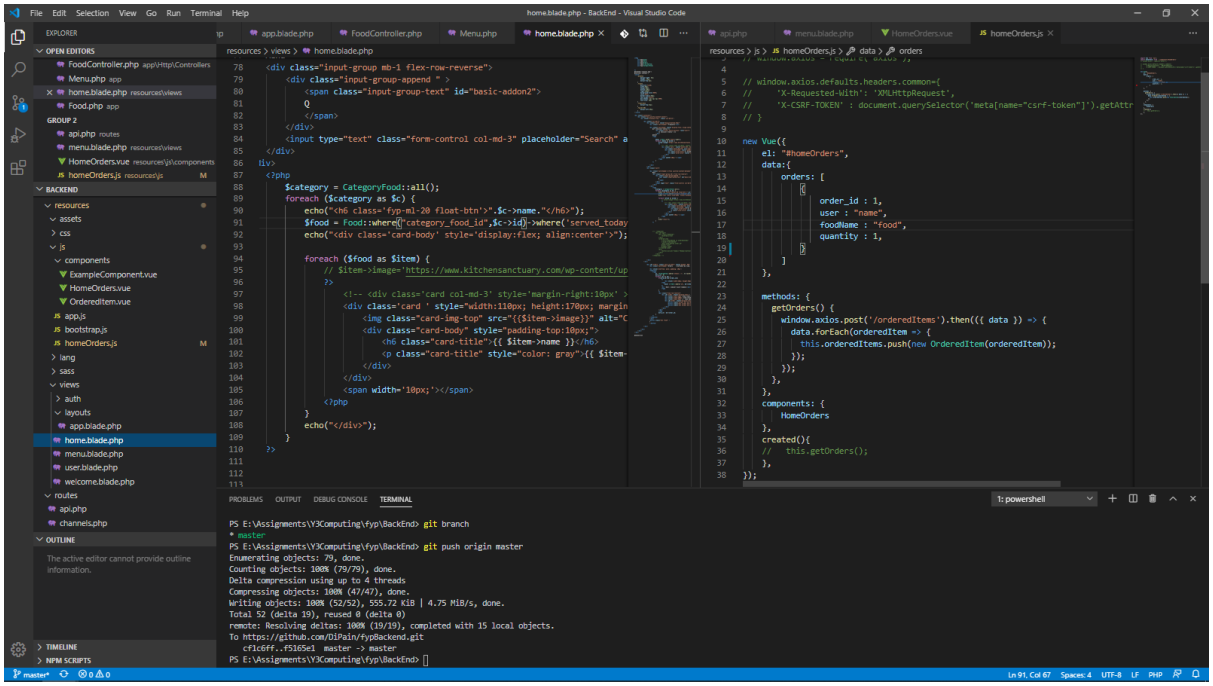


Figure 20: Implementation Screenshot (Home page(left) and orders logic(right))

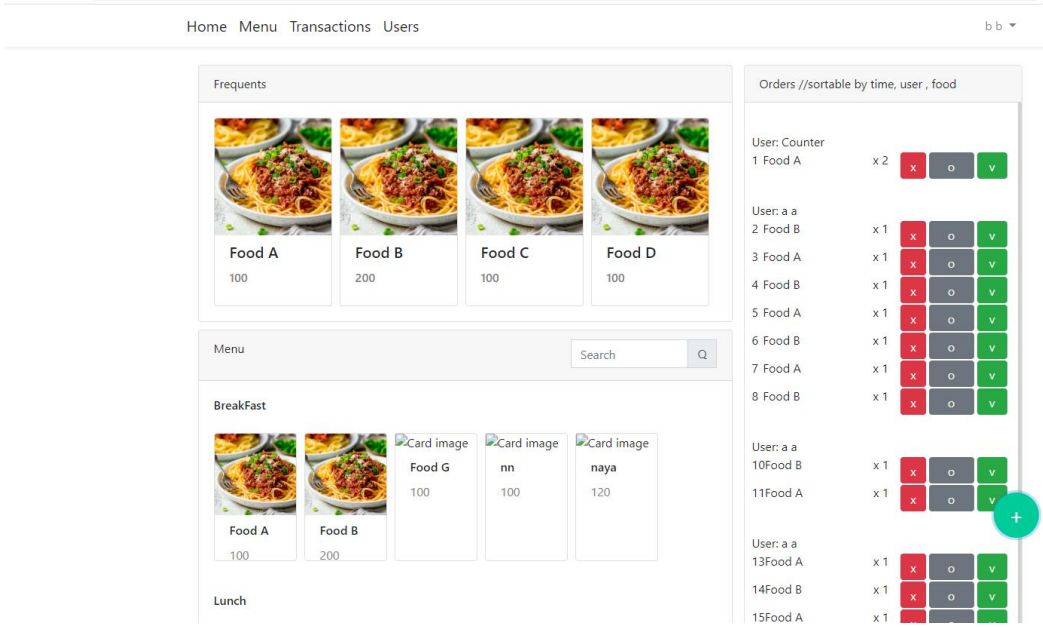


Figure 21: Implementation Screenshot (home page)

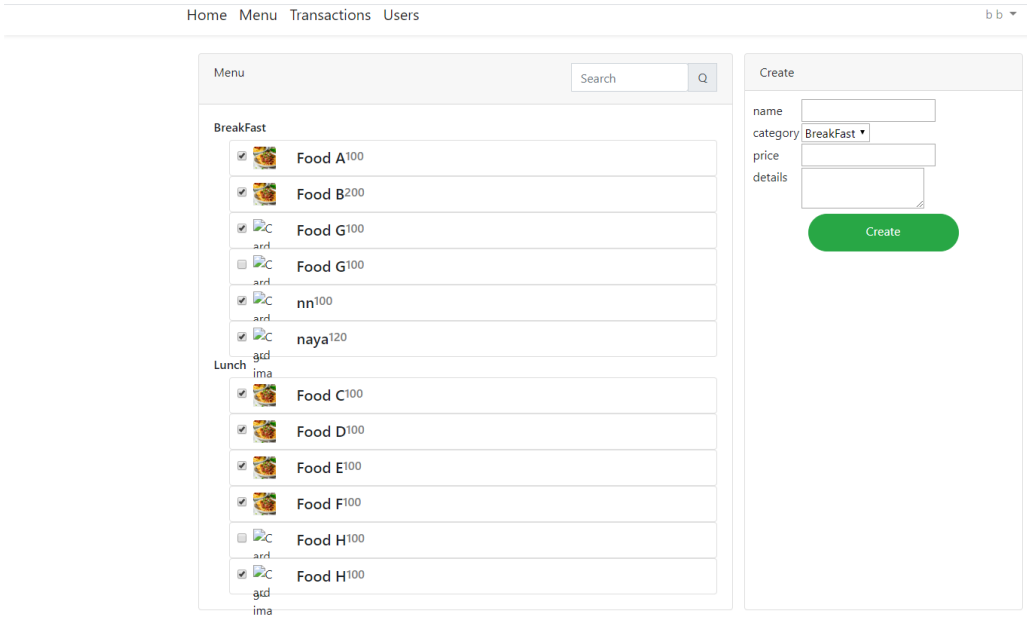


Figure 22: Implementation Screenshot (Menu)

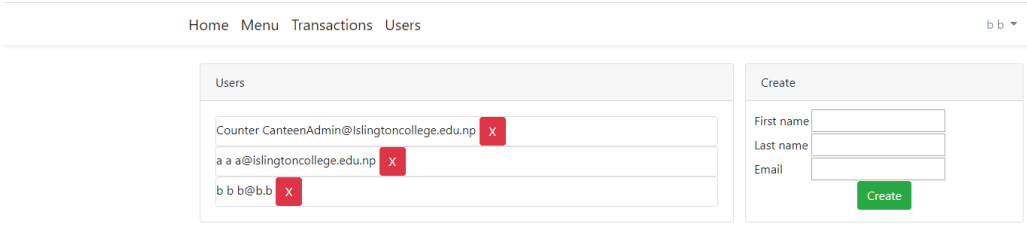


Figure 23: Implementation Screenshot – Users

3.6.3. THIRD (FINAL) ITERATION

3.6.3.1 ANALYSIS

The third iteration had to started had to be started 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

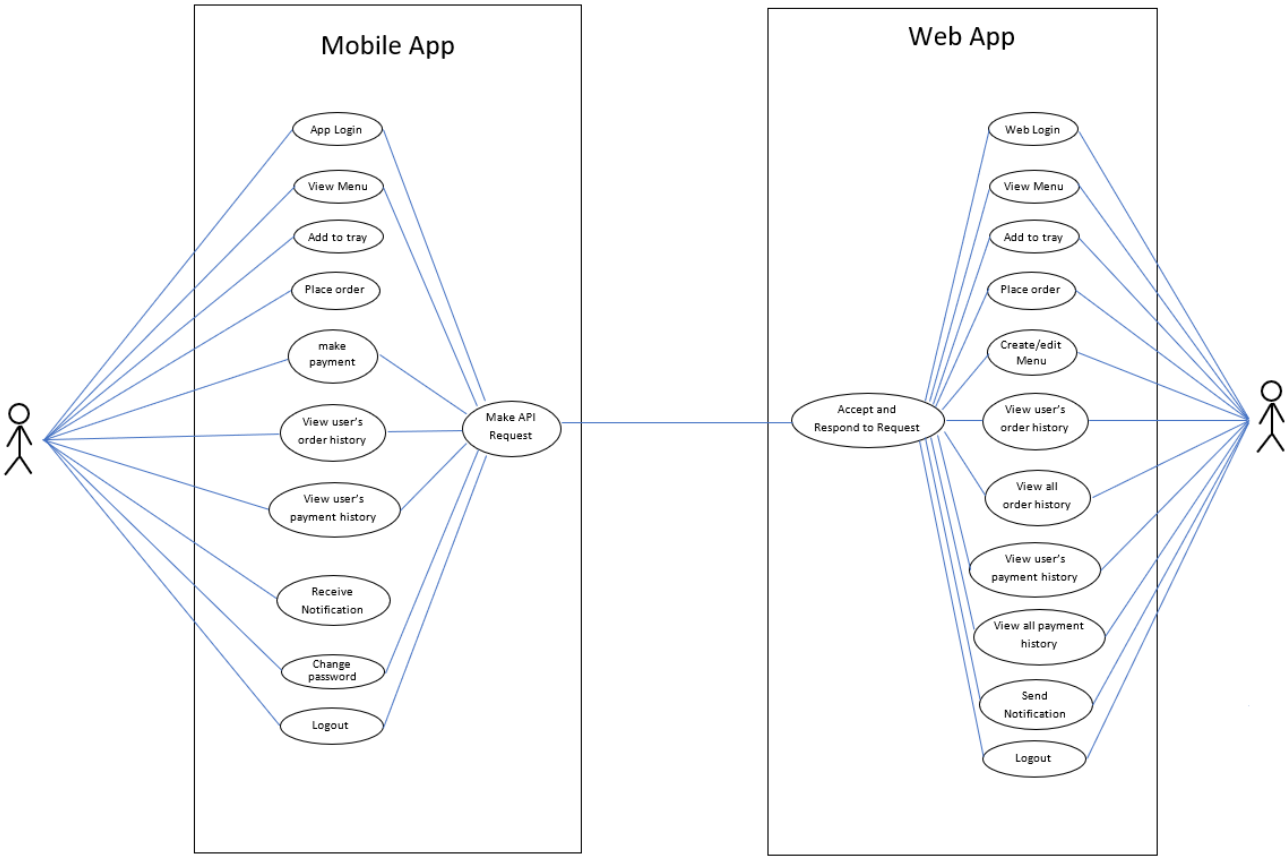


Figure 24: Use Case – Third Iteration

Data Dictionary

Users

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	name	VARCHAR	191	☐	No default
3	email	VARCHAR	191	☐	No default
4	api_token	VARCHAR	191	☑	NULL
5	password	VARCHAR	191	☐	No default
6	role_id	VARCHAR	191	☐	'1'
7	balance	VARCHAR	191	☐	'0'
8	created_at	TIMESTAMP		☑	NULL
9	updated_at	TIMESTAMP		☑	NULL

Figure 25: Data Table - Users

Roles

#	Name	Datatype	Length/Set	Allow NULL	Default
1	id	INT	11	☑	NULL
2	title	VARCHAR	50	☑	NULL

Figure 26: Data Table - Roles

Foods

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	category_food_id	INT	11	☐	No default
3	name	VARCHAR	191	☐	No default
4	served_today	INT	1	☑	0
5	details	VARCHAR	191	☐	No default
6	image	VARCHAR	150	☑	NULL
7	price	INT	11	☐	No default
8	created_at	TIMESTAMP		☑	NULL
9	updated_at	TIMESTAMP		☑	NULL

Figure 27: Data Table - Foods

Category Food

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	name	VARCHAR	191	☐	No default
3	created_at	TIMESTAMP		☑	NULL
4	updated_at	TIMESTAMP		☑	NULL

Figure 28: Data Table – Category Food

Orders

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	user_id	INT	11	☐	No default
3	for	DATETIME		☐	No default
4	created_at	TIMESTAMP		☑	NULL
5	updated_at	TIMESTAMP		☑	NULL

Figure 29: Data Table - Orders

Ordered_items

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	order_id	INT	11	☐	No default
3	food_id	INT	11	☐	No default
4	quantity	INT	11	☐	No default
5	price	INT	11	☑	NULL
6	status	INT	2	☑	0
7	created_at	TIMESTAMP		☑	NULL
8	updated_at	TIMESTAMP		☑	NULL

Figure 30: Data Table – Ordered Item

Payments

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	user_id	INT	11	☐	No default
3	pay_method_id	INT	11	☐	No default
4	order_id	INT	11	☑	NULL
5	amount	INT	11	☑	0
6	description	VARCHAR	222	☐	''
7	created_at	TIMESTAMP		☑	NULL
8	updated_at	TIMESTAMP		☑	NULL

Figure 31: Data Table - Payments

Pay Methods

#	Name	Datatype	Length/Set	Allow NULL	Default
1	**id**	**BIGINT**	**20**	☐	**AUTO_INCREME...**
2	name	VARCHAR	191	☐	No default
3	particular	VARCHAR	191	☐	No default
4	created_at	TIMESTAMP		☑	NULL
5	updated_at	TIMESTAMP		☑	NULL

Figure 32: Data Table – Pay methods

3.6.4.3. IMPLEMENATION

MOBILE APP

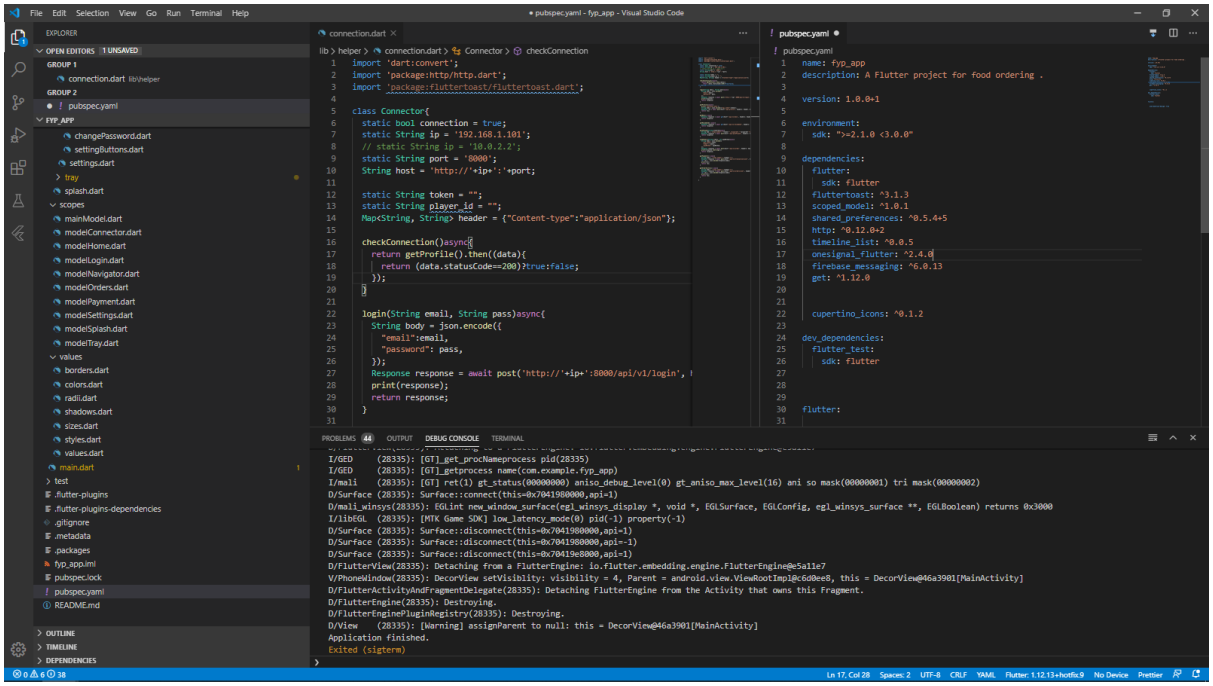


Figure 33: Implementation Screenshot (API Connector (left) and flutter packages user(right))

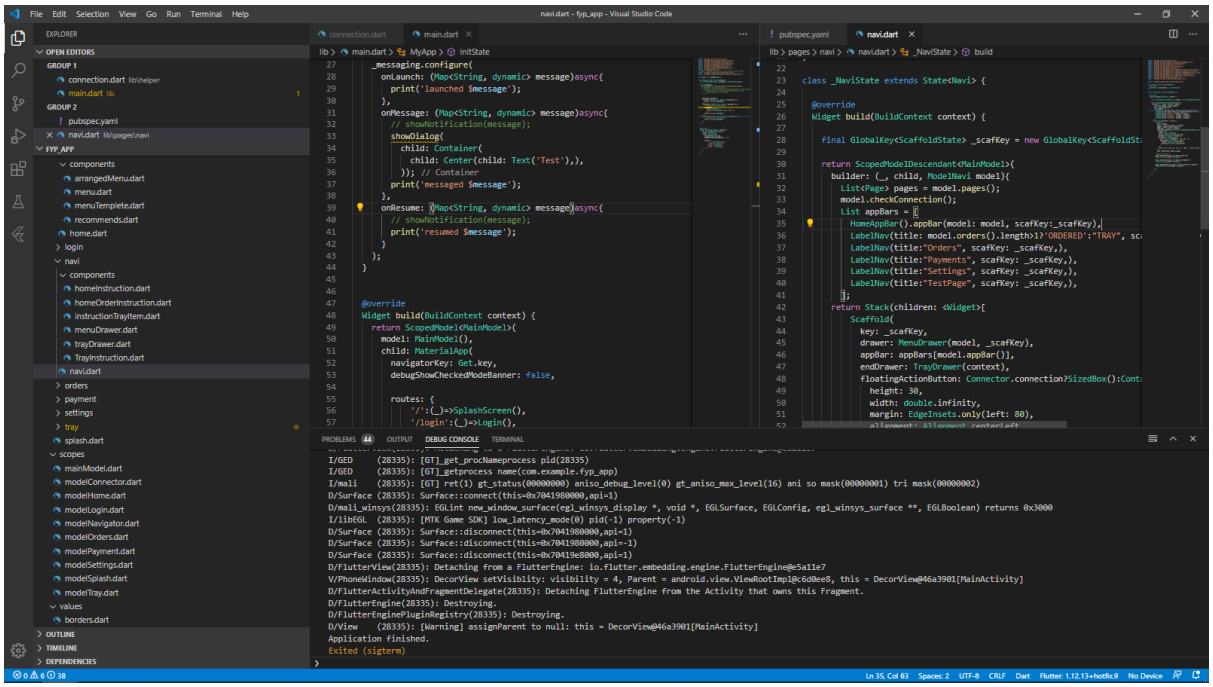


Figure 34: Implementation Screenshot (main.dart (left) and navi.dart (right))

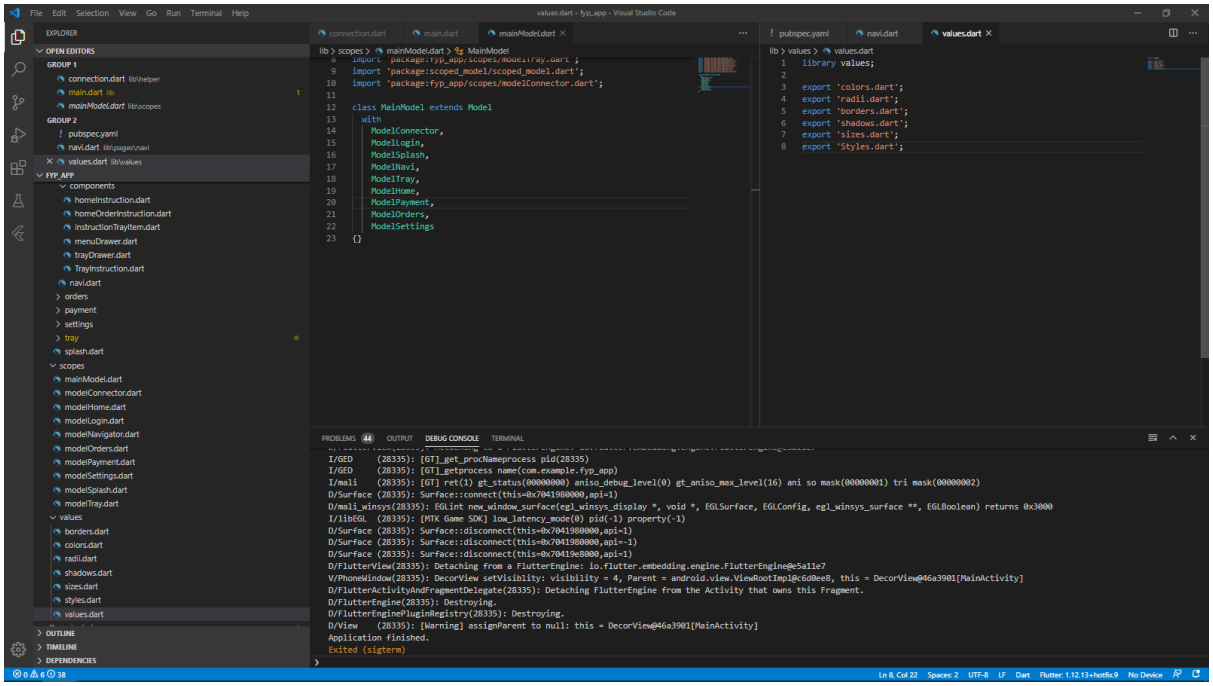


Figure 35: Implementation Screenshot (Scoped models (left) and custom libraries(right))

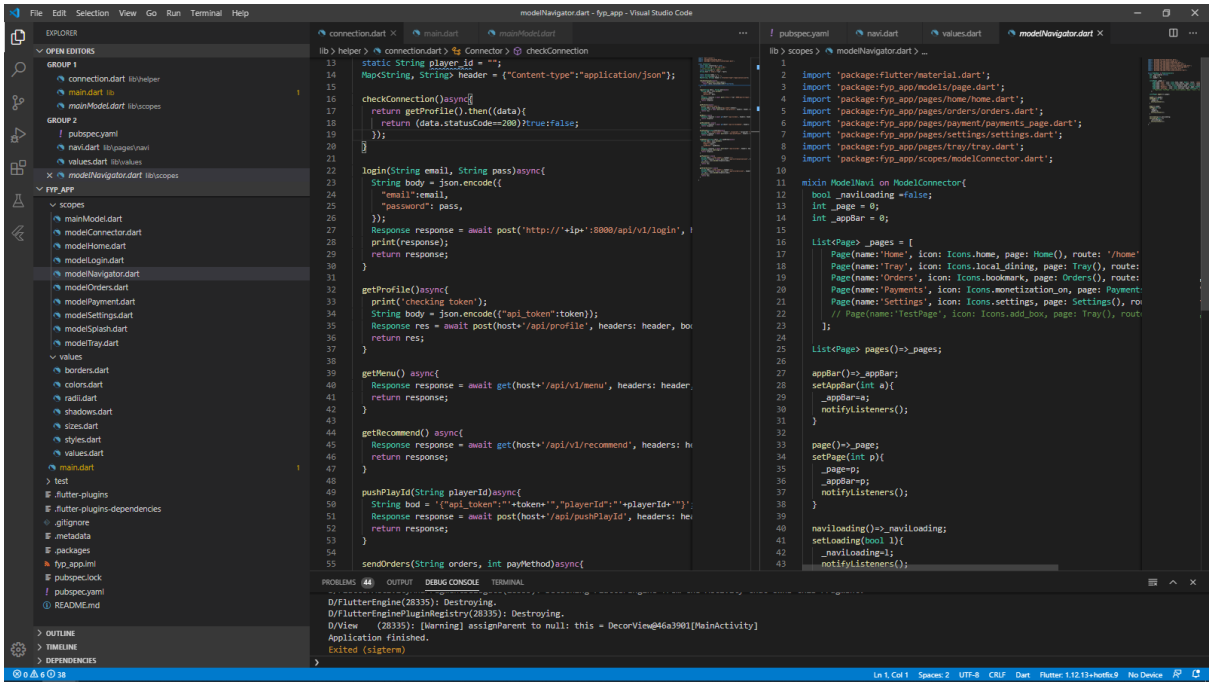


Figure 36: Implementation Screenshot (API Request modules (left) and navigation logic(right))

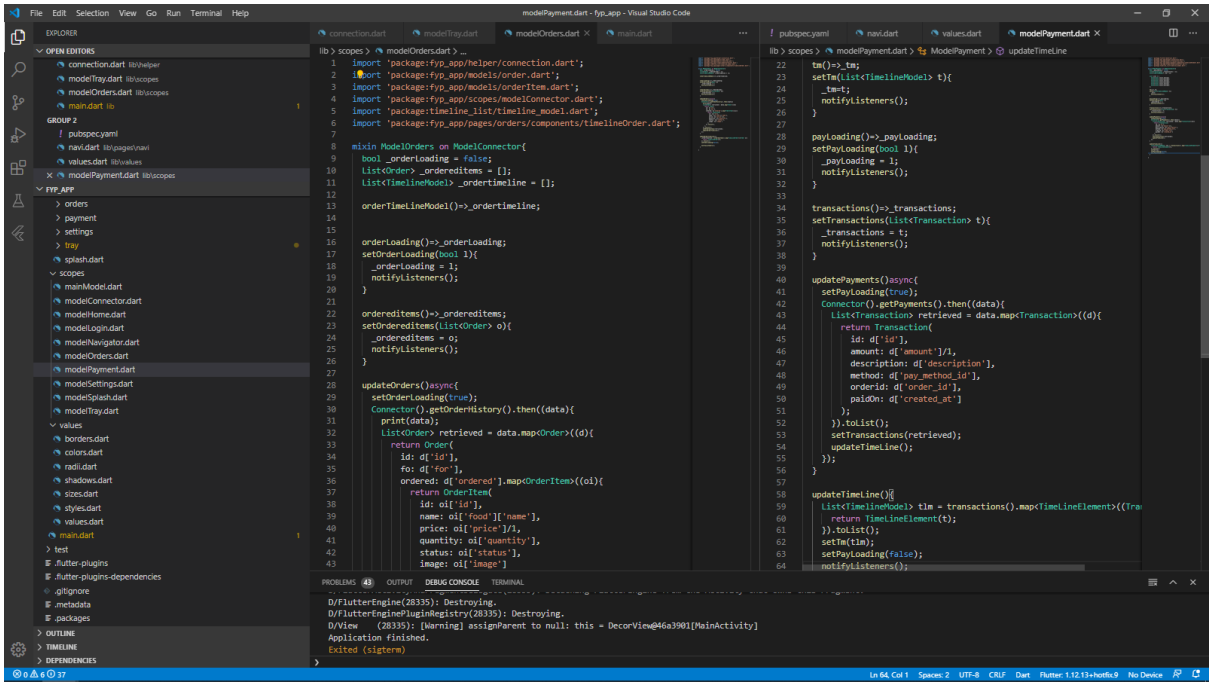


Figure 37: Implementation Screenshot (Orders Logic(left) and Payment logic(right))

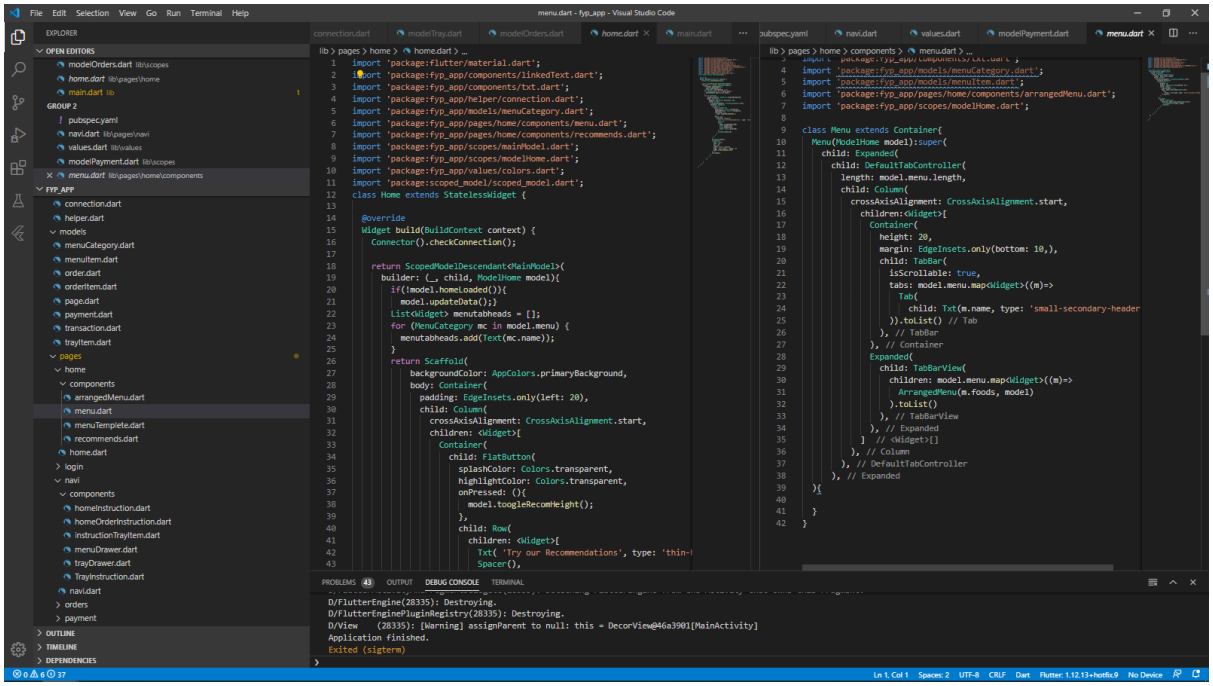


Figure 38: Implementation Screenshot (home widget(left) and menu component(right))

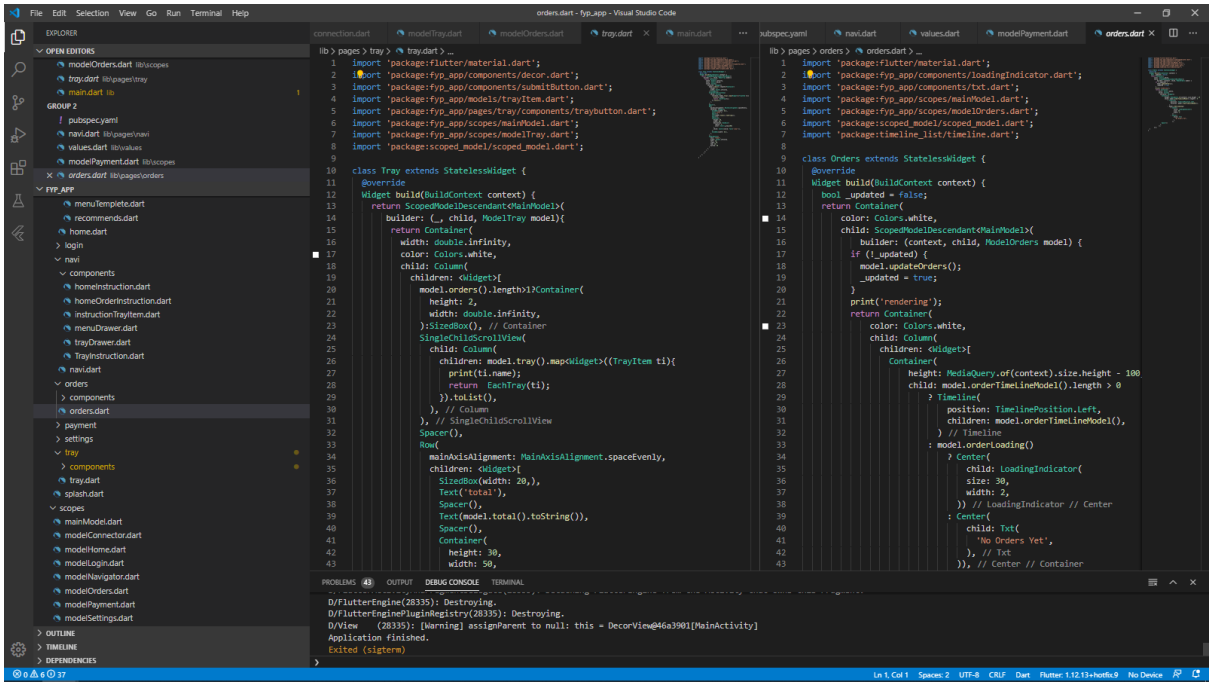


Figure 39: Implementation Screenshot (tray widget (left) and orders widget(right))

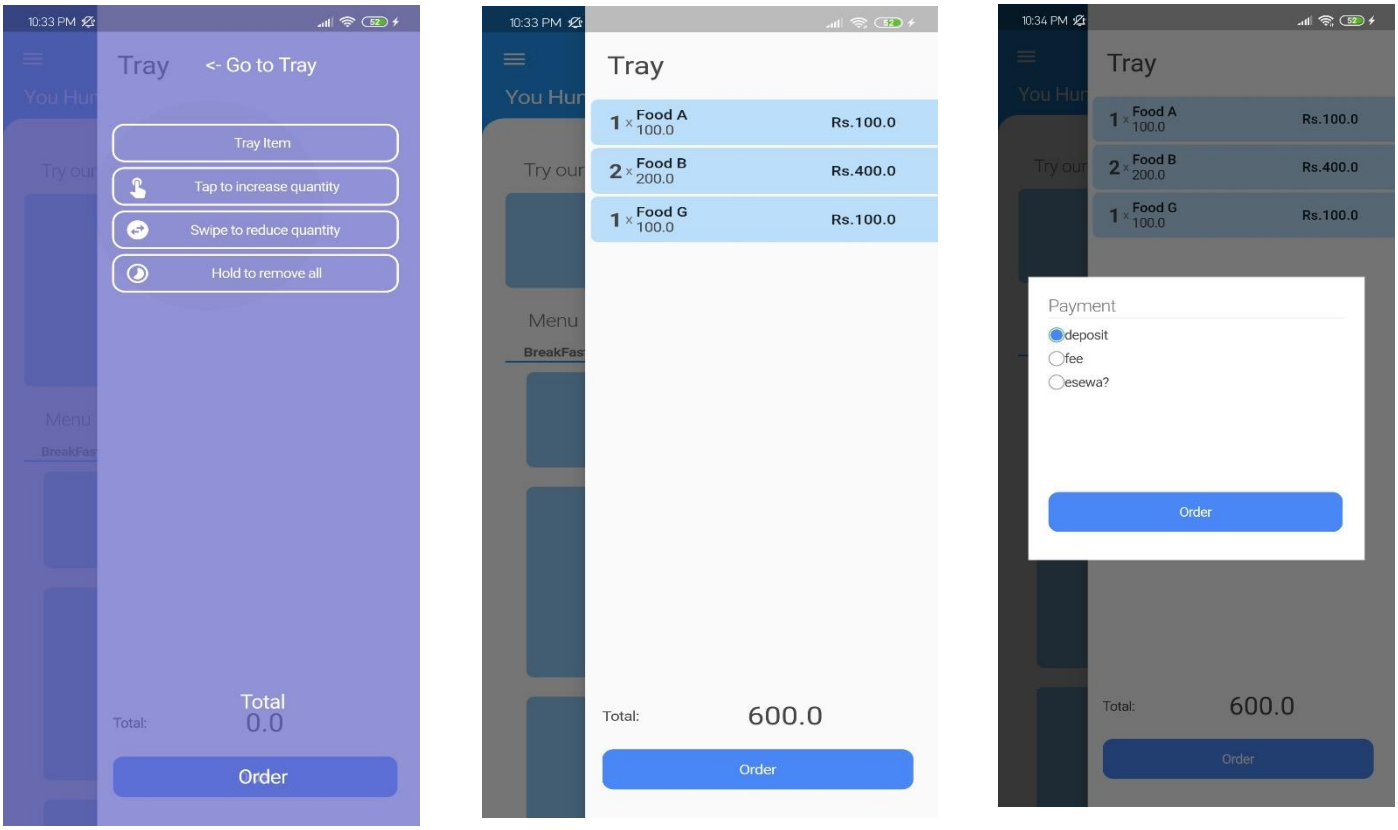


Figure 40: Implementation Screenshot (from left: tray instruction, tray drawer, payment before order)

Web

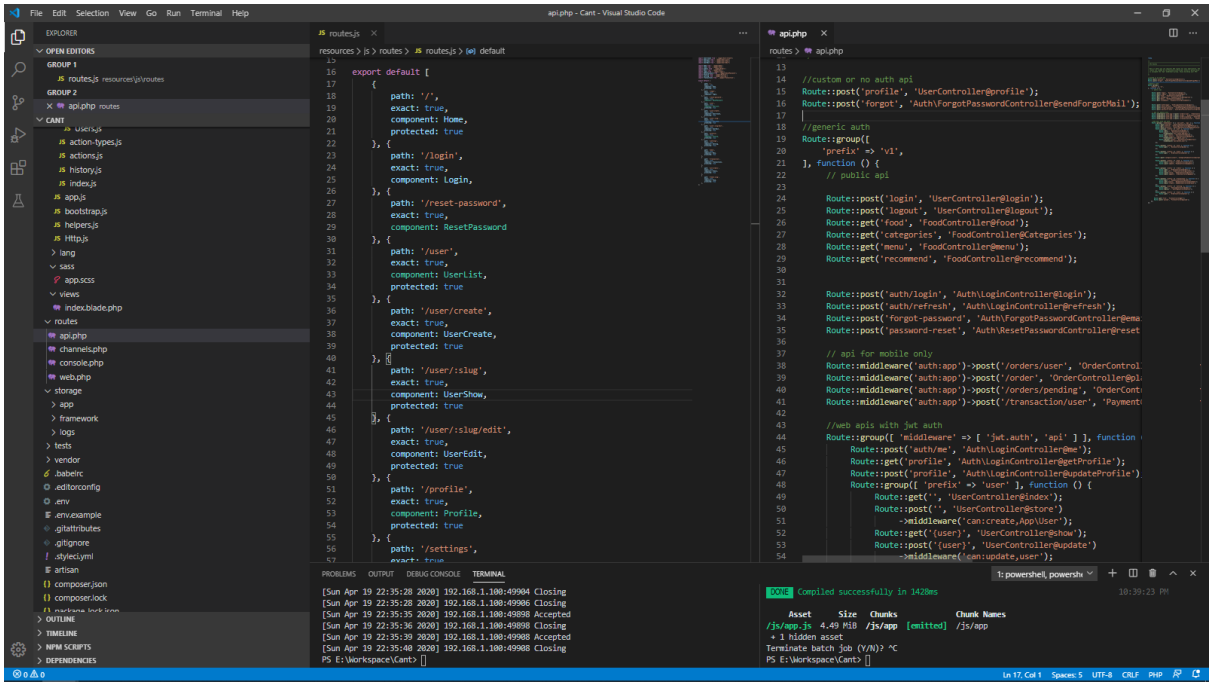


Figure 41: Implementation Screenshot (React routes (left) and API Routes(right))

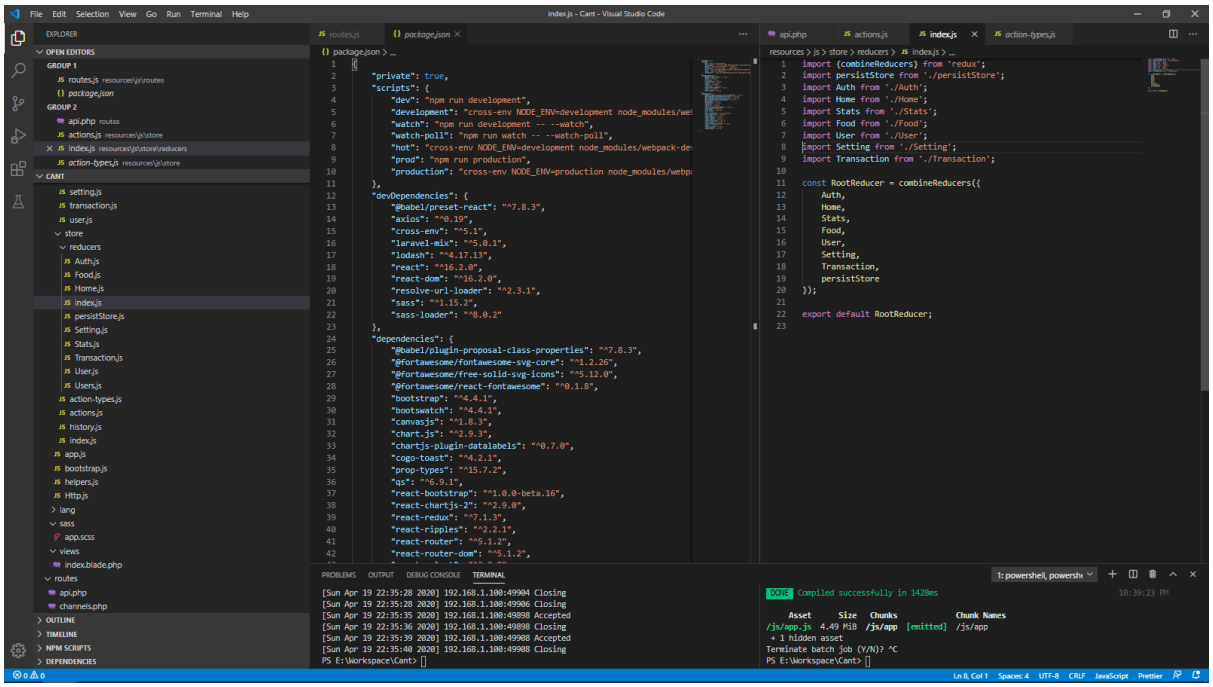


Figure 42: Implementation Screenshot (NPM packages (left) and Reducers created for REDUX (right))

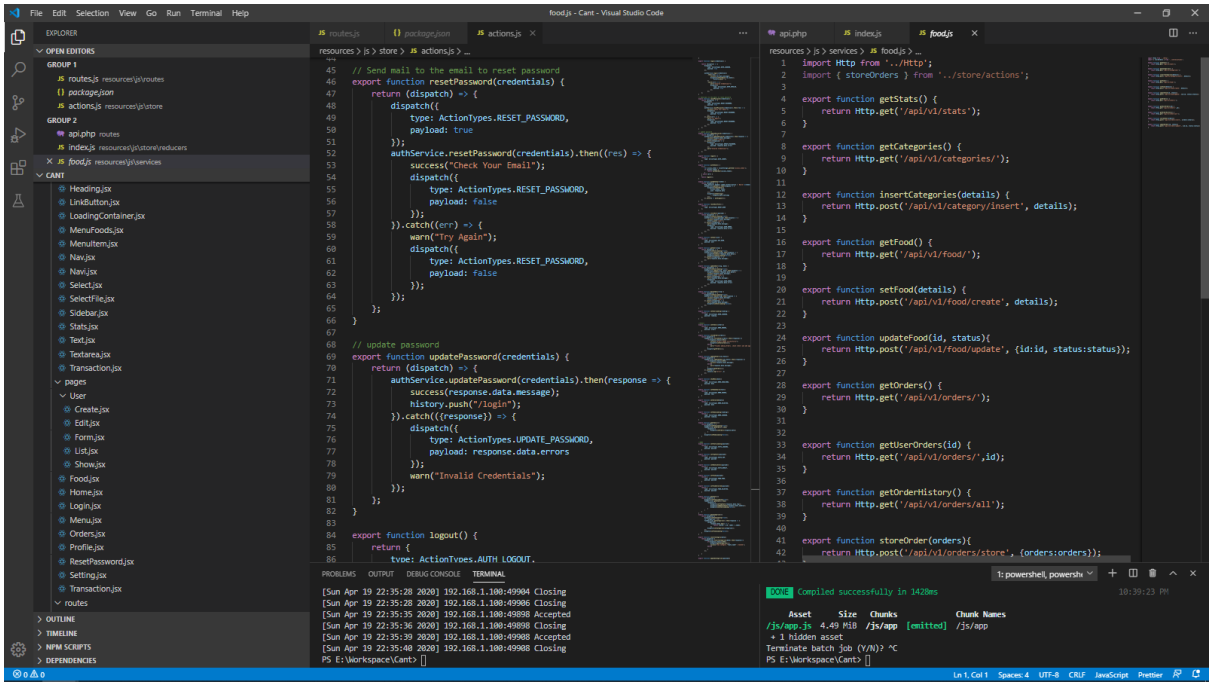


Figure 43: Implementation Screenshot (actions (left) and API requesting services(right))

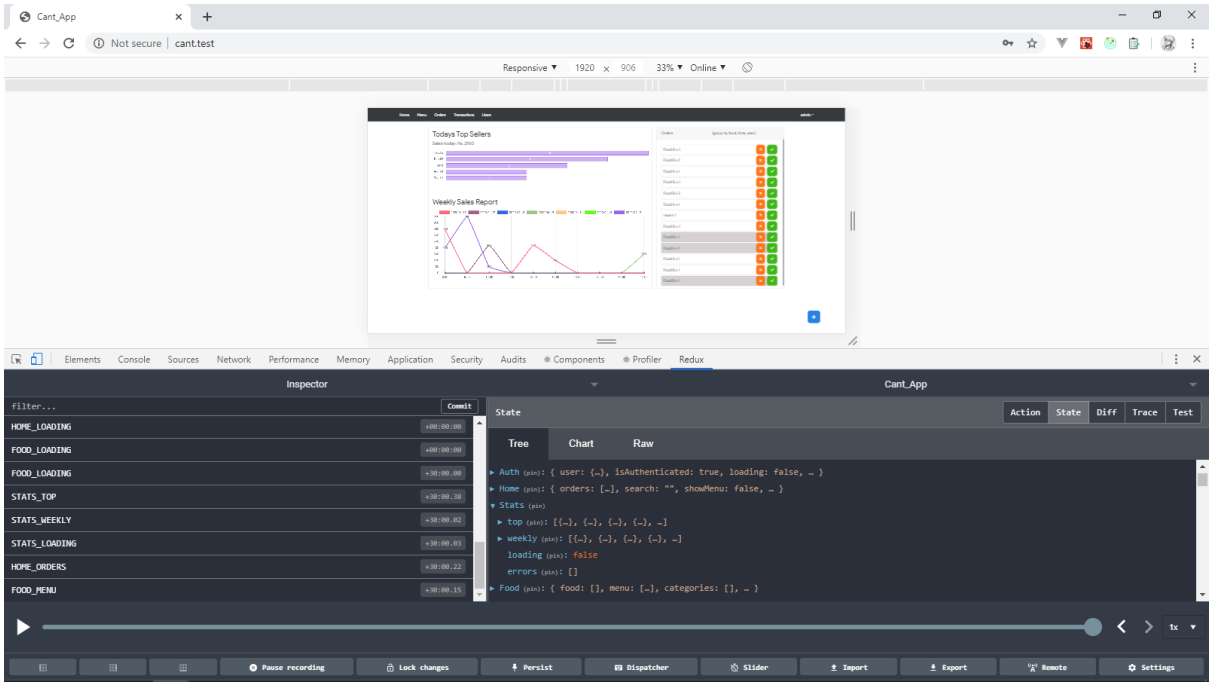


Figure 44: Implementation Screenshot – Home Page statistics with REDUX states

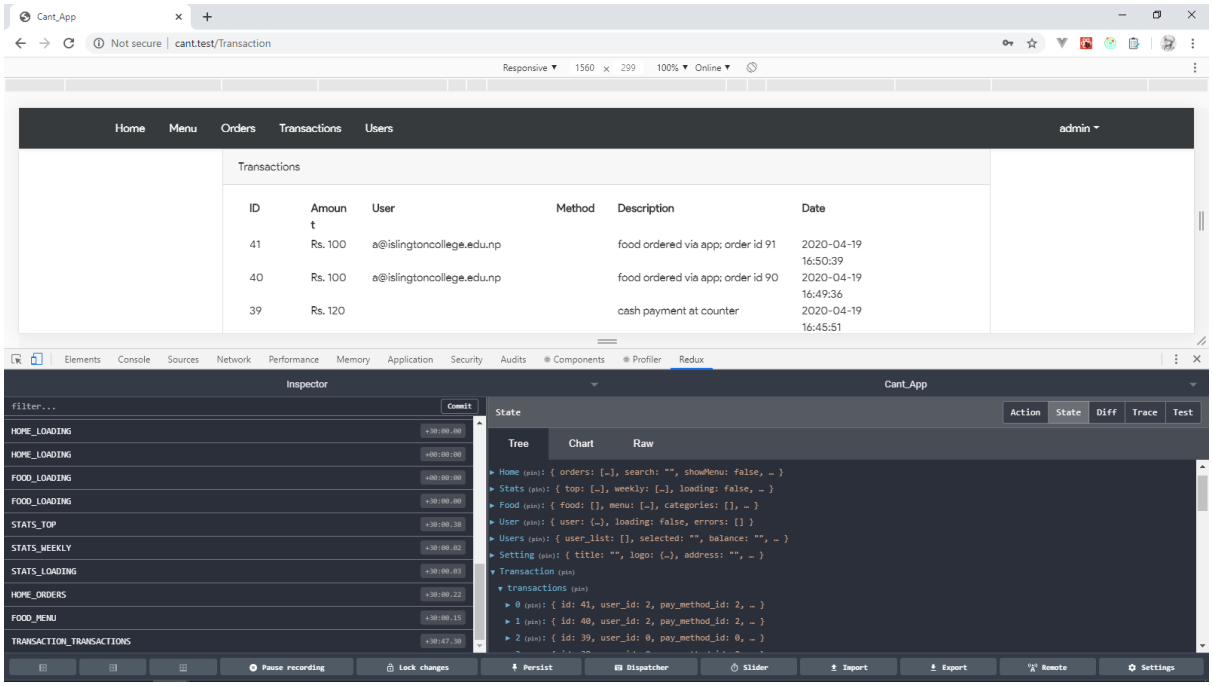


Figure 45: Implementation Screenshot – Transactions with REDUX states

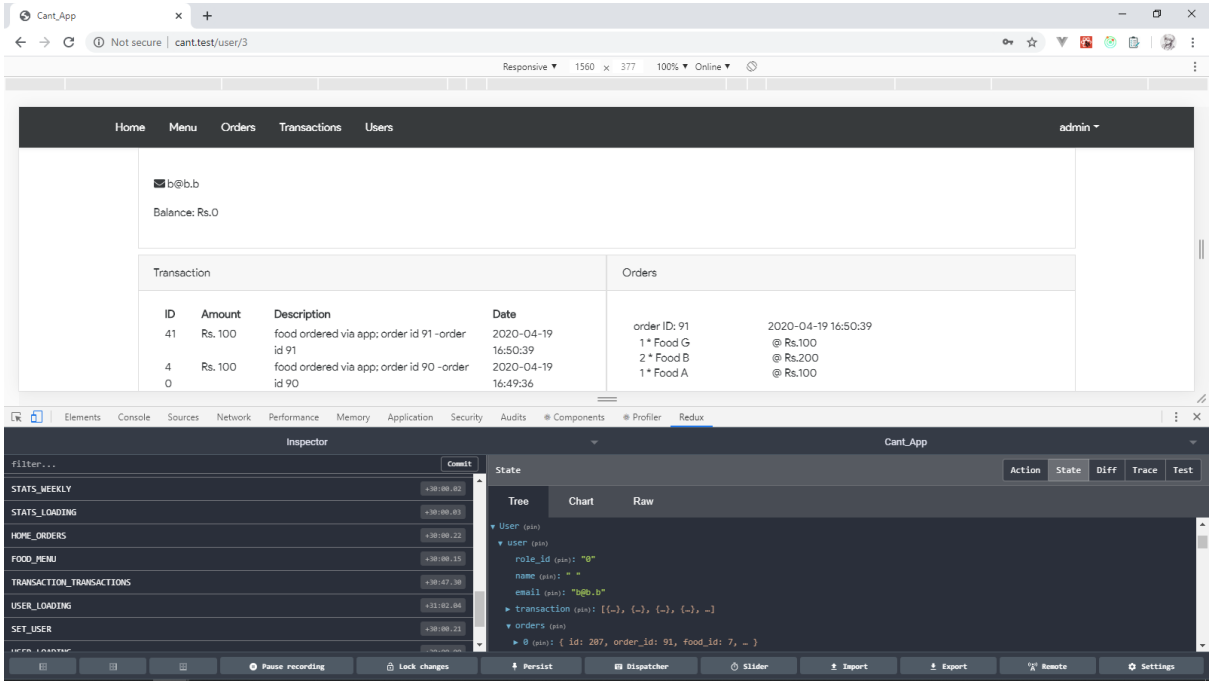


Figure 46: Implementation Screenshot – User Detail with REDUX states

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 bested 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 to 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

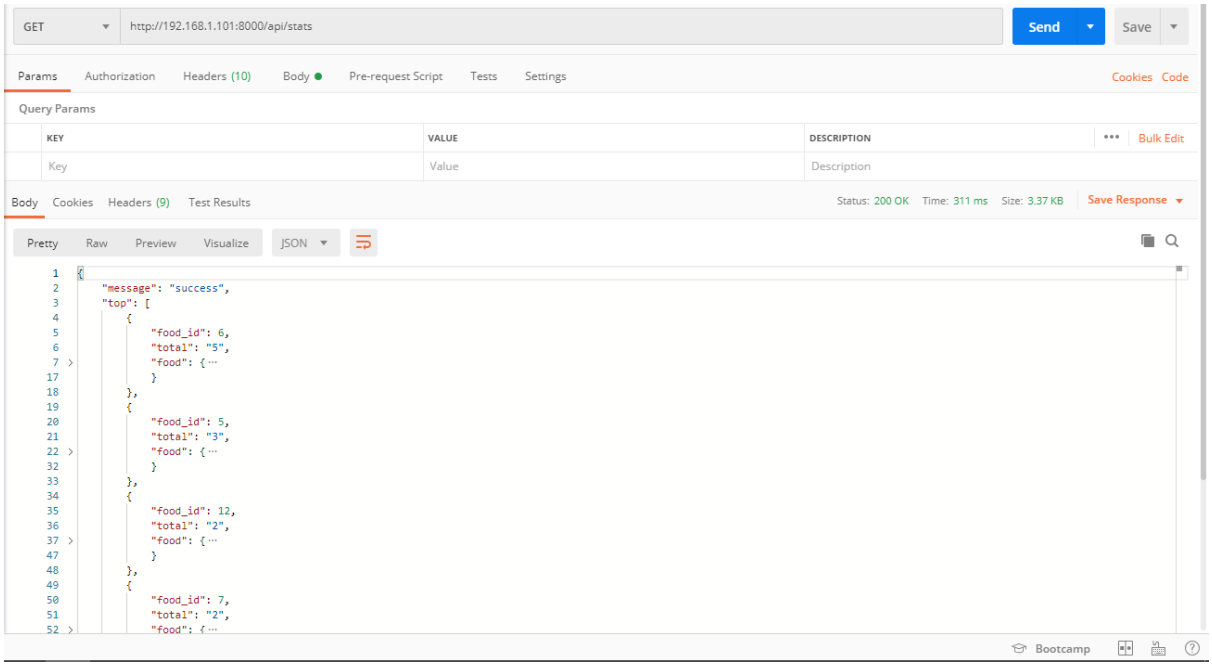


Figure 47: Test – Statistics API test

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

```
subprojects {
        project.configurations.all {
            resolutionStrategy.eachDependency { details ->
                if (details.requested.group == 'com.android.support'
                        && !details.requested.name.contains('multidex') ) {
                    details.useVersion "27.1.1"
                }
                if (details.requested.group == 'androidx.core'
                        && !details.requested.name.contains('androidx') ) {
                    details.useVersion "1.0.1"
                }
            }
        }
    }
```

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
```

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

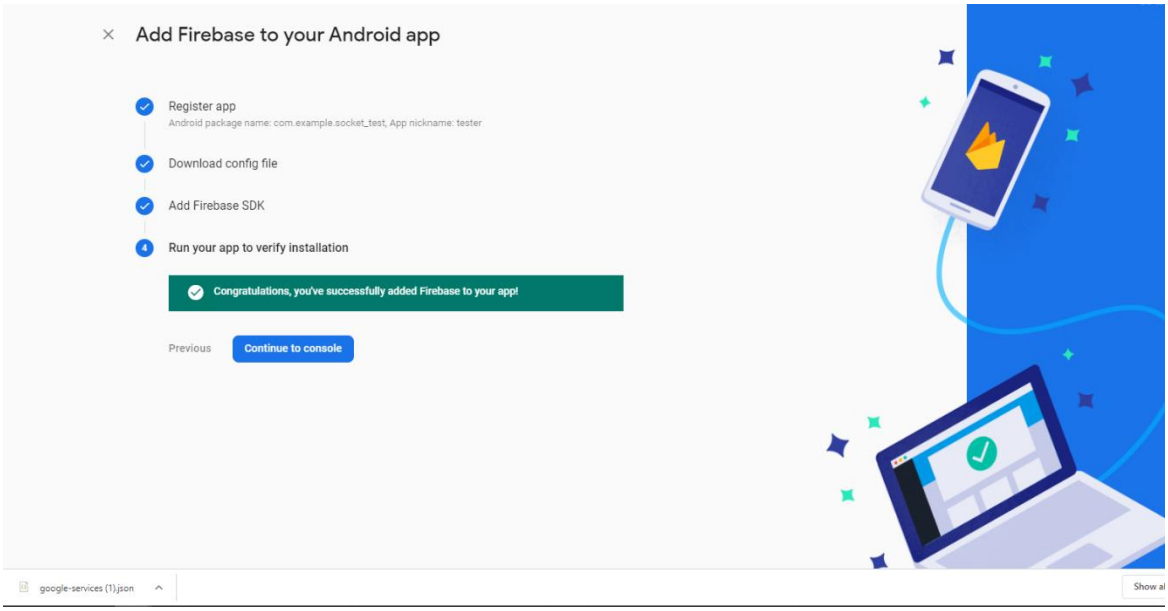


Figure 52: Test – FCM implementation

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

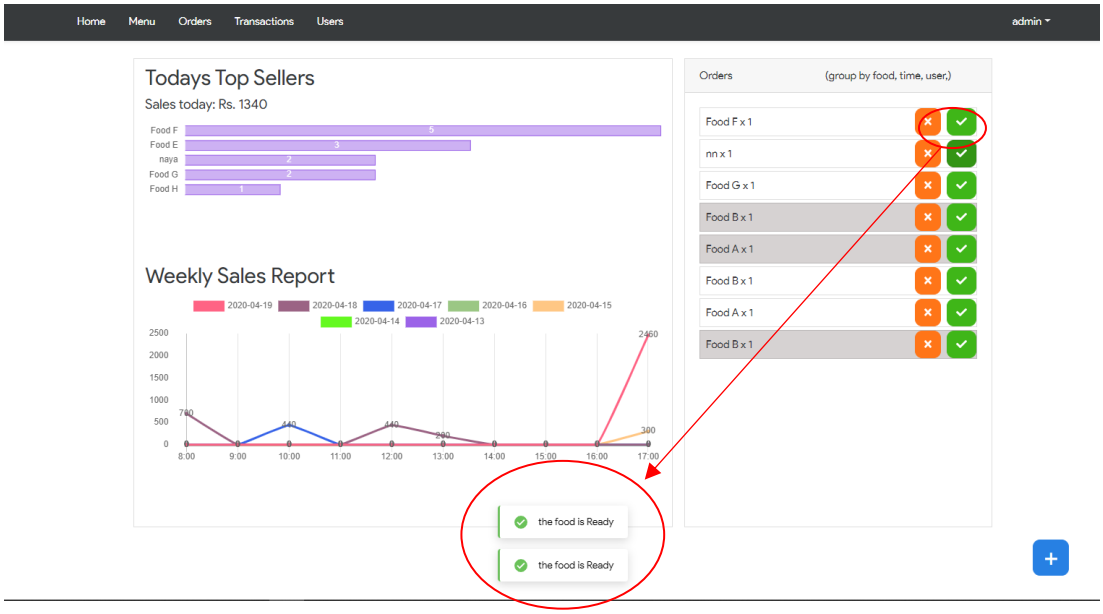


Figure 53: Test – changing order status

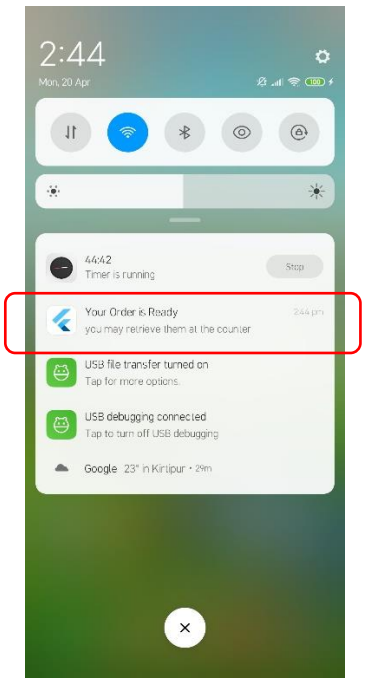


Figure 54: Test - mobile notification

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

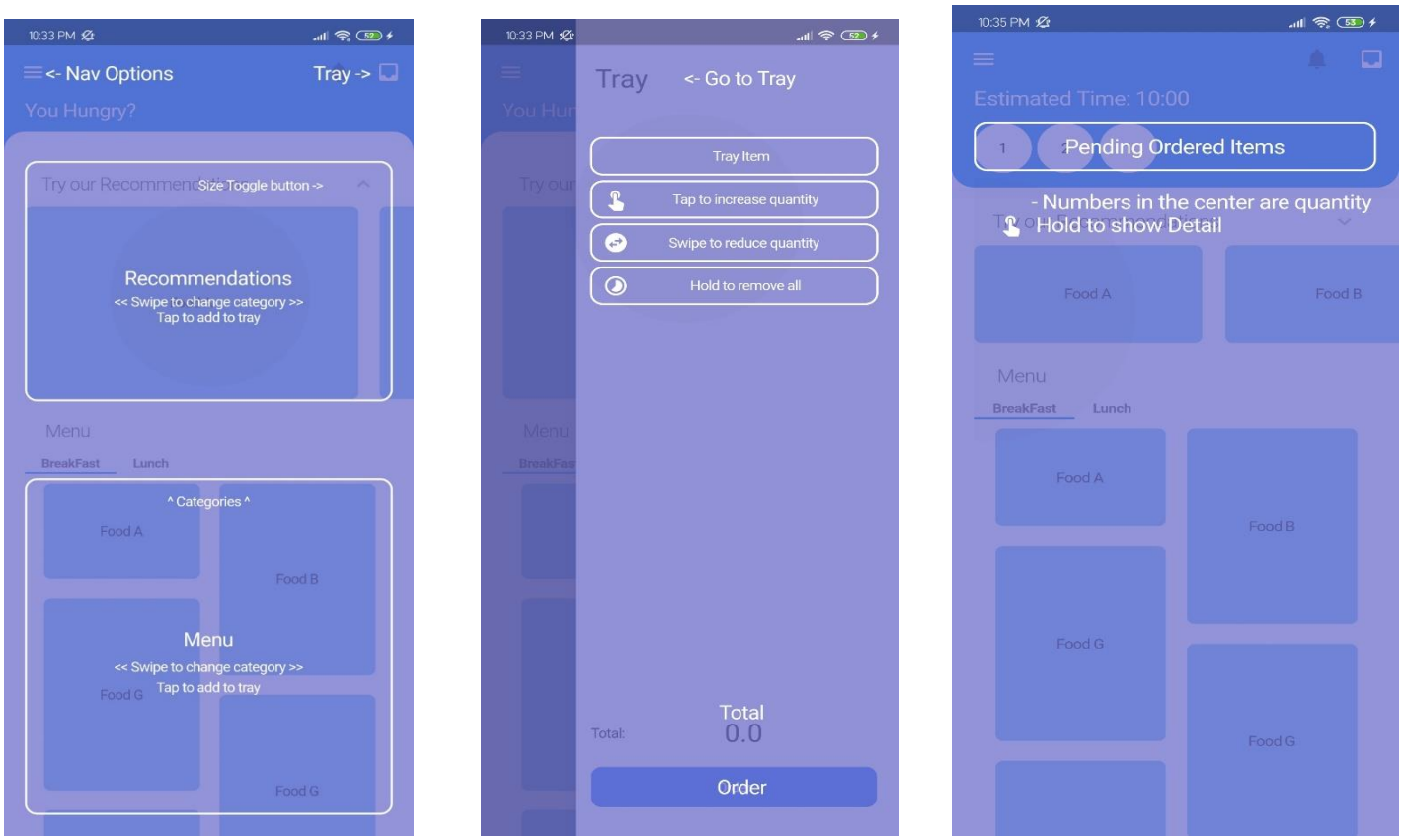


Figure 55: Test - mobile Instruction Overlay

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

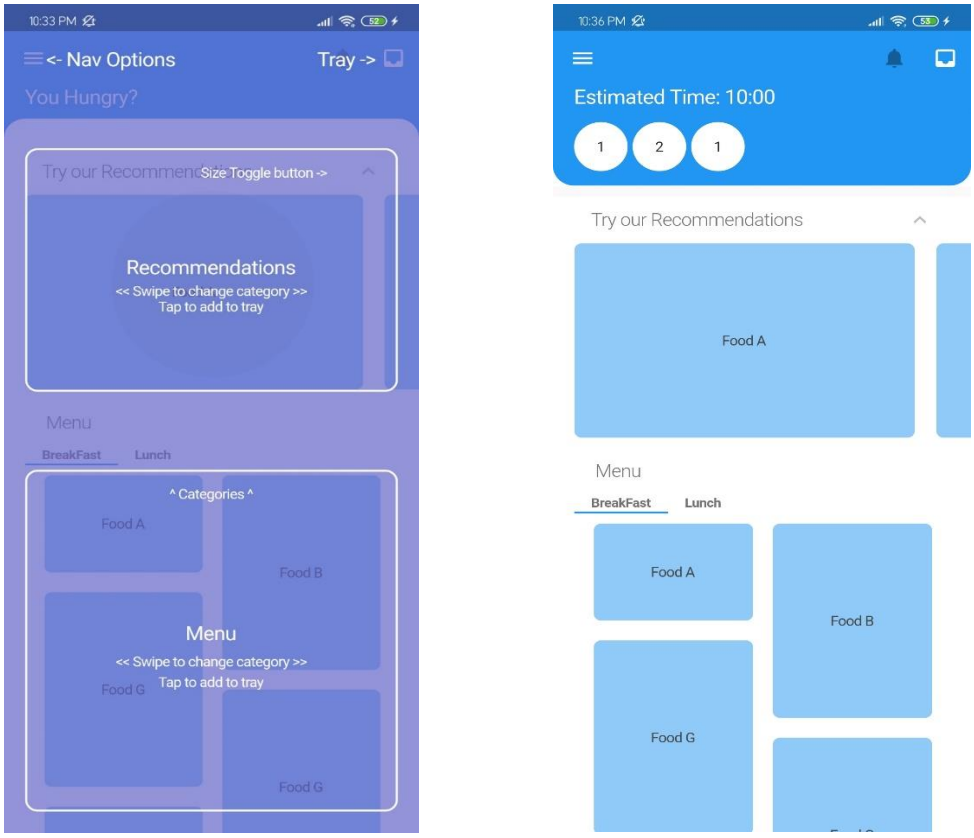


Figure 56: Test – Tutorial repetition

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

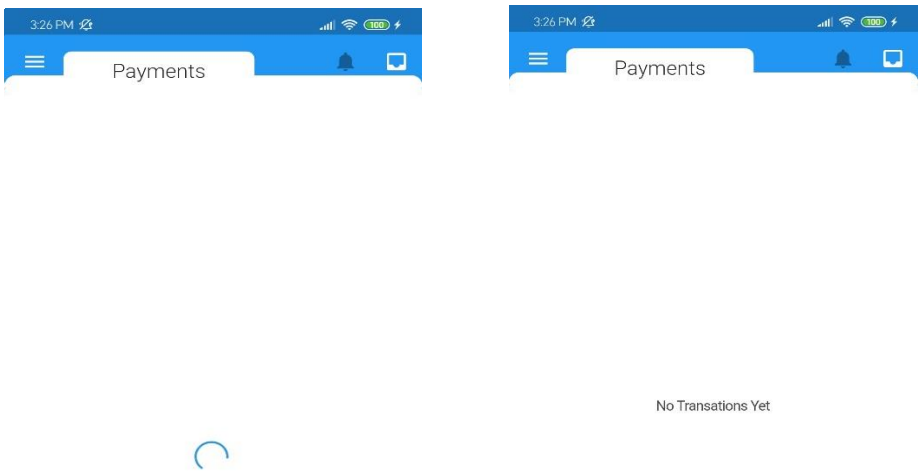


Figure 57: Test - disconnection

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

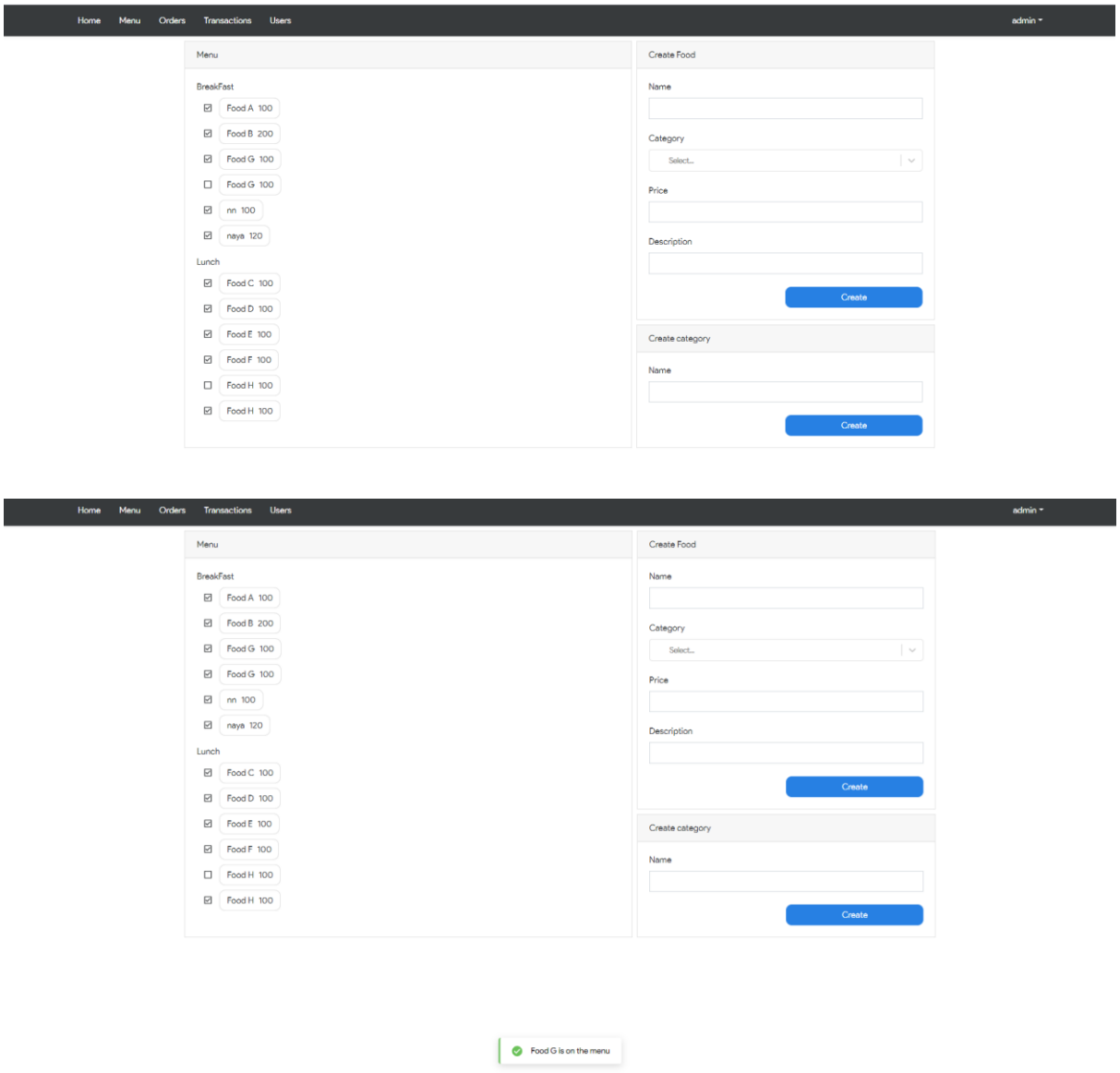


Figure 58: Test – Menu Update

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 made 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]
Available at: http://mealnepal.com/online-food-delivery-in-nepal-top-5-services/
[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-chartjs-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].

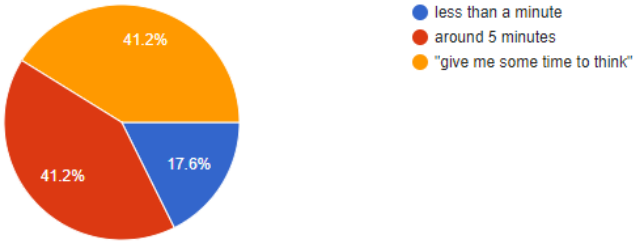
w3school, 2019. *AJAX Introduction | w3school.* [Online]
Available at: https://www.w3schools.com/xml/ajax_intro.asp
[Accessed 27 12 2019].

CHAPTER 8: APPENDIX

8.1 APPENDIX A: SURVEY

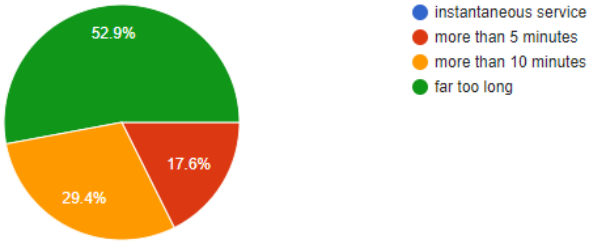
How long do you usually spend choosing meals in-front of the counter?

17 responses



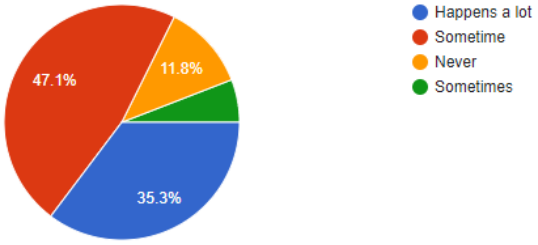
How long do you usually wait for meals once ordered in the college cafeteria?

17 responses



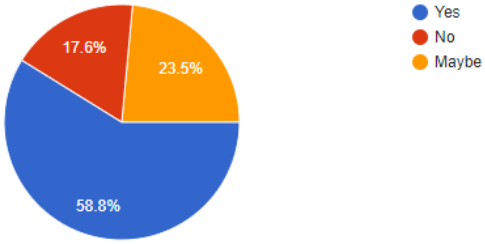
Have you had to move elsewhere because the cafeteria was too crowded?

17 responses



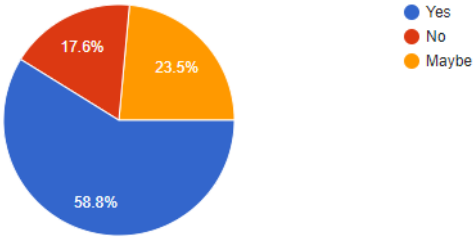
Do you have trouble deciding what to order often?

17 responses



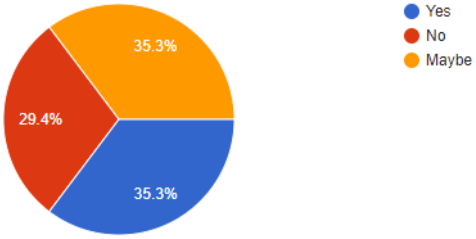
Does what other people order change your mind?

17 responses



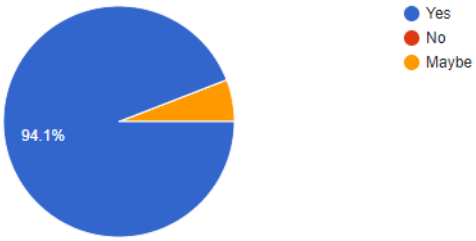
do you tend to avoid interaction with strangers?

17 responses



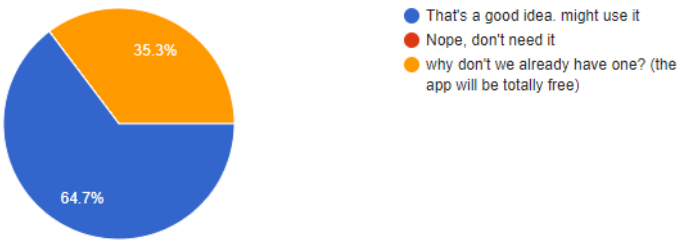
Would you want someone to place your orders and notify you when its ready?

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 authGetUser(access_token);
        }
    } catch (err) {
    }
    return logout();
}

export function authGetUser(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. chech 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(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: <Widget>[
                  Container(
                    child: FlatButton(
                      splashColor: Colors.transparent,
                      highlightColor: Colors.transparent,
                      onPressed: (){
                        model.toogleRecomHeight();
                      },
                      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

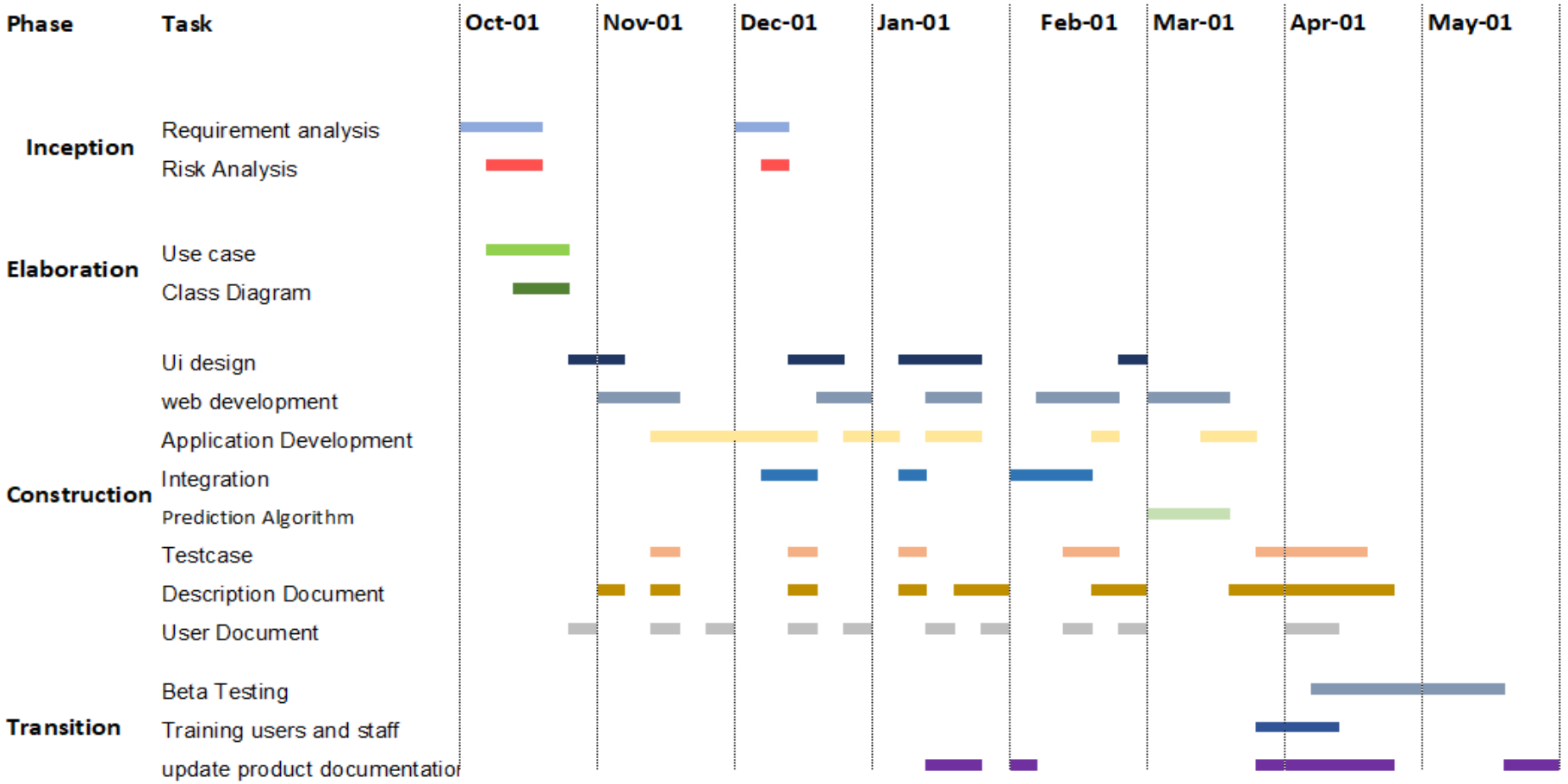
```
class Orders extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    bool _updated = false;
    return Container(
        color: Colors.white,
        child: ScopedModelDescendant<MainModel>(
            builder: (context, child, ModelOrders model) {
          if (!_updated) {
            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

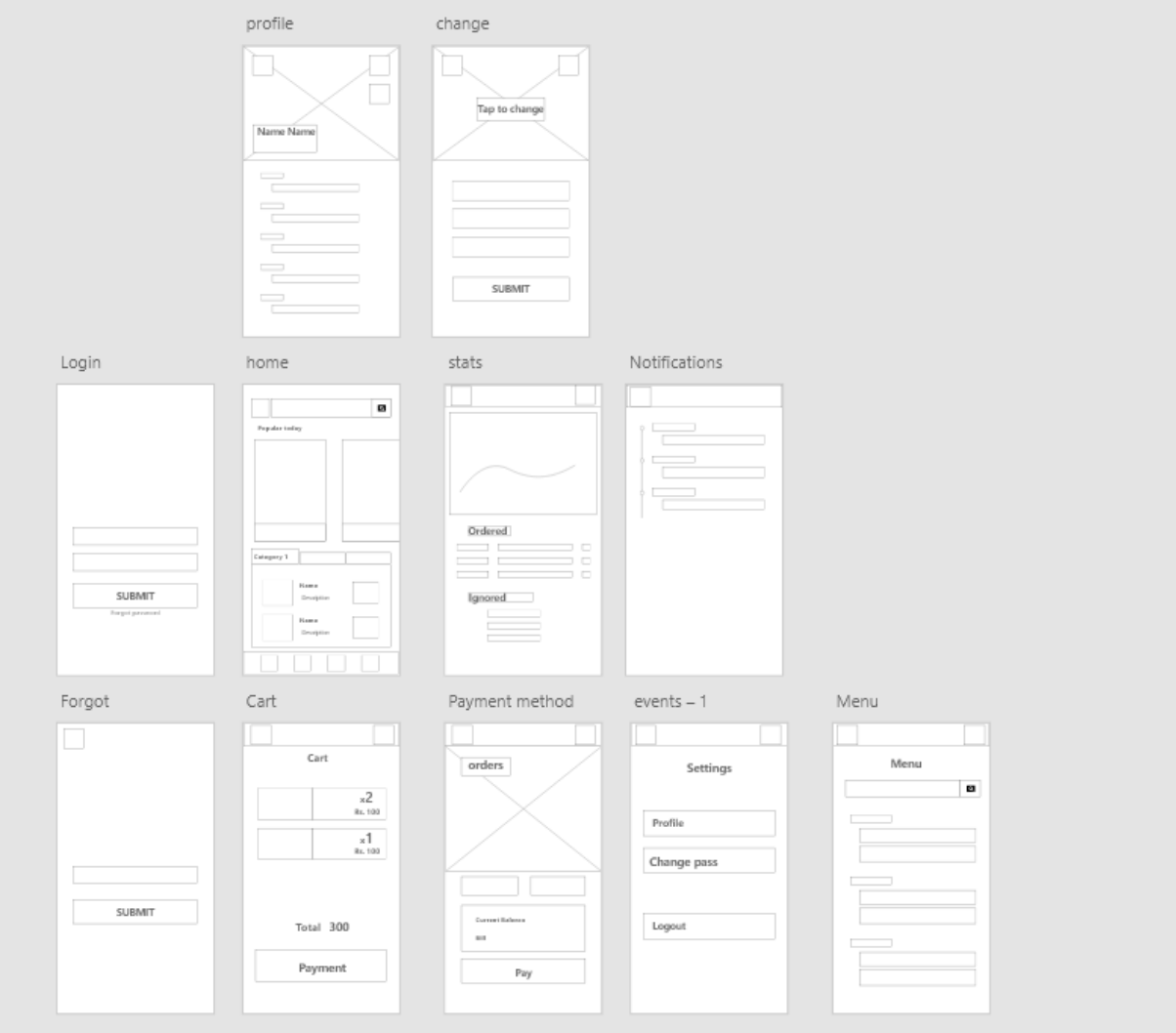
Name		Deepan Singh				
Project Name		Canteen app				
Student ID		id - 17031147				
Internal Supervisor		Subeksha Shrestha				
External Supervisor		Ishwor Shrestha				
Section	**Feature**	**Task**	**Completed**	**Completed On**		
				first iteration	**second Iteration**	**Third Iteration**
Web	**Login**	Create login api	☑	24/11/2019		5/3/2020
		Add form validation	☑	24/11/2019		5/3/2020
		implement jwt auth for future api	☑			5/3/2020
		Implement in front end	☑	24/11/2019		5/3/2020
	Navigation	create page routes	☑	9/12/2019		5/3/2020
		implement in front end	☑	9/12/2019		5/3/2020
	Statistics component	create api	☑			14/4/2020
		create algorithm	☑			14/4/2020
		implement in front end	☑			14/4/2020
	Show Menu component	create menu api	☑	10/12/2019		6/3/2020
		Implement in front end	☑	10/12/2019		6/3/2020
	Reccomendation system	create recommendation api	☑	11/12/2019		6/3/2020
		create recommendation algorithm	☑			14/4/2020
		implement in front end	☑			8/3/2020
	tray component	create backend function	☑			16/3/2020
		implement in front end	☑			16/3/2020
	Order function	create order api	☑	11/12/2019		6/3/2020
		implement in front end	☑	11/12/2019		7/3/2020
	User Orders List	create pending order api	☑	8/1/2020	22/1/2020	6/3/2020
		implement in front end	☑	8/1/2020	22/1/2020	9/3/2020
	Order Response funciton	create respond to order api	☑			6/3/2020
		add notification feature	☑			12/4/2020
		implement on front end	☑			7/3/2020
	Food and menu CMS	create api	☑	14/1/2020		6/3/2020
		add validation	☑			17/4/2020
		create backend function	☑			17/3/2020
		implement in front end	☑	14/1/2020		17/3/2020
	Order History	create orders retrival api	☑	16/1/2020		6/3/2020
		implement in front end	☑	16/1/2020		16/3/2020
	Transactions	create transaction retrieval api	☑			1/4/2020
		implement in front end	☑			1/4/2020
	Users details page	create user list api	☑			1/4/2020
		implement in front end	☑			15/4/2020
	Account Balance update	create api to update balance	☑			17/4/2020
		implement in front end	☐			
	Testing	create test cases	☑			17/4/2020
		test application	☑			17/4/2020

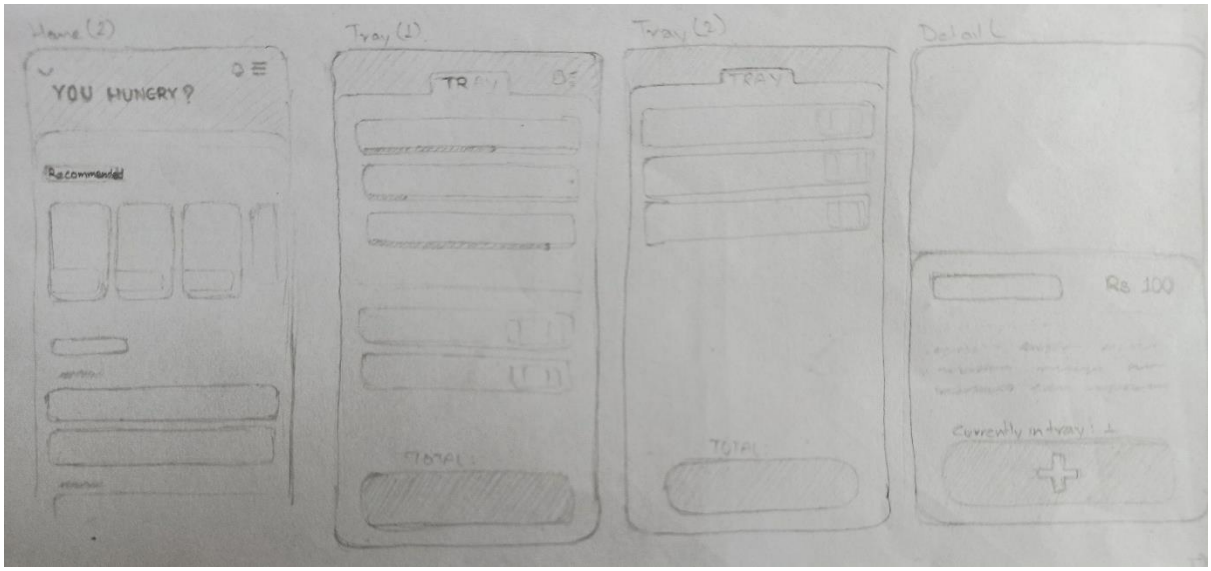
	A	B	C	D	E	F	G
55	**Name**		Deepan Singh				
56	**Project Name**		Canteen app				
57	**Student ID**		id - 17031147				
58	**Internal Supervisor**		Subeksha Shrestha				
59	**External Supervisor**		Ishwor Shrestha				
60							
61							
62	**Mobile App**						
63		**Login**	Create login api	☑	11/12/2019	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 recieved 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

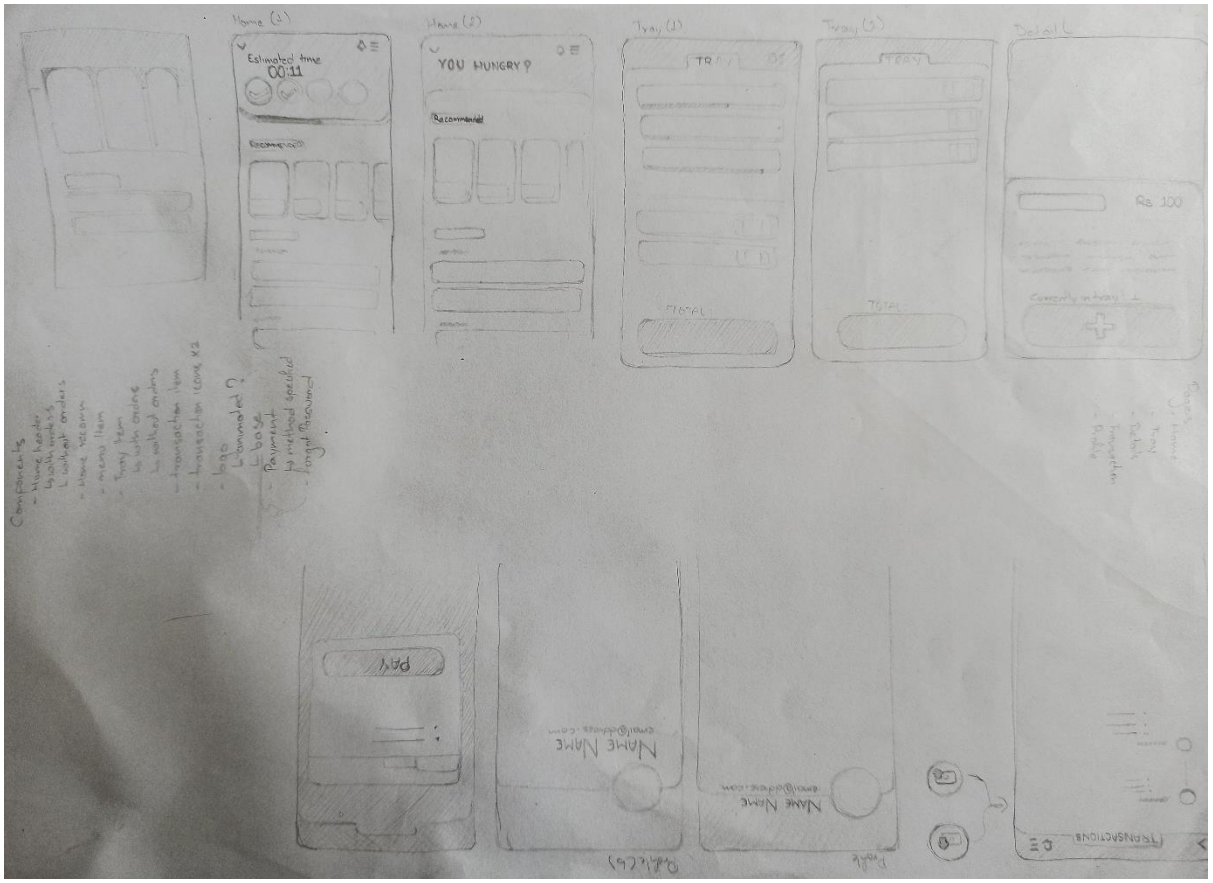
Add 1000 more rows at bottom.

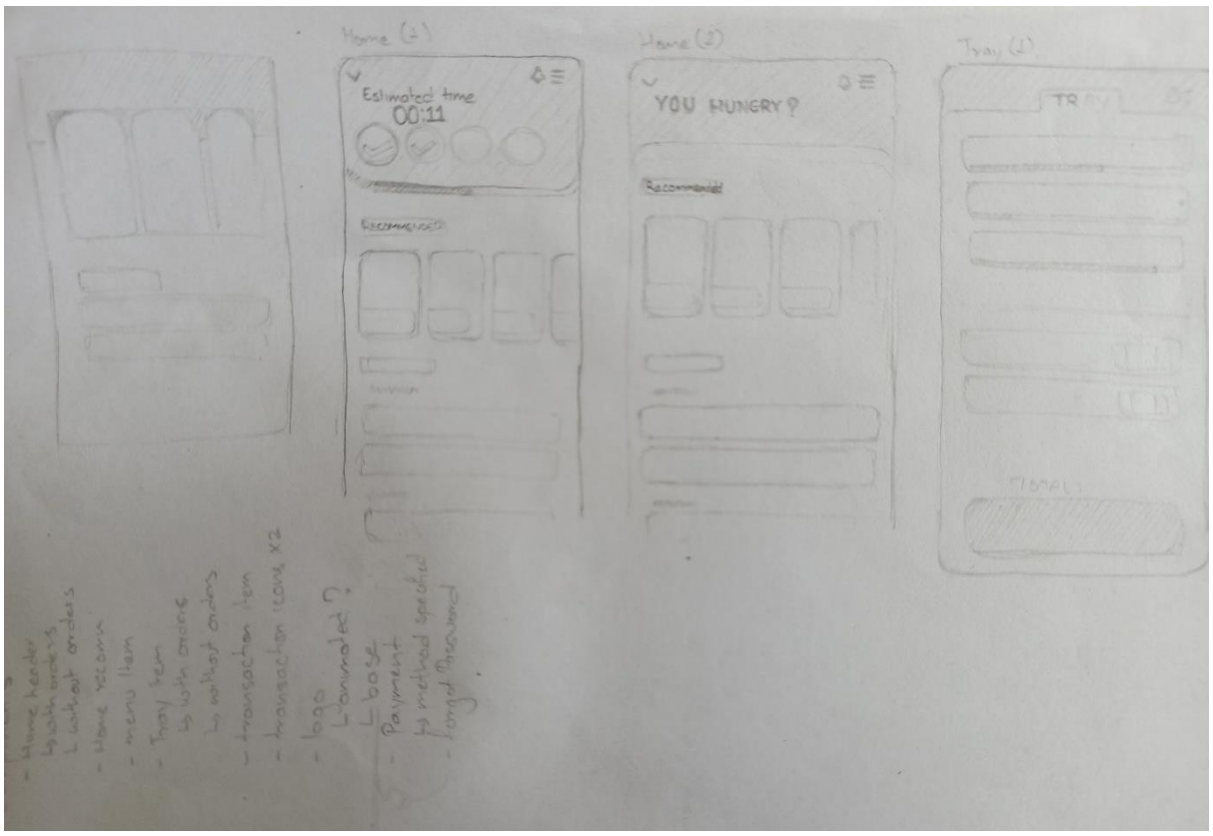
8.3.3 WIREFRAME

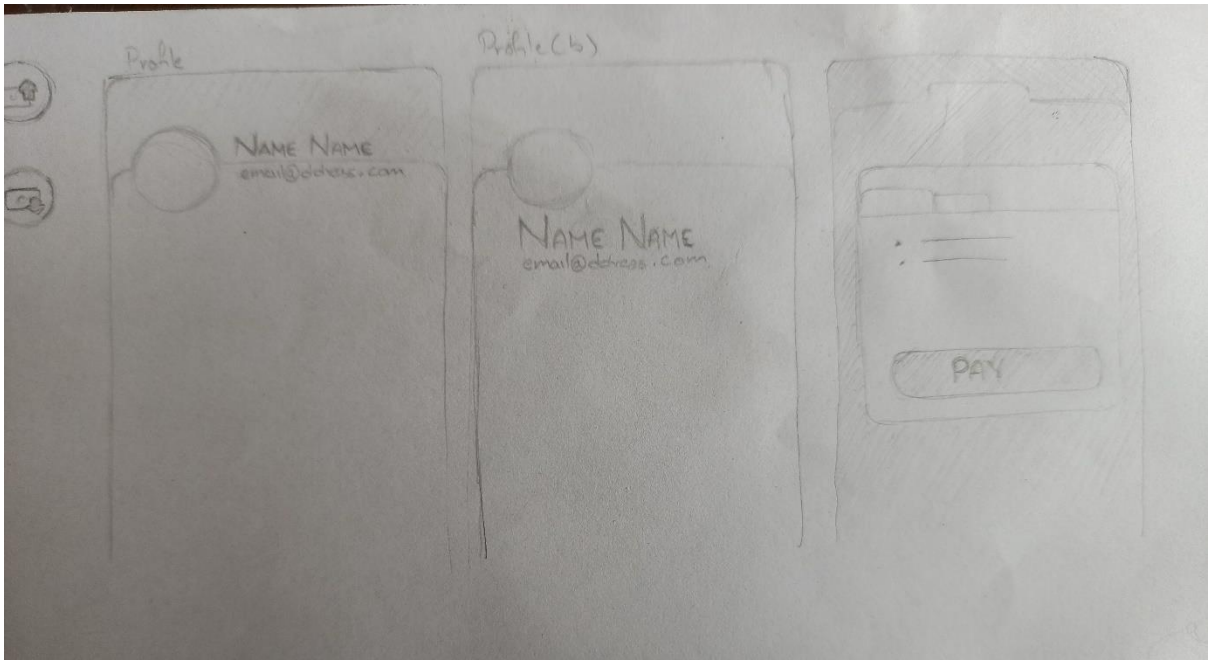
The following wireframe was designed in Adobe XD software for windows

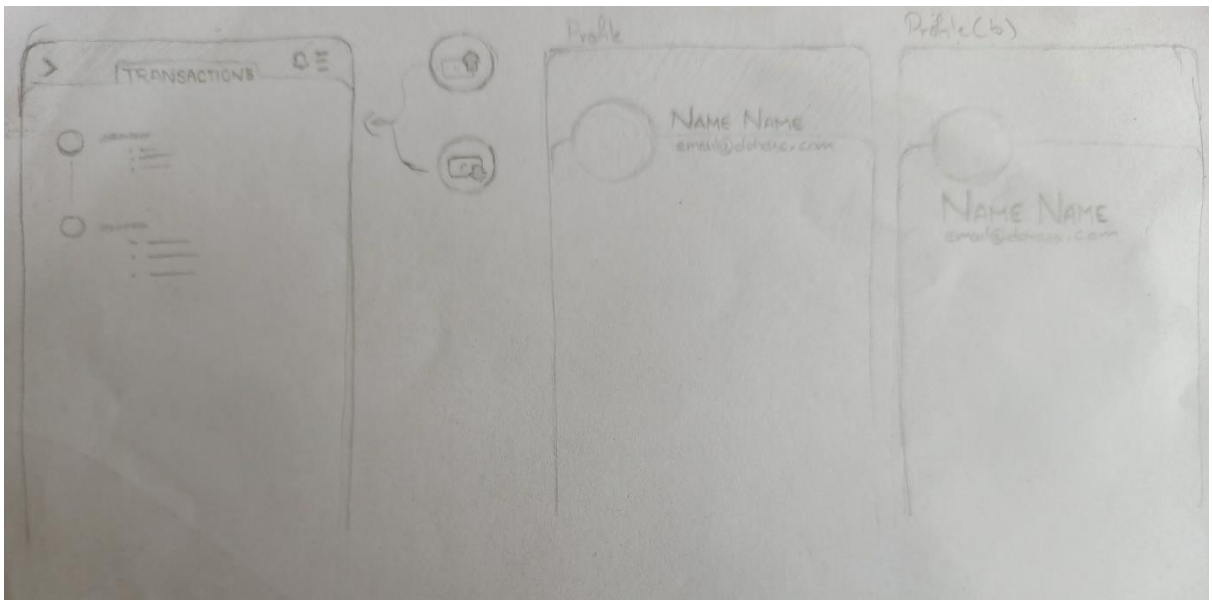


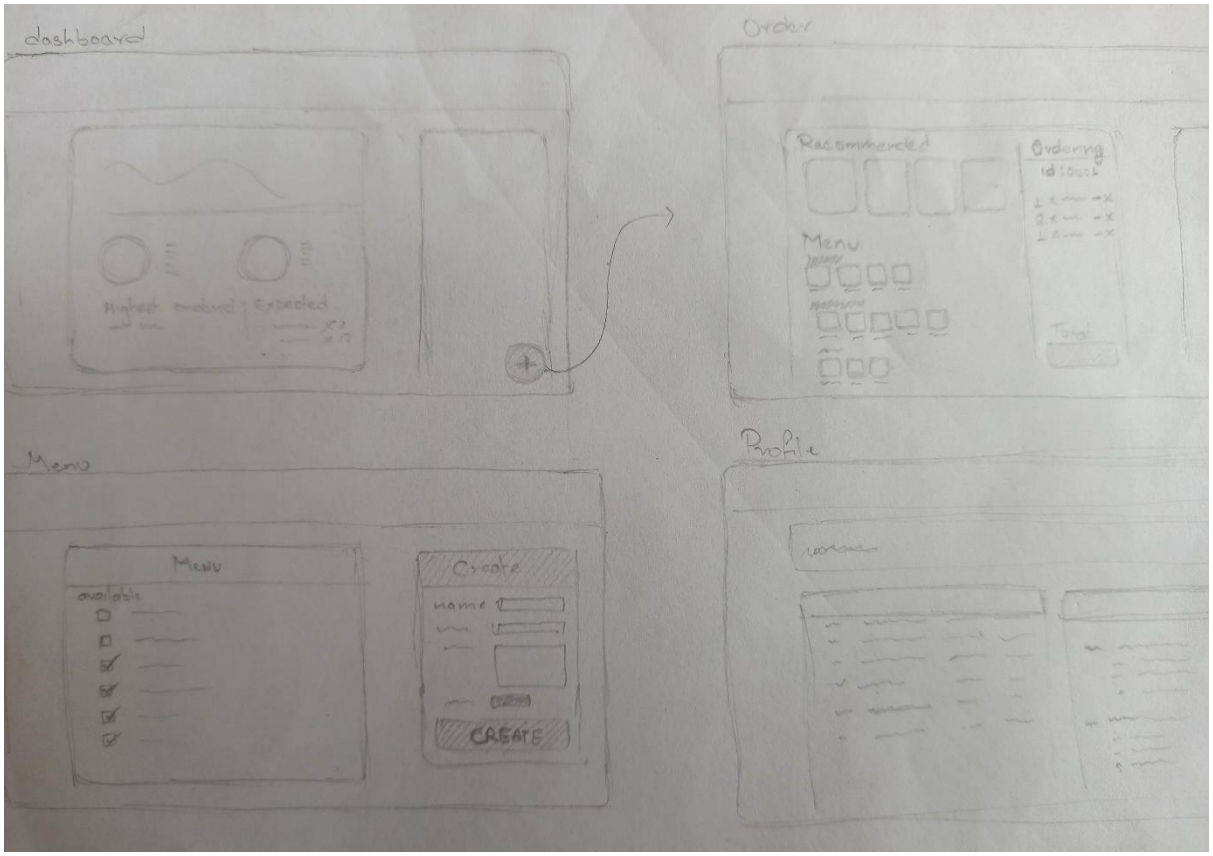



Home (1)
Estimated time
00:11
Home (2)
YOU HUNGRY?
Recommended
Tray (1)
TOTAL
Tray (2)
TOTAL
Detail
Rs 100
Components
- Home header
- Menu item
- Tray item
- Transaction item
- Logo
Payment
Name Name
TRANSACTIONS


Home (1)
Estimated time
00:11
Home (2)
YOU HUNGRY?
Recommended
Tray (1)
TOTAL
- Home header
- Menu item
- Tray item
- Transaction item
- Logo
Payment

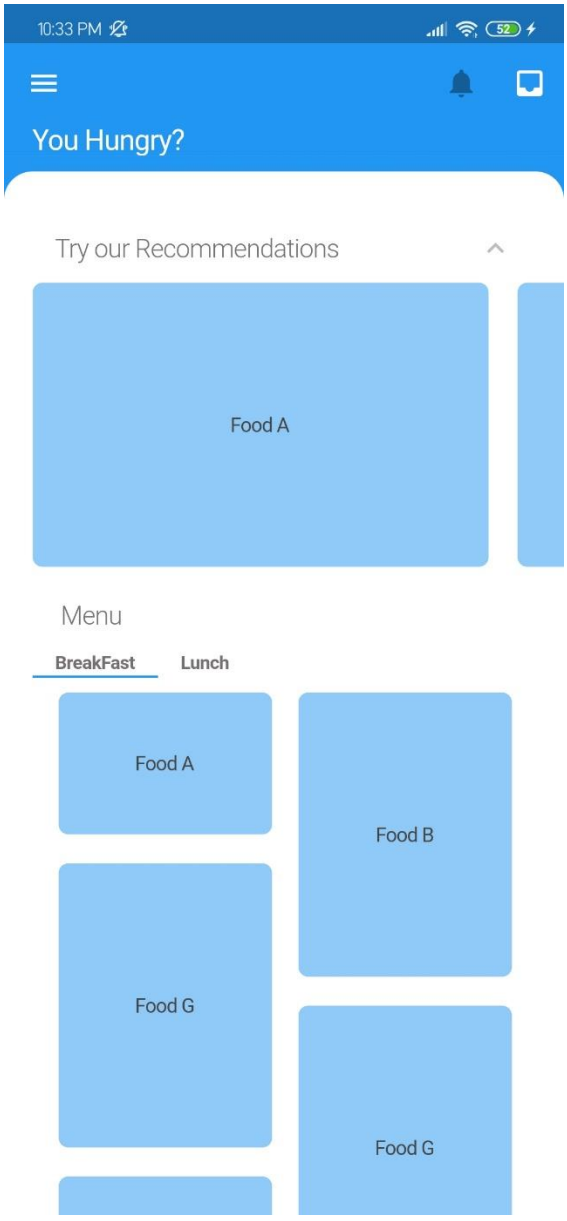

Profile
NAME NAME
email@address.com
Profile(b)
NAME NAME
email@address.com
PAY

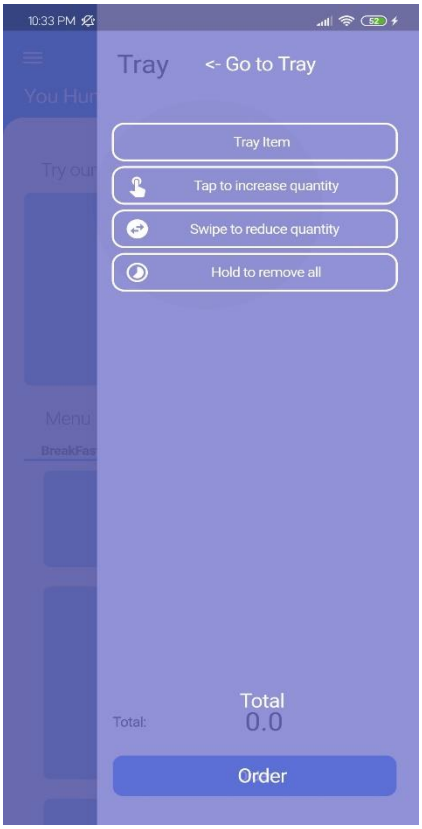

TRANSACTIONS
Profile
NAME NAME
email@address.com
Profile(b)
NAME NAME
email@address.com

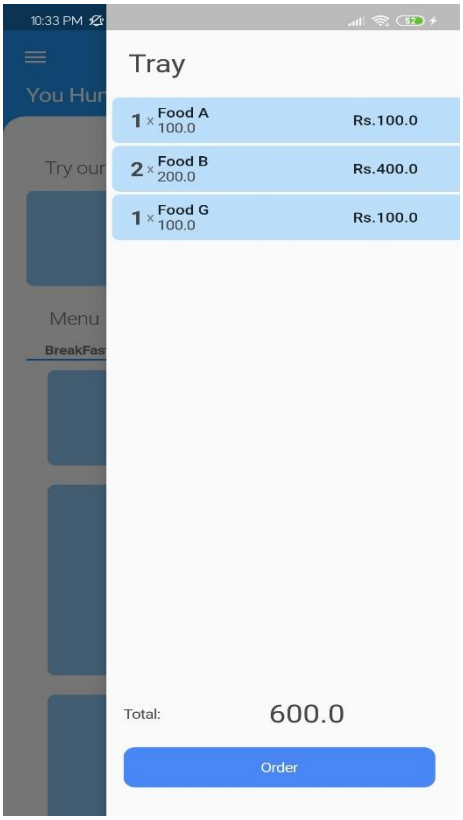

dashboard
Order
Recommended
Ordering
Menu
Total
Menu
Menu
available
Create
name
CREATE
Profile

8.4 APPENDIX E: SCREENSHOTS OF THE SYSTEM

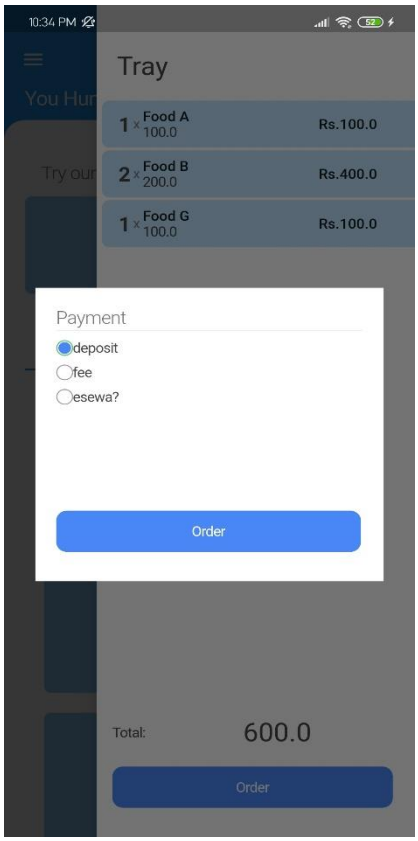
8.4.1 MOBILE APP

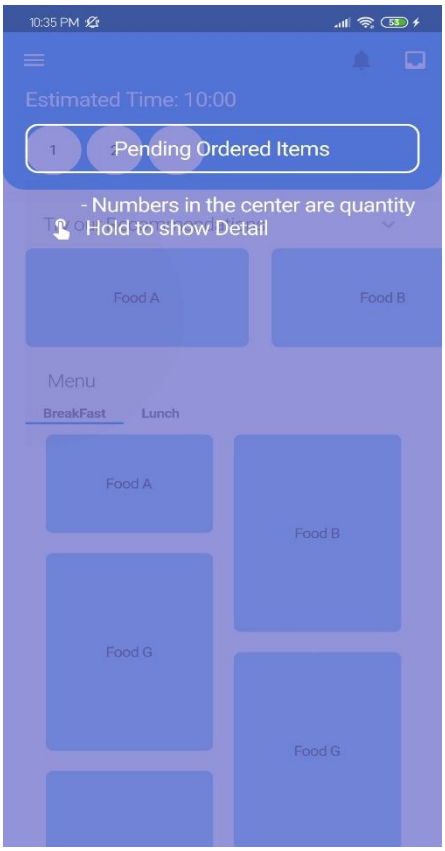


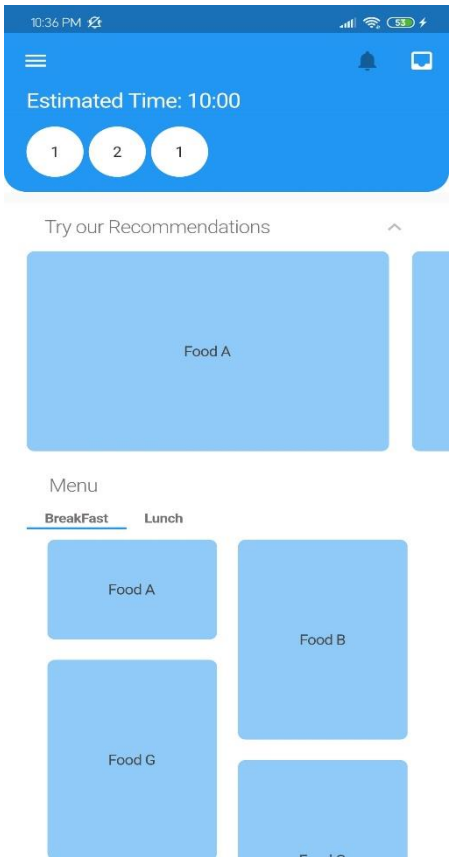

Tray
<- Go to Tray
Tray Item
Tap to increase quantity
Swipe to reduce quantity
Hold to remove all
Total:
Total
0.0
Order

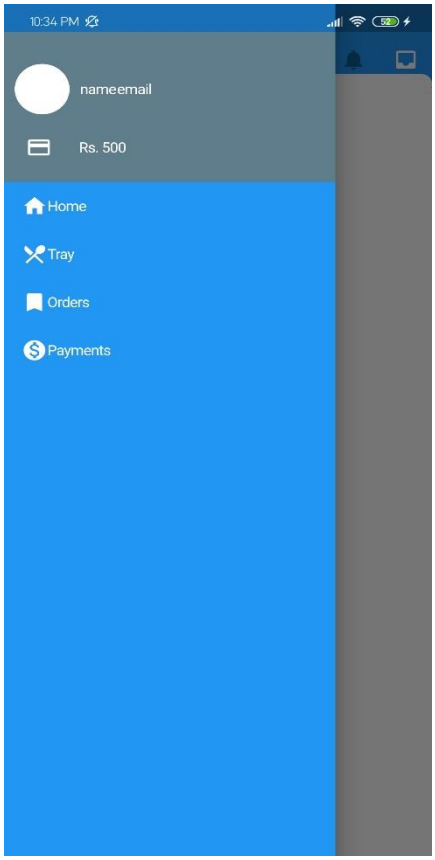

Tray
1 x Food A 100.0 Rs.100.0
2 x Food B 200.0 Rs.400.0
1 x Food G 100.0 Rs.100.0
Total:
600.0
Order

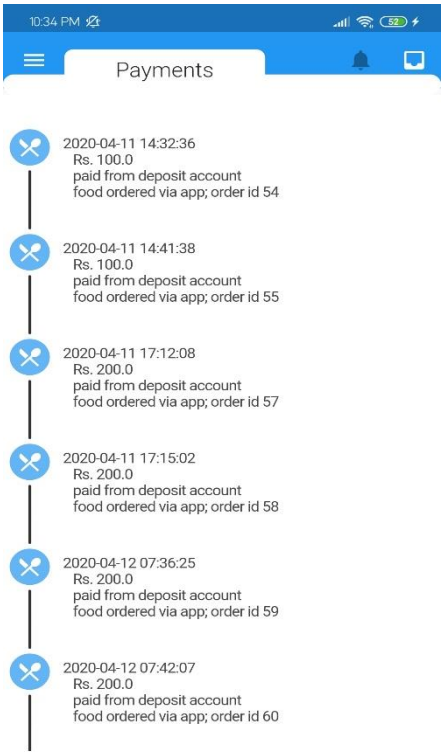

TRAY
100.0 Food A 1
200.0 Food B 2
100.0 Food G 1
total
600.0
clear
Order


Tray
1 x Food A 100.0 Rs.100.0
2 x Food B 200.0 Rs.400.0
1 x Food G 100.0 Rs.100.0
Payment
deposit
fee
esewa?
Order
Total:
600.0
Order


Estimated Time: 10:00
Pending Ordered Items
- Numbers in the center are quantity
Hold to show Detail
Food A
Food B
Menu
BreakFast
Lunch
Food A
Food B
Food G
Food G

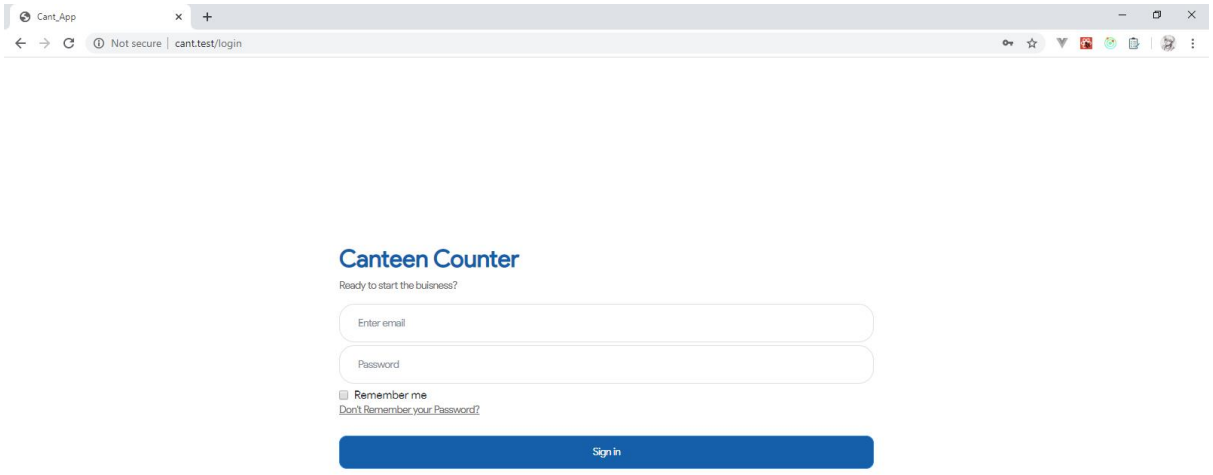

Estimated Time: 10:00
1
2
1
Try our Recommendations
Food A
Menu
BreakFast
Lunch
Food A
Food B
Food G

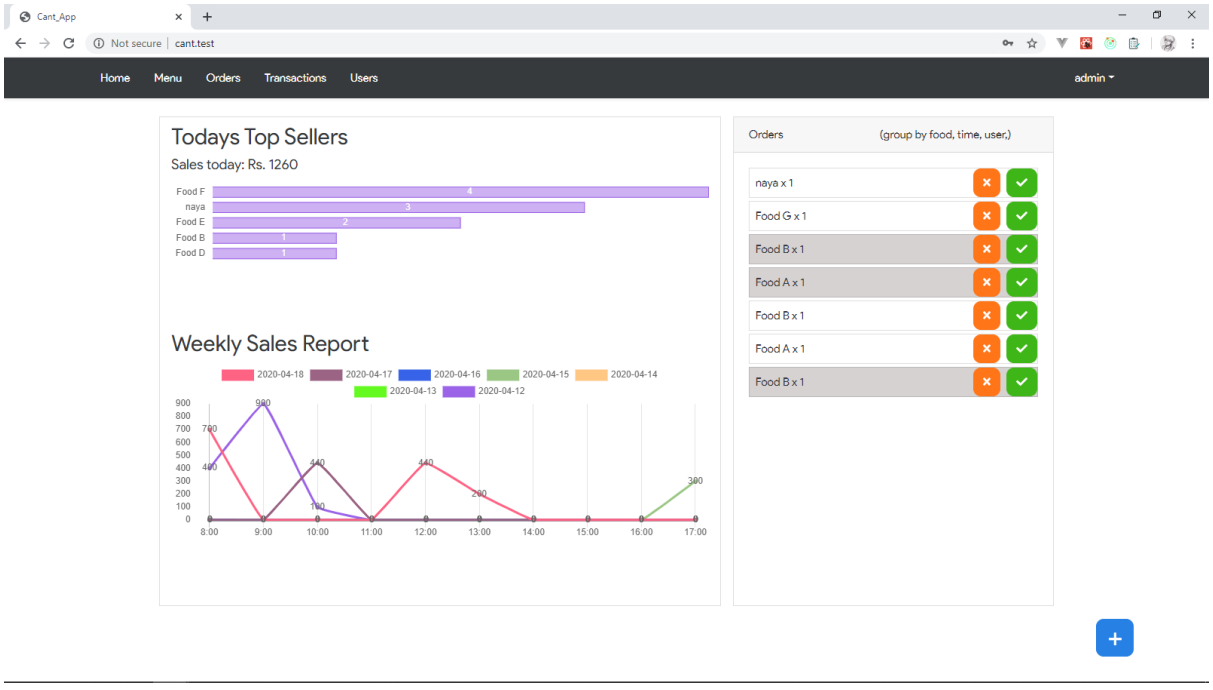

10:34 PM
nameemail
Rs. 500
Home
Tray
Orders
Payments

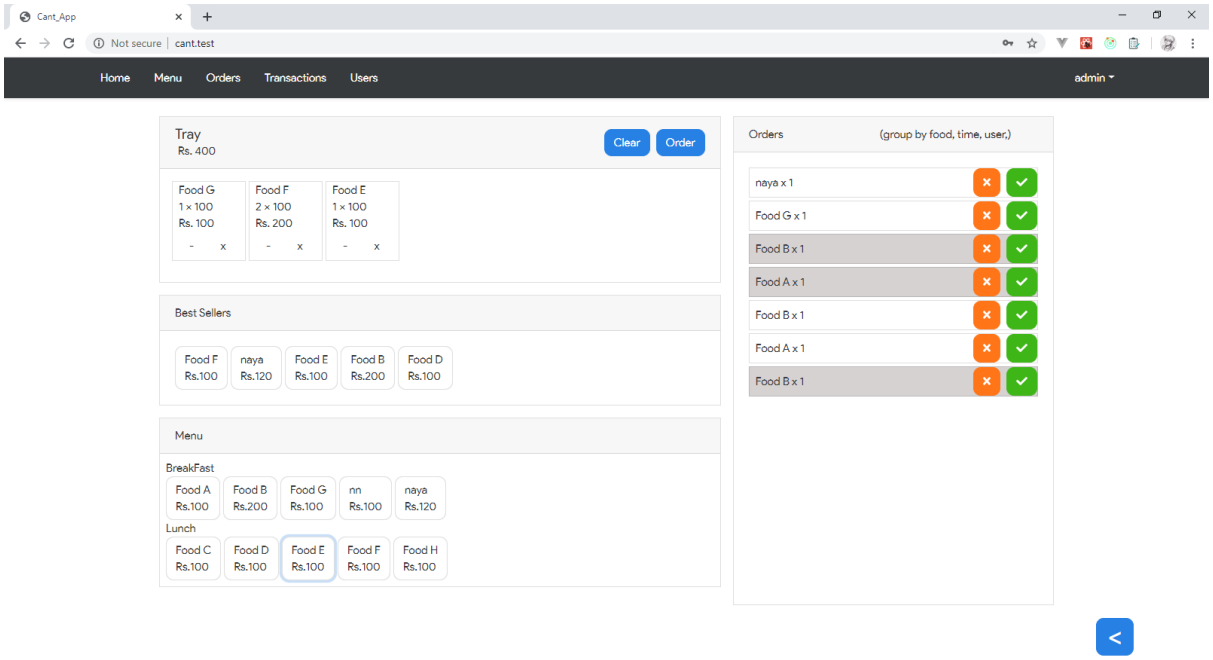

10:34 PM
Payments
2020-04-11 14:32:36
Rs. 100.0
paid from deposit account
food ordered via app; order id 54
2020-04-11 14:41:38
Rs. 100.0
paid from deposit account
food ordered via app; order id 55
2020-04-11 17:12:08
Rs. 200.0
paid from deposit account
food ordered via app; order id 57
2020-04-11 17:15:02
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

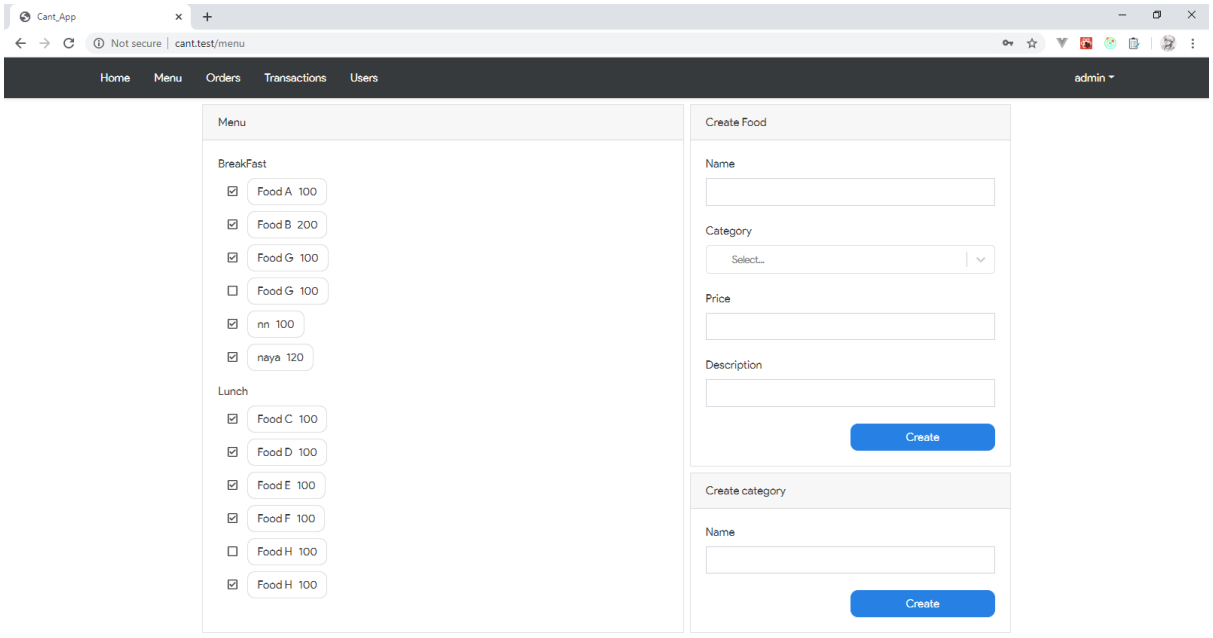

10:34 PM
Orders
Order id - 89
Total - Rs. 120.0
2020-04-19 00:00:00
1 * naya @ 120.0
Order id - 88
Total - Rs. 200.0
2020-04-19 00:00:00
2 * Food F @ 100.0
Order id - 87
Total - Rs. 500.0
2020-04-19 00:00:00
2 * Food E @ 100.0
1 * Food B @ 200.0
1 * Food D @ 100.0
Order id - 86
Total - Rs. 440.0
2020-04-19 00:00:00
2 * naya @ 120.0
2 * Food F @ 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

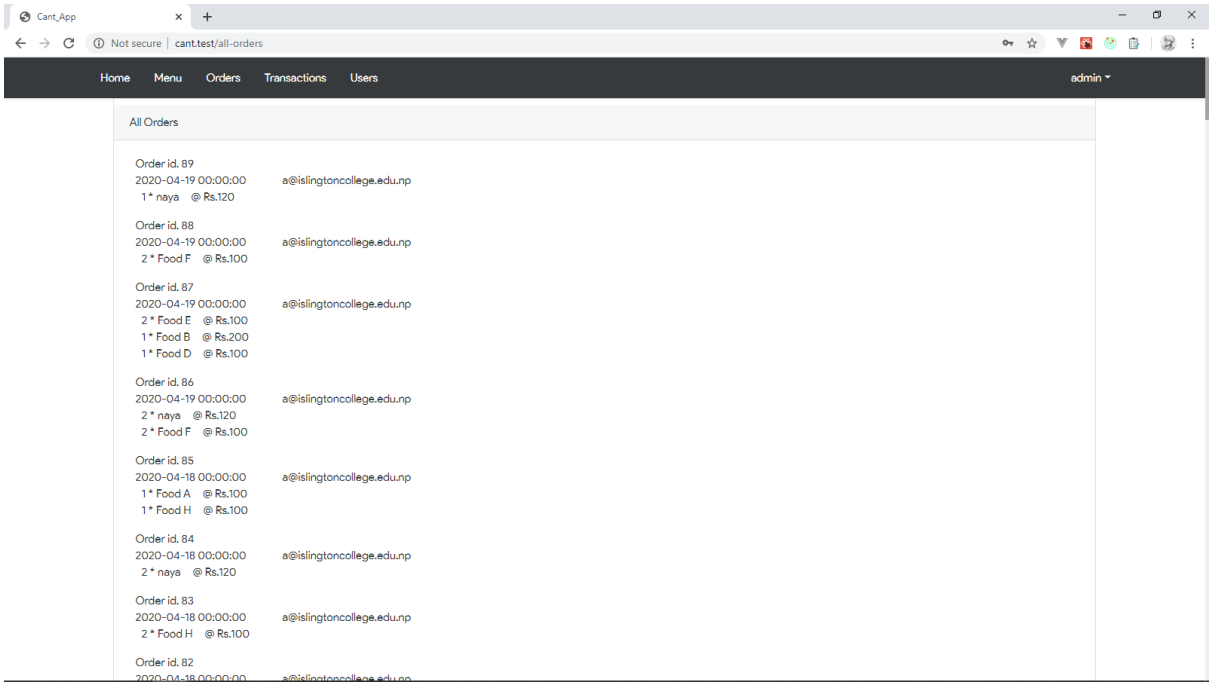
8.4.2 WEB APPLICATION

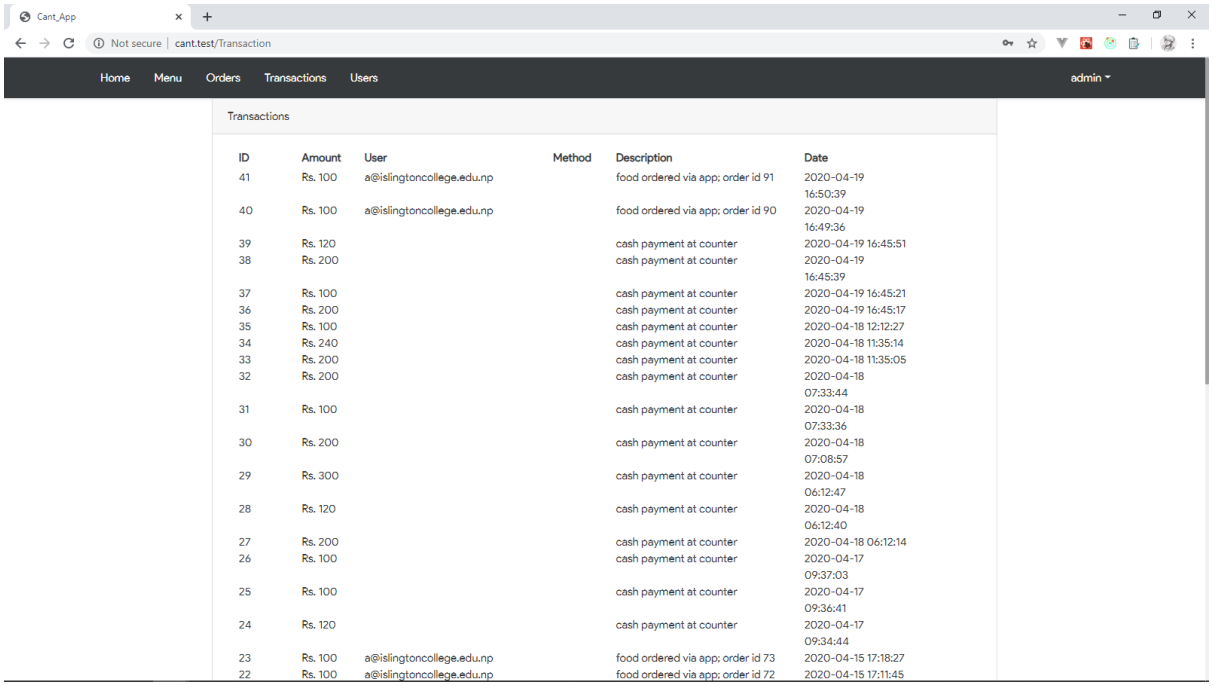


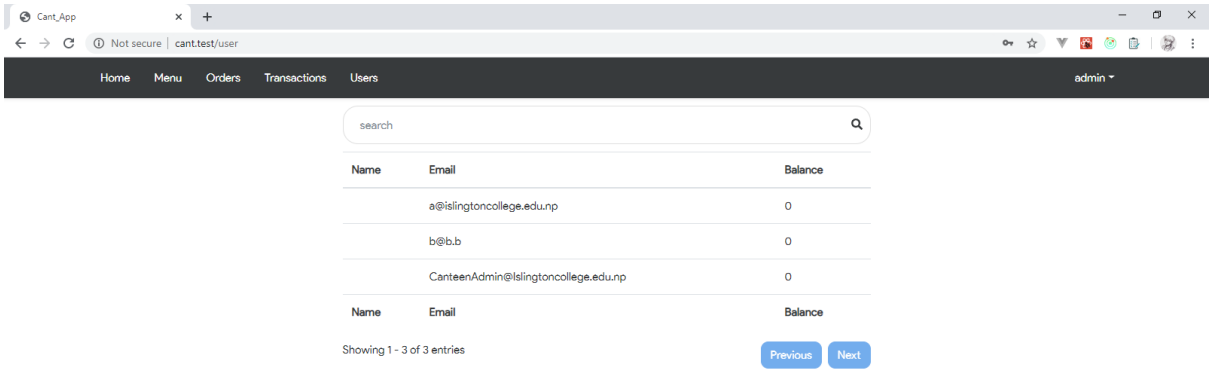


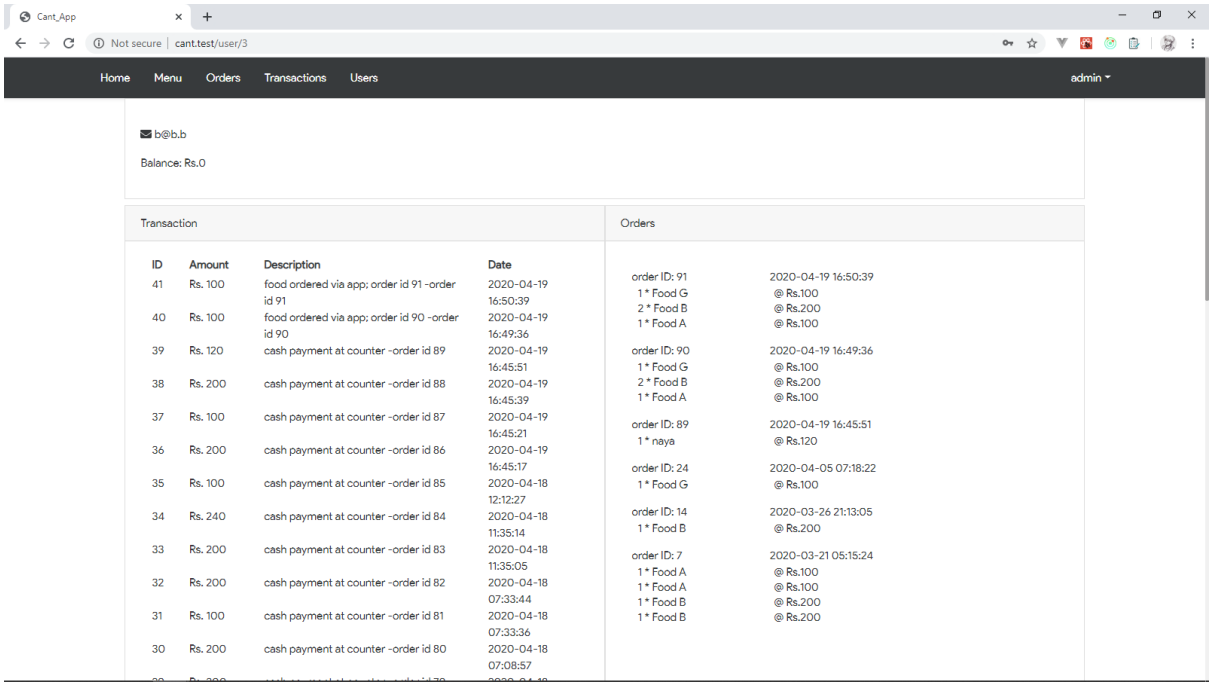

Cant_App
Not secure | cant.test
Home
Menu
Orders
Transactions
Users
admin
Tray
Rs. 400
Clear
Order
Food G
1 × 100
Rs. 100
Food F
2 × 100
Rs. 200
Food E
1 × 100
Rs. 100
Best Sellers
Food F Rs.100
naya Rs.120
Food E Rs.100
Food B Rs.200
Food D Rs.100
Menu
BreakFast
Food A Rs.100
Food B Rs.200
Food G Rs.100
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
Food G x 1
Food B x 1
Food A x 1
Food B x 1
Food A x 1
Food B x 1

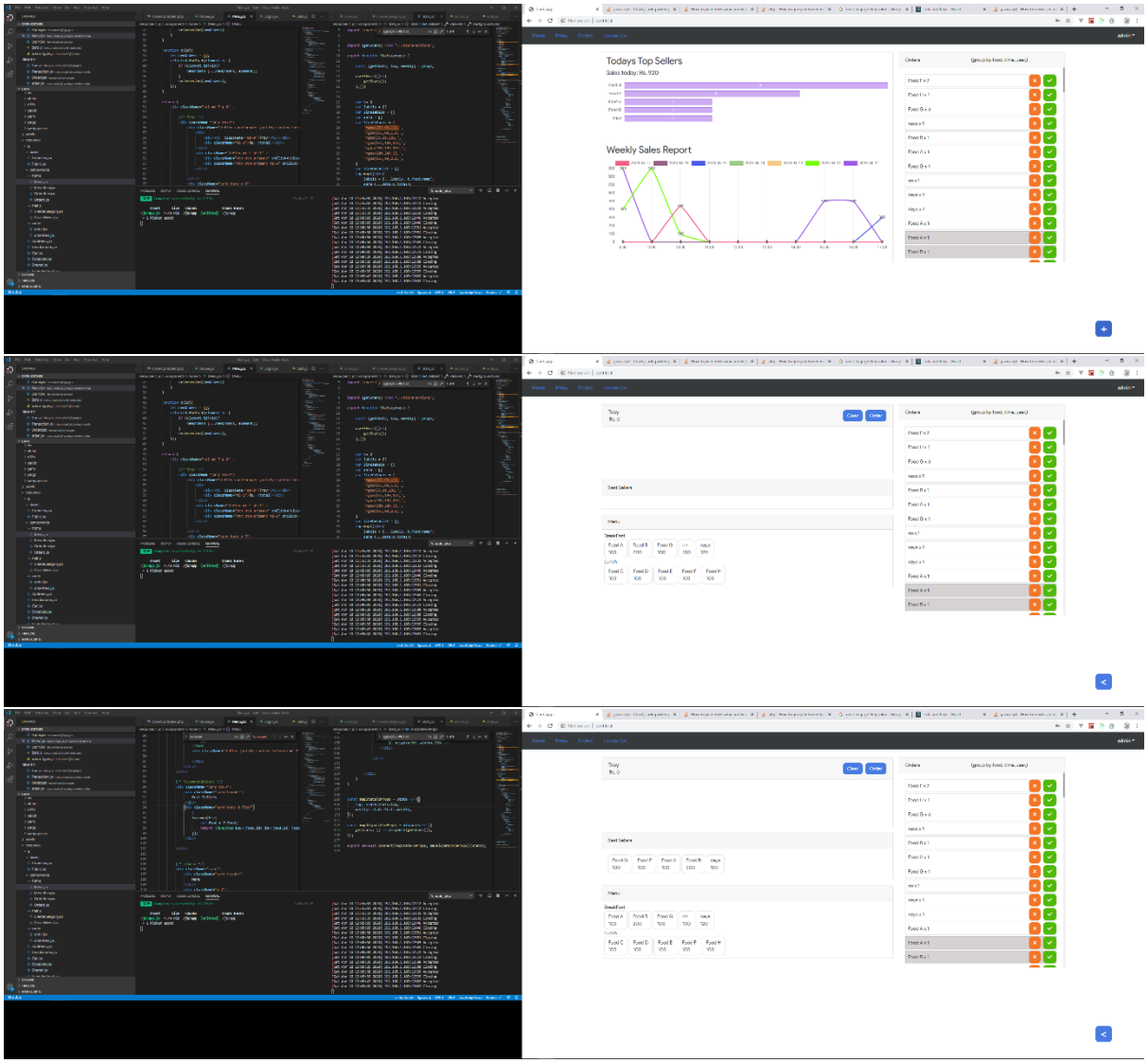

Cant_App
Not secure | cant.test/menu
Home
Menu
Orders
Transactions
Users
admin
Menu
BreakFast
Food A 100
Food B 200
Food G 100
Food G 100
nn 100
naya 120
Lunch
Food C 100
Food D 100
Food E 100
Food F 100
Food H 100
Food H 100
Create Food
Name
Category
Select...
Price
Description
Create
Create category
Name
Create


Cant_App
Not secure | cant.test/all-orders
Home
Menu
Orders
Transactions
Users
admin
All Orders
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


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

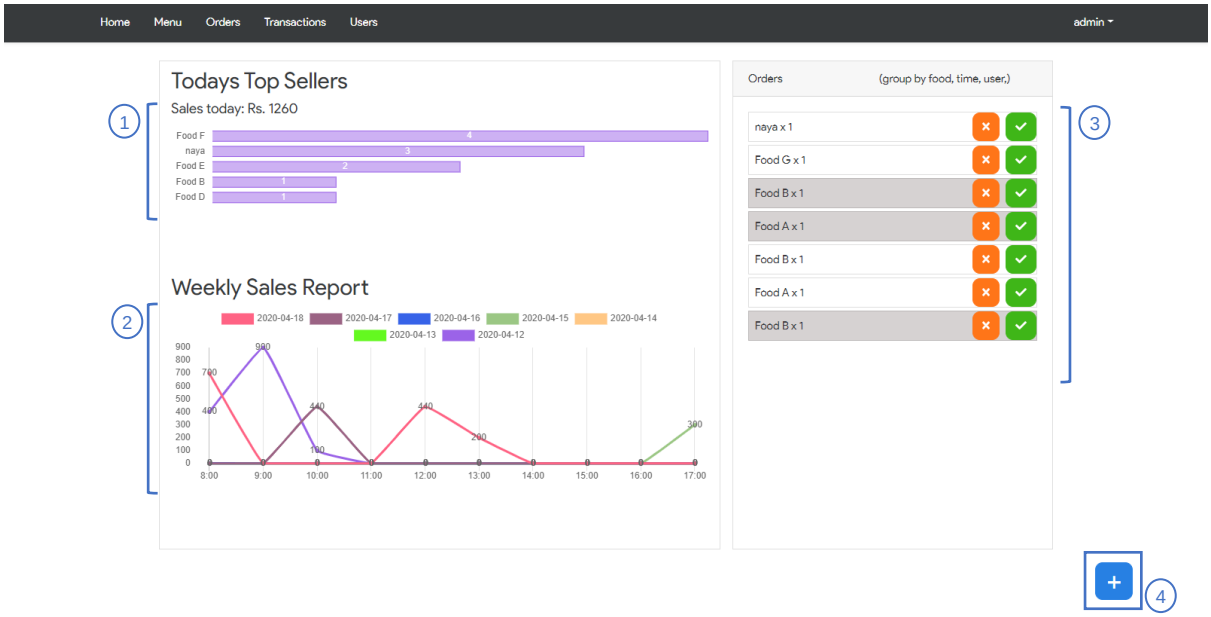

Cant_App
Not secure | cant.test/user
Home
Menu
Orders
Transactions
Users
admin
search
Name
Email
Balance
a@islingtoncollege.edu.np
0
b@b.b
0
CanteenAdmin@Islingtoncollege.edu.np
0
Name
Email
Balance
Showing 1 - 3 of 3 entries
Previous
Next


Cant_App
Not secure | cant.test/user/3
Home
Menu
Orders
Transactions
Users
admin
b@b.b
Balance: Rs.0
Transaction
ID
Amount
Description
Date
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
2020-04-18 12:12:27
34
Rs. 240
cash payment at counter -order id 84
2020-04-18 11:35:14
33
Rs. 200
cash payment at counter -order id 83
2020-04-18 11:35:05
32
Rs. 200
cash payment at counter -order id 82
2020-04-18 07:33:44
31
Rs. 100
cash payment at counter -order id 81
2020-04-18 07:33:36
30
Rs. 200
cash payment at counter -order id 80
2020-04-18 07:08:57
Orders
order ID: 91
2020-04-19 16:50:39
1 * Food G @ Rs.100
2 * Food B @ Rs.200
1 * Food A @ Rs.100
order ID: 90
2020-04-19 16:49:36
1 * Food G @ Rs.100
2 * Food B @ Rs.200
1 * Food A @ Rs.100
order ID: 89
2020-04-19 16:45:51
1 * naya @ Rs.120
order ID: 24
2020-04-05 07:18:22
1 * Food G @ Rs.100
order ID: 14
2020-03-26 21:13:05
1 * Food B @ Rs.200
order ID: 7
2020-03-21 05:15:24
1 * Food A @ Rs.100
1 * Food A @ Rs.100
1 * Food B @ Rs.200
1 * Food B @ Rs.200

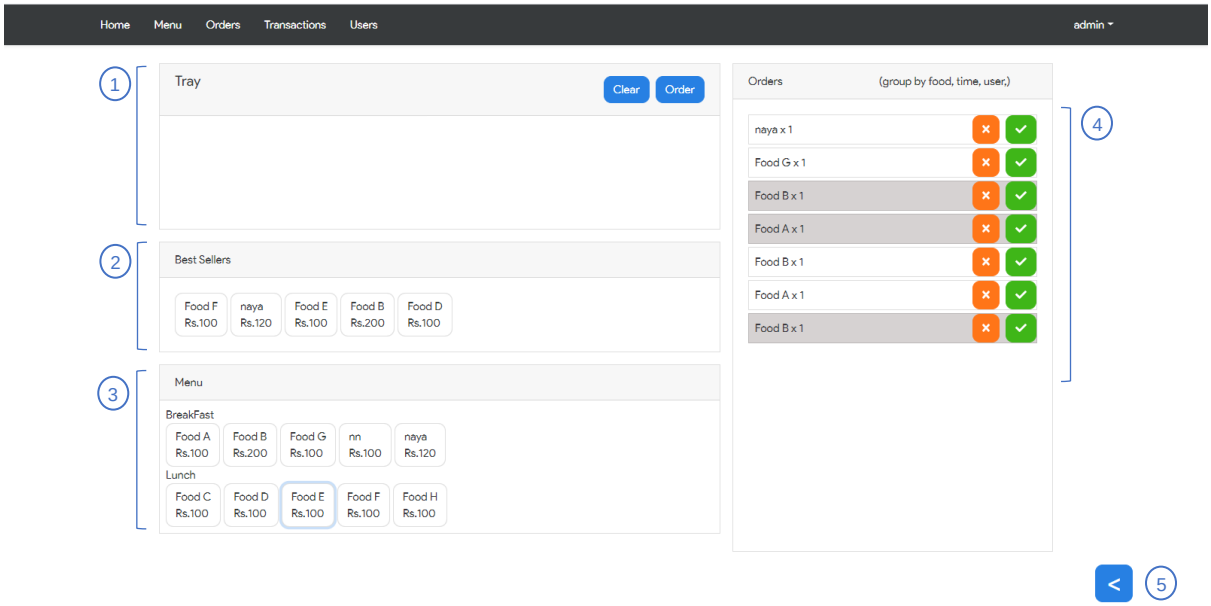

Todays Top Sellers
Weekly Sales Report

8.6: USER MANUAL

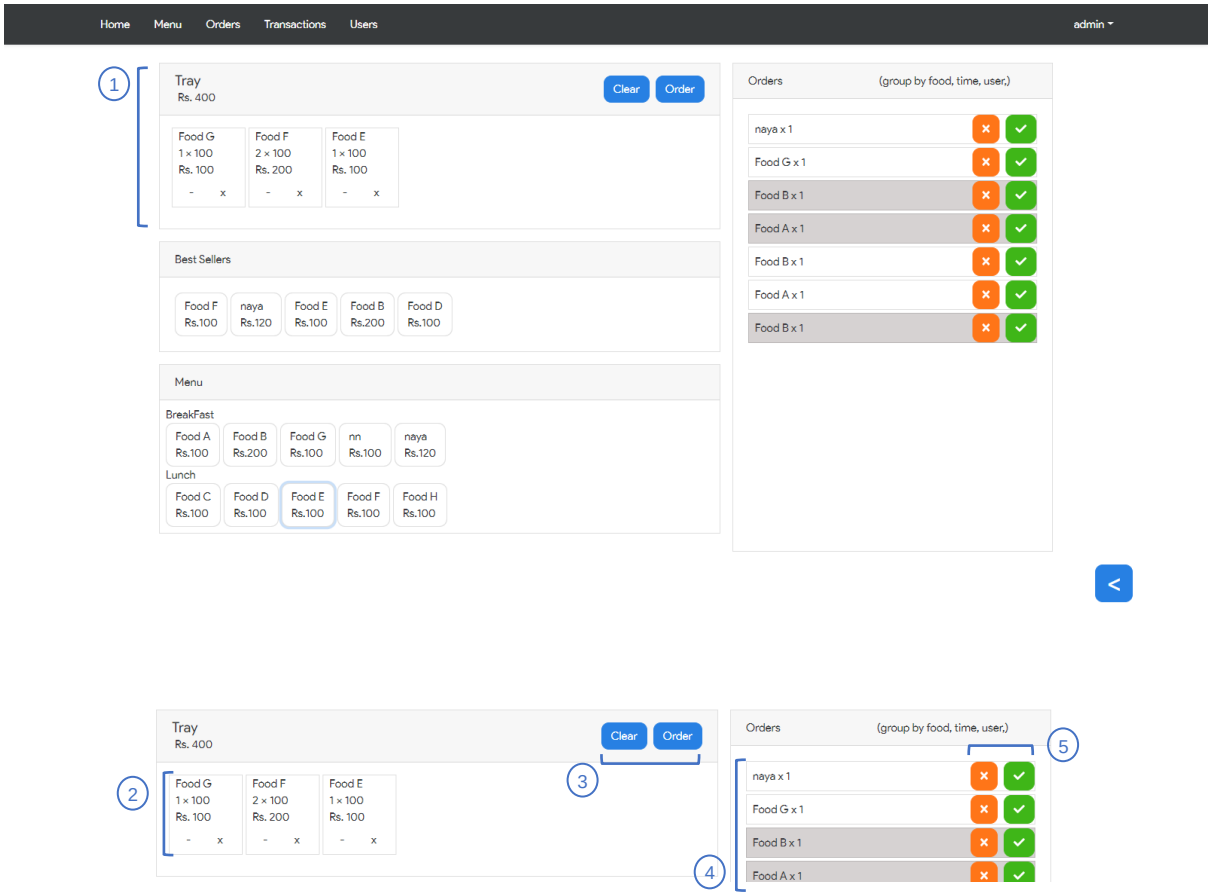
Home Page



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.



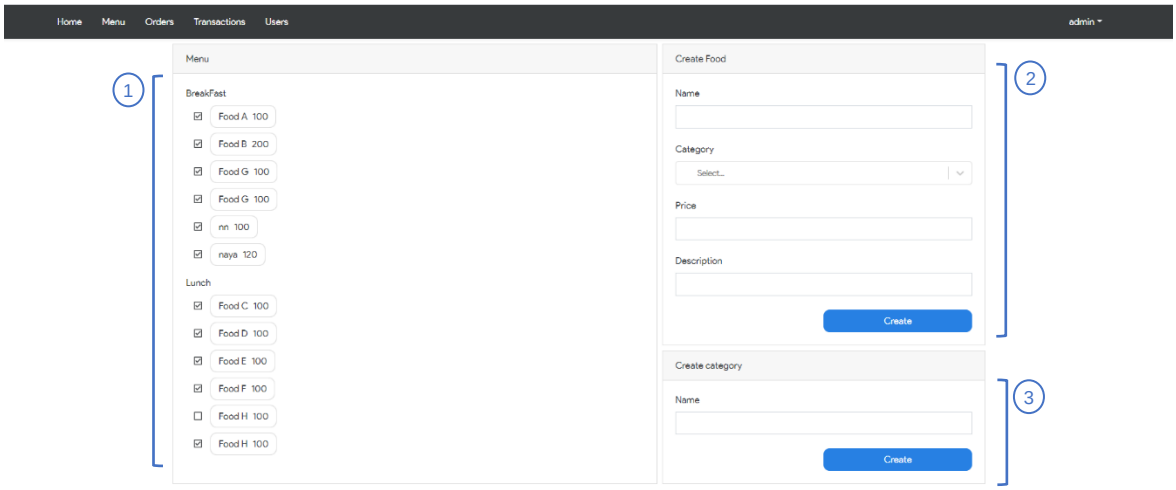
Clicking the bottom right button swaps the stats page with the order page. The user can revert to the stats panel by clicking same button (5). The top section (1) is the tray which contains the selected items. Items can be added from the recommendation (2) or the menu (3) by clicking on the item itself.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



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.

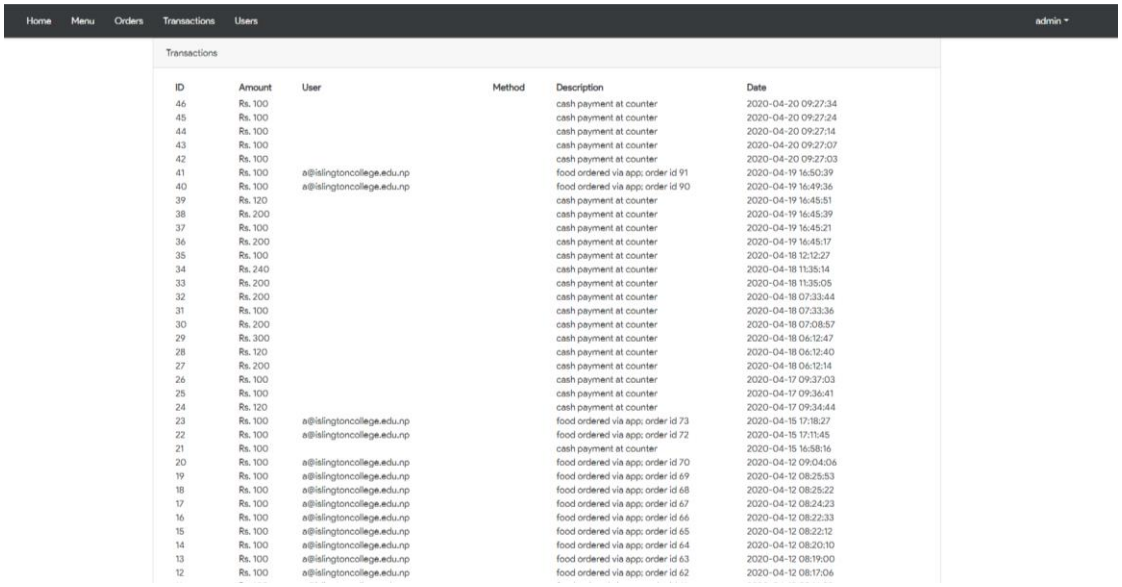
Once ordered, the items are listed on the right panel (4) which can be used to update the status and notify the customers. Clicking on the item itself sets the item as “cooking” and are marked grey. Clicking the buttons (5) ‘ x ’ and ‘ ✓ ’ will cancel or complete the order and notify the users accordingly.

Menu Page



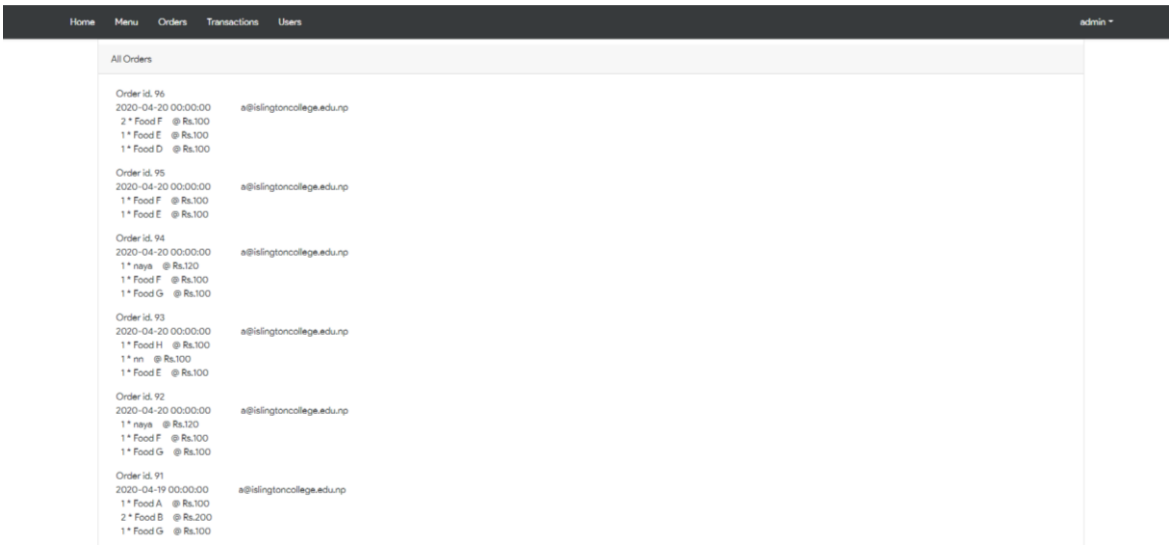
The menu creation page can be accessed from the menu tab in the navigation. The page lists all created food items in the central component (1). The item that are available on the menu are checked (☑). User can change the availability of the item by simply checking or unchecking the boxes for each item. 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



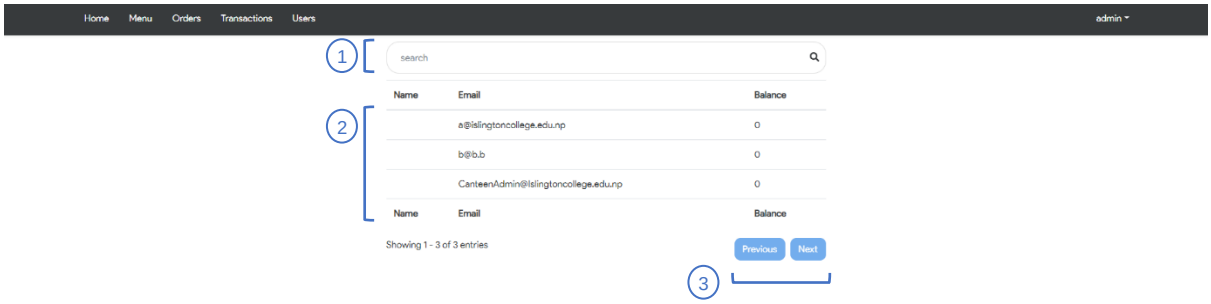
The transaction page accessible from the navigation bar list all the transactions till date. More specific data can be seen on individual user pages.

Orders Page



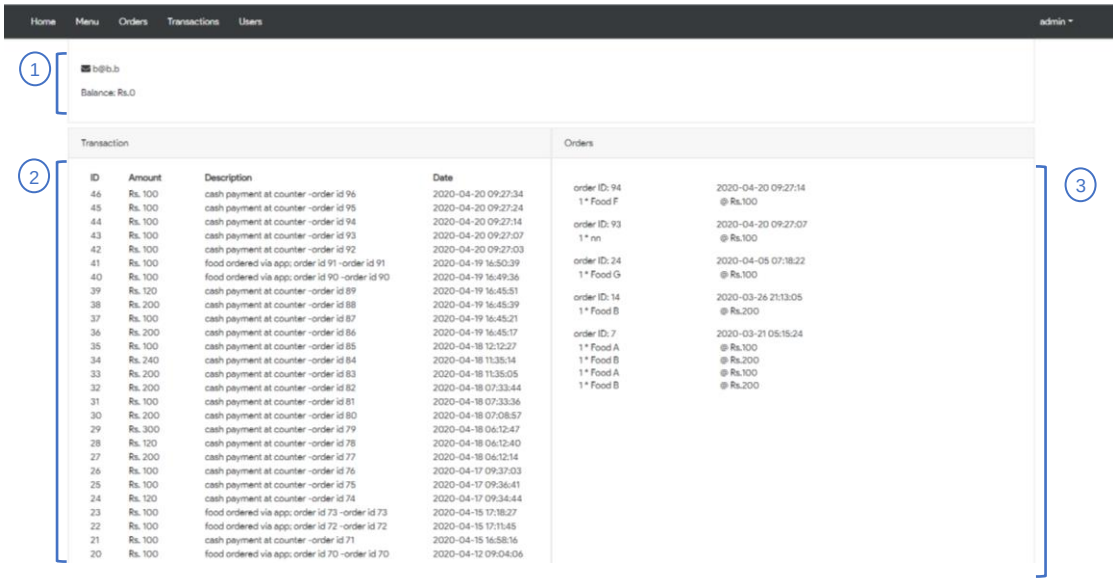
The Orders page, similar to transaction, list all the orders sorted according to time. Also, more specific data can be seen on individual user pages.

Users Page



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 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.