BABLUS

2021-2022

A PROJECT REPORT

Submitted by

VIGHNESHA N U - 191322743

KARTHIK V RAO - 191322718

NISHMITHA J SHETTY - 191322723

In partial fulfilment for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS

Under the guidance of

Mrs SWARNAGOWRI S S

Lecturer

Department of Computer Science

GOVERNMENT FIRST GRADE COLLEGE

THENKANIDIYOOR, UDUPI

MANGALORE UNIVERSITY

MANGALORE UNIVERSITY

GOVERNMENT FIRST GRADE COLLEGE

THENKANIDIYOOR, UDUPI

Department of Computer Science

Certificate

Certified that the project work entitled

Bablus

..... is a benefited work carried out by

VIGHNESHA N U	**KARTHIK V RAO**	**NISHMITHA J SHETTY**
Reg No: 191322743	Reg No: 191322718	Reg No: 191322723

in partial fulfilment for the award of degree of Bachelor of Computer Applications of the Mangalore University during the year 2021-22. The project report has been approved as it satisfies the academic requirements of project work prescribed for the Bachelor of Computer Applications.

Signature of the Project Guide	***Signature of the HOD***	***Signature of the Principal***

Examiners: 1. 2.

DECLARATION

We Vighnesha N U (Reg-191322743), Karthik V Rao (Reg-191322718) and Nishmitha J Shetty (Reg-191322723), do hereby declare that, the project entitled “Bablus” is the original work carried out by me and my teammates during the sixth semester BCA, submitted to Mangalore University, for the partial fulfillment of the requirement of Bachelor of Computer Applications

I also declare this project has not been previously submitted for any other degree or award.

Place : Thenkanidiyur, Udupi
Date : 26/08/2022

Name : Vighnesha N U
Register No: 191322743

Name: Karthik V Rao
Register No: 191322718

Name: Nishmitha J Shetty
Register No: 191322723

ACKNOWLEDGEMENT

The magnitude of this project demanded the co-operation and assistance from a number of people and by the grace of God we have been fortunate enough to have this in the entire Process of our project work.

We would like to thank our principal Dr. Suresh Rai K for providing us the facilities to carry out the project work.

We thank Mr. Basavaraj U, Head of the Computer Science Department Govt. First Grade College, Thenkanidiyur for having permitted us to take this project.

We express grateful thanks to our internal guide Mrs.Swarnagowri S S, Department of Computer Science, Govt First Grade College, Udupi for her guidance throughout the work.

We would like to thank all the teaching and non-teaching staff member of the Computer Science Department, Govt First Grade College Thenkanidiyur for their co-operation and constant encouragement which help us in successfully completing our project.

Finally our heart fully gratitude to all those who helped us directly and indirectly in completing this project successfully.

- **Vighnesha N U**
- **Karthik V Rao**
- **Nishmitha J Shetty**

Table of Contents

1. Introduction

1.1 Introduction of the System

1.1.1 Project Title:

Bablus (Saloon Booking App)

1.1.2 Category:

Mobile Application

1.1.3 Overview

Bablus is an application that helps users to find the Salon and beauty parlors around by searching pin code, address, or location. The application allows users to find the best service centers in various locations based on the reviews and ratings. The primary objective of the platform includes booking a service or an appointment.

In a modern busy day-to-day life, it is always convenient to look for services that one requires through mobile phones is the best practice to save time and effort. Bablus is designed using cutting-edge technologies that help users to find solutions at their fingertips. The app is user-friendly, convenient, free of cost, and adds value to its users' needs.

1.2 Background

1.2.1 Introduction of the Company

Not applicable

1.2.2 Brief note on Existing System

Currently a person, who wishes to have saloon service, will go to his nearby saloon shops and ask for a seat booking and wait for some time if it's not available. After some time when his number comes then, he can avail of the service.

1.3 Objectives of the System

- To help find the Salons and Beauty parlors nearby
- Easily choose the best center/shop based on information, reviews, and ratings
- Users can directly book appointments for their needs without hassling which helps save time
- It helps local businesses to increase more leads and sales that generate excellent opportunities as well as income

1.4. Scope of the System

- Transforming the traditional businesses to online
- Reducing the waiting time for salon services

1.5 Structure of the System

1.5.1 Authentication

1.5.1.1 Registration

Allows new users to register on the application

1.5.1.2 Login

Helps existing users to login into the application

1.5.1.3 Forget Password

Helps the user to reset the password, whenever they forgot.

1.5.2 Admin

1.5.2.1 Manage Categories

Helps Admin to create, view, update or delete different categories.

1.5.2.2 Manage Shops

Helps Admin to view, update or delete existing shops in the app.

1.5.2.3 Manage Users

Helps Admin to view, update or delete existing users in the app.

1.5.3 Shop

1.5.3.1 Register/ login Shop

Allows the user to register their shop/business with our app

1.5.3.2 Manage Shop/Business

Helps Shop to view, update or delete shop details.

1.5.3.3 Manage Services

Helps Shop to add, view, update or delete different services in the shop.

1.5.3.4 Manage Appointments

Helps Shop to view, accept or reject appointment requests.

1.5.4 User

1.5.4.1 Profile Management

Helps users to view and update their profile information.

1.5.4.2 View Shops

Helps users to view all shops based on relativity and ratings

1.5.4.3 Shop Details

Helps the users to view the shop details, contact, services, ratings, and reviews.

1.5.2.4 Appointment

Helps users to book appointments, and view and manage appointments.

1.5.2.5 Review

Allows the users to share reviews on a shop.

1.6 System Architecture

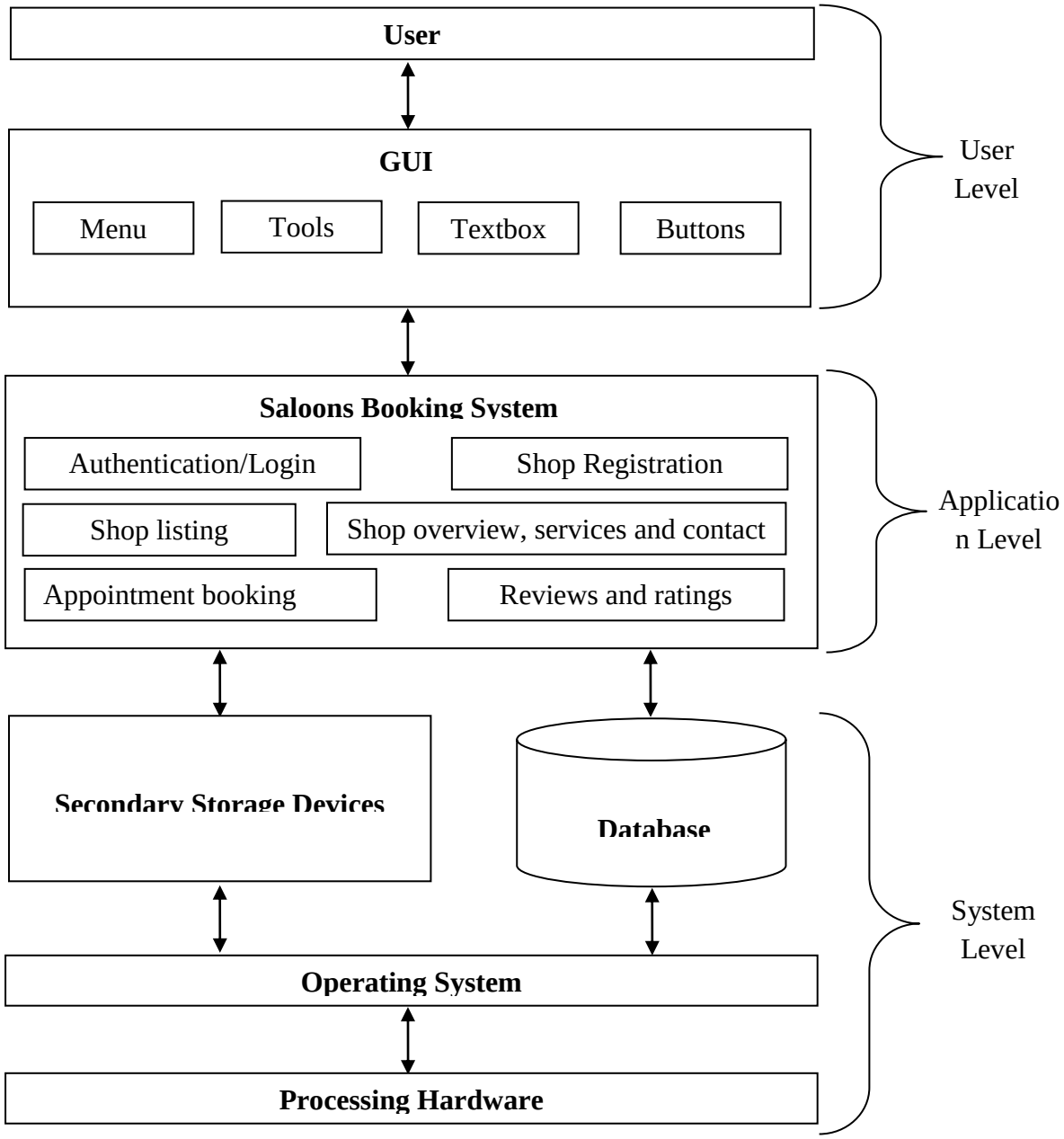


Fig.No: 1.1 - System Architecture

1.7 End Users

- Admin who is managing the whole system
- Shop owner and members running saloon services
- User who is booking appointments

1.8 Software/Hardware used for the development

1.8.1 Software

1.8.1.1 Programming languages: **Dart, JavaScript.**
1.8.1.2 Libraries and frameworks: **Flutter, Nodejs, Express.js, etc**.
1.8.1.3 Applications: **Visual Studio Code, Android Studio, Chrome, Postman.**
1.8.1.4 Database: **MongoDB**
1.8.1.5 Version: **MongoDB**

1.8.2 Hardware

- Laptop with processor i3 or higher, any operating system, with at least 4GB of RAM, and minimal storage of Hard Disk Drive or Solid-State Drive.
- Mobile Device

1.9 Software/Hardware required for the implementation

1.9.1 Software:

- Visual Studio Code
- GitHub
- Azure Cloud Services
- Google Play console
- MongoDB Atlas

1.9.2 Hardware

- Virtual Private Cloud with processor i3 or higher, any operating system, with at least 4GB of RAM, and minimal storage of Hard Disk Drive or Solid-State Drive
- Mobile Device

2. SRS

2.1 Introduction (Brief write-up about SRS)

The Software Requirements Specification (SRS) outlines what the software should do and how it should be expected to perform. The specification also outlines the functionality required by the product to meet all the needs of its users.

2.2 Overall Description

This section describes general factors that affect the product, but the specific requirements are not discussed, but the general overview is presented to aid in understanding the specific requirements.

2.2.1 Product perspective

This project provides an effective way to handle all the beauty & well-being needs of the customer. Helps the customer to easily find out the required services in less time.

The project is intended to reduce the time required to avail of beauty services. It helps customers to easily book the service on time as per their requirements early and helps the shop owners effectively manage their customers. Enabling holding of the customer and providing ease of service to the customer. This website relay on the shop owners to get the data and there is no external data required.

2.2.2 Product Functions

A general abstract description of a function to be performed by a product is given in the product function. This system contains modules like User Registration & login. User verification, password recovery, and user profile for handing the User. Shop Registration, adding services, and managing the shop information in the shop modules. The appointment module handles appointments by the users and to the shops. Users can review the shop and services. Admin has access to everything.

2.2.3 User characteristics

The user needs to have the basic computer knowledge to operate the system. The types of users and their characteristics are

- **Admin**

  The admin has a high level of authority compared to other users. He has the authority to change the passwords and to create a new admin. Oversee all the data flow in the system. Perform necessary action to the data if any kind of malfunction happens in the system.

- **Shop Owners**

  The shop owner should have basic knowledge of working with mobile devices. And should be respond to the client's requests on time.

- **Users**

  Users can find a service and book an appointment and provide feedback/reviews.

2.2.4 General constraints

- GUI is only in English.
- Requires all users' registration.
- Requires all mandatory fields to be filled with valid information.
- Minimum capacity to run the application.

2.2.5 Assumptions

- Users should be familiar with using mobile applications.
- The information provided by the shop is assumed to be genuine and trustworthy.
- The user has an active internet connection.

2.3 Special Requirements

Not applicable for now

2.4 Functional requirements

2.4.1 Authentication

2.4.1.1 Registration

2.4.1.1.1 Email - String
2.4.1.1.2 Name - String
2.4.1.1.3 Phone - Number
2.4.1.1.4 Password - String

2.4.1.2 Login

2.4.1.2.1 Email - String
2.4.1.2.2 Password - String

2.4.1.3 Forget Password

2.4.1.3.1 Email - String
2.4.1.3.2 OTP - String
2.4.1.3.3 New Password - String
2.4.1.3.4 Confirm Password - String

2.4.2 Admin

2.4.2.1 Manage Categories

2.4.2.1.1 Create or Update Categories

2.4.2.1.1.1 Categories Name - String
2.4.2.1.1.2 Photo - File
2.4.2.1.1.3 Description - String

2.4.2.2 Manage Shop

2.4.2.2.1 Update Shop Info

2.4.2.2.1.1 Shop Name - String
2.4.2.2.1.2 Categories - List
2.4.2.2.1.3 Shop Id - String
2.4.2.2.1.4 Type - String
2.4.2.2.1.5 About Shop - String

2.4.2.2.2 Update Shop Contact

2.4.2.2.2.1 Email address - String
2.4.2.2.2.2 Phone - Number
2.4.2.2.2.3 Address - String
2.4.2.2.2.4 Pin Code - Number
2.4.2.2.2.5 Whatsapp - Number
2.4.2.2.2.6 Website - String
2.4.2.2.2.7 Facebook - String
2.4.2.2.2.8 Instagram - String
2.4.2.2.2.9 Twitter - String

2.4.3 Shop

2.4.3.1 Register Shop.

2.4.3.1.1 Shop Id - String
2.4.3.1.2 Email - String
2.4.3.1.3 Phone - Number
2.4.3.1.4 Pin Code - Number
2.4.3.1.5 Address - String

2.4.3.2 Update Shop Info

2.4.3.2.1 Shop Name - String
2.4.3.2.2 Categories - List
2.4.3.2.3 Shop Id - String
2.4.3.2.4 Type - String
2.4.3.2.5 About Shop - String

2.4.3.3 Update Shop Contact

2.4.3.31 Email address - String
2.4.3.3.2 Phone - Number
2.4.3.3.3 Address - String
2.4.3.3.4 Pin Code -Number
2.4.3.3.5 Whatsapp - Number
2.4.3.3.6 Website - String
2.4.3.3.7 Facebook - String
2.4.3.3.8 Instagram - String
2.4.3.3.9 Twitter - String

2.4.3.4 Add or Update Service

2.4.3.4.1 Service Name - Name
2.4.3.4.2 Price - Number
2.4.3.4.3 Time - Number
2.4.3.4.4 Category - List
2.4.3.4.5 Description - String
2.4.3.4.6 Photo - File

2.4.4 User

2.4.4.1 Edit or Update Profile.

2.4.4.1.1 Name - String
2.4.4.1.2 Photo - File
2.4.4.1.3 Phone - Number
2.4.4.1.4 City - String
2.4.4.1.5 Pin Code - Number
2.4.4.1.6 Date of Birth - Date
2.4.4.1.7 Gender - String

2.5 Design Constraints

2.5.1 Hardware Constraint

- RAM- 4GB
- CPU- i3 or more, or any modern processing unit
- Storage- minimum of 1GB or more. Hard disk drive or Solid-state drive
- Mobile Device

2.5.2 Software Constraint

- OS- any operating system
- Visual studio code
- Flutter
- Node JS.
- MongoDB Compass

2.5.3 Fault Tolerance

- Fault tolerance requirements can place a major constraint on how the system is to be designed.
- Fault tolerance requirements often make the system more complex and expensive, so they should be minimized
- To make the system fault tolerant exception handling, validation, and verification features are implemented.

2.5.4 Security

- Currently security requirements have become essential and major for all types of systems.

- To make this system more secure at the level of sign-up authentication techniques are used by sending verification links to valid email IDs and at the time of sign-in passwords and email id are used as login credentials, these techniques avoid the unusual authentication to access data.

2.5.5 Standard Compliance

- To maintain the system standard, this product follows all the specific requirements.
- All types of standard reports, navigations, and naming conventions for all types of components and containers are included in this system.

2.6. System Attributes

- Availability:
  - Availability refers to the percentage of time that the infrastructure, system, or solution remains operational under normal circumstances to serve its intended purpose.
  - According to its definition, this application is ready to perform its function when it is needed by the user.

- Portability:
  - The portability of a computer program refers to its ability to run under another operating system without requiring major alterations.
  - The application is able to run on a variety of operating systems.

- Reliability:
  - Software Reliability means Operational reliability. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.
  - Output of this product is accurate with a desired time of duration, and can also handle different types of users.

- Maintainability:
  - This application source code is easy to read, understand, and easy for other developers to maintain and for further changes in the future.
- Scalability:
  - Scalability is the measure of a system's ability to increase or decrease in performance.
  - an Increased number of users can be easily handled.

2.7 Other Requirements

2.7.1 The safety requirements are as follows:

- The password should not be visible while typing
- The system should use a secure connection to connect API services

2.7.2 Security Requirements

- The user (customer or owner and admin) requires a username and password to login the system
- Password should be stored in an encrypted format.
- Any keys used for the API should be stored securely.

3. System Design (Functional Design)

3.1. Introduction (brief write-up about System Design)

• System design is the process of defining the architecture, module interfaces, and data for a system to satisfy specified requirements.
• The purpose of the design phase is to plan the solution to the problem specified by the required documents.
• This is the first step that moves from the problem domain to the solution domain.
• The design of the system is essentially a blueprint or a plan for a solution for the system.

3.2 Assumptions and Constraints

An assumption is a condition you think to be true, and a constraint is a fixed limitation of project development.

- All the functional requirements collected from the client are sufficient for the project life cycle.
- All the non-functional and specific requirements specified in SRS are well enough for the development of the system.
- Time constraint.

3.3. Functional decomposition

This is the process of taking a complex process and breaking it down into smaller, simpler parts. Using functional decomposition large or complex functionalities are more easily understood. It is mainly used during the project analysis phase, so each phase can be viewed as software. So, this has modular with some sub-modules.

3.4 Description of Programs

3.4.1 Context Flow Diagram (CFD)

In CFD entire system is considered as single process. It shows input and outputs of the system. It shows all the external entities that interact with the system and how the data flows between these external entities and system.

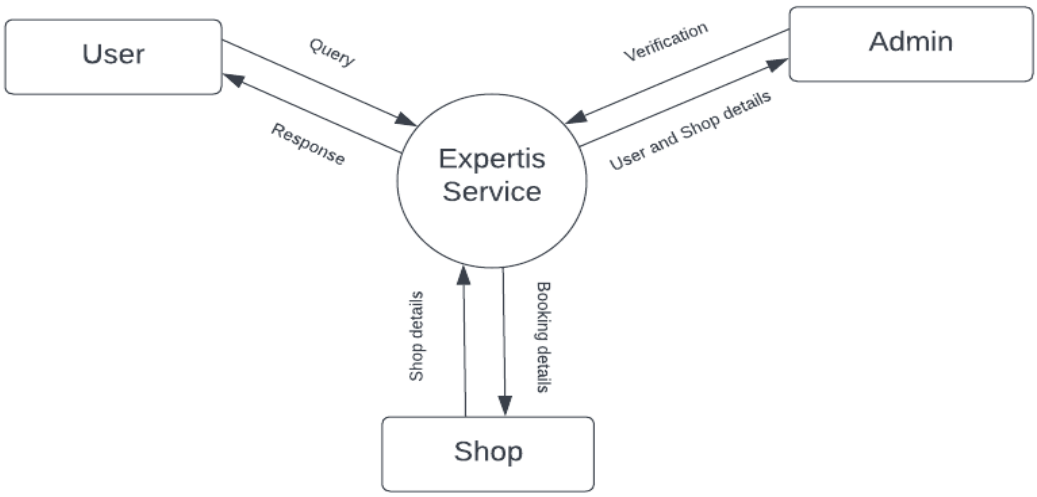


Fig.No: 3.1 - System Architecture

3.4.2 Data Flow Diagrams (DFDs – Level 0, Level 1, Level 2)

Data flow diagram shows the flow of data through system. Data flow diagrams also called the data flow graphs. It views a system as a function that transforms the inputs into desired outputs. It aims to capture the transformation that taken place within a system to the input data so that eventually the out data is produced.

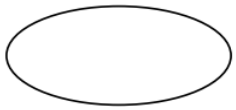
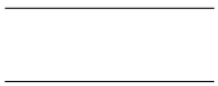
Symbols	Name	Description
	Process	It performs transformation of data from one state to another.
	Source/Sink	It represents the external entity that may be either source or sink.
	Flow of Data	It represents the flow of data from source to destination.
	Data Source/Data Storage	It is the place where data is stored.

Fig.No: 3.2 – Data flow Diagram

3.4.2.1 Level 1 (Admin)

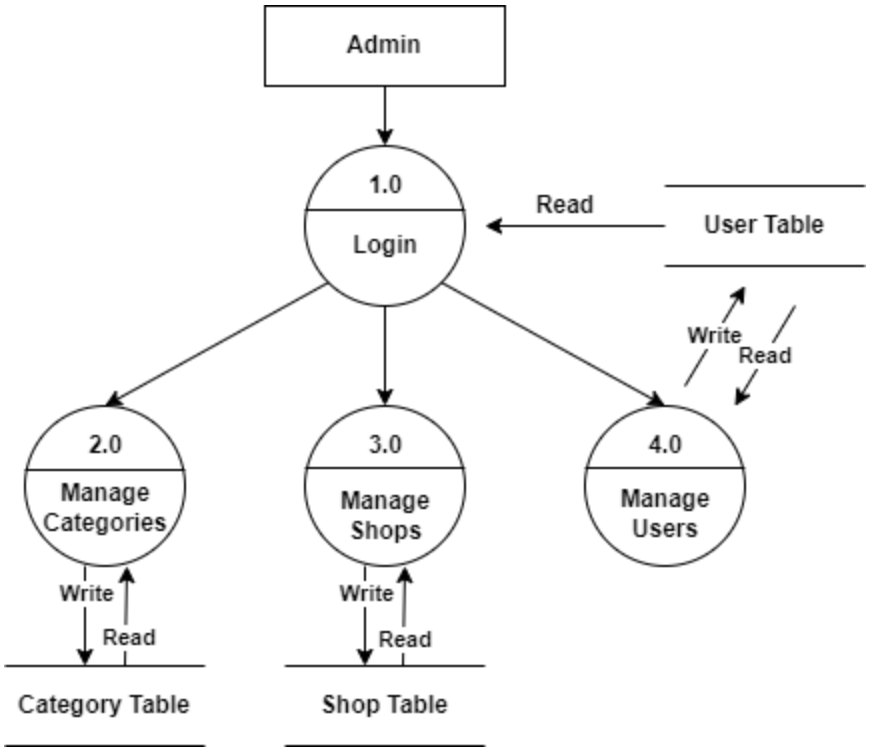


Fig.No: 3.3 – Admin DFD - level 1

3.4.2.1.1 Level 2 (Admin)

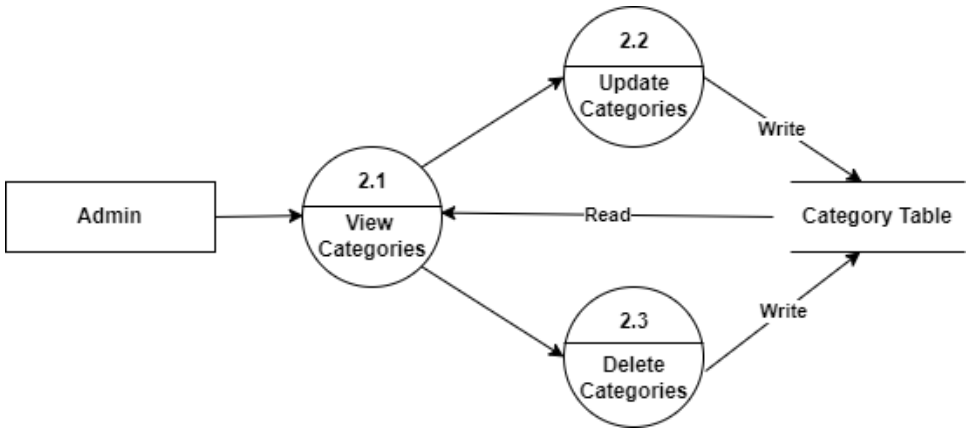


Fig.No: 3.4 – Admin manage category DFD

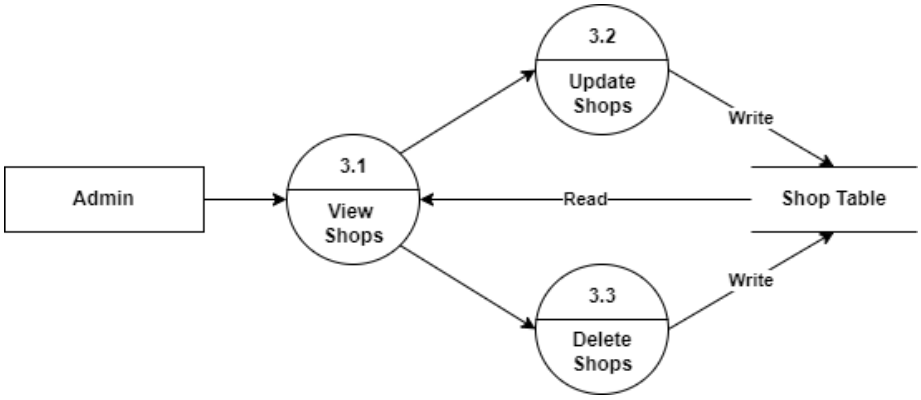


Fig.No: 3.5 – Admin manage shop DFD

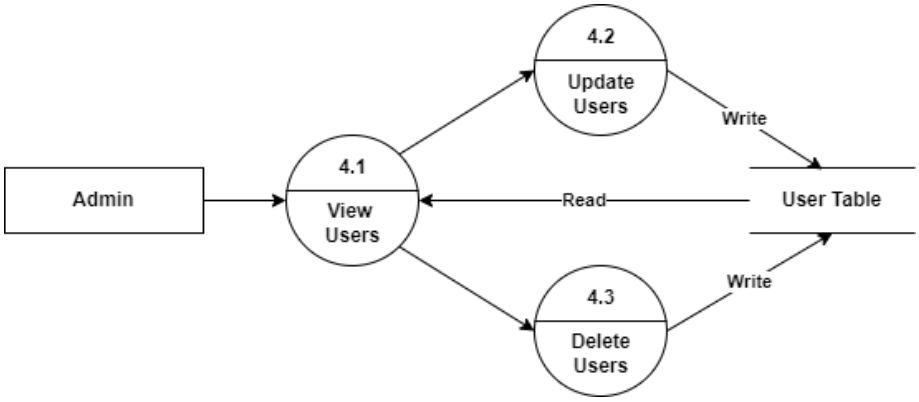


Fig.No: 3.6 – Admin manage users category DFD

3.4.2.2 Level 1 (Shop)

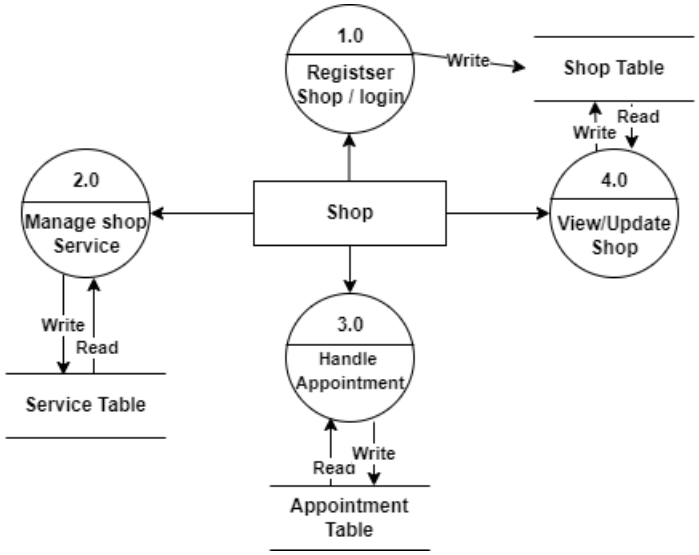


Fig.No: 3.7 – Shop DFD Level 1

3.4.2.2.1 Level 2 (Shop)

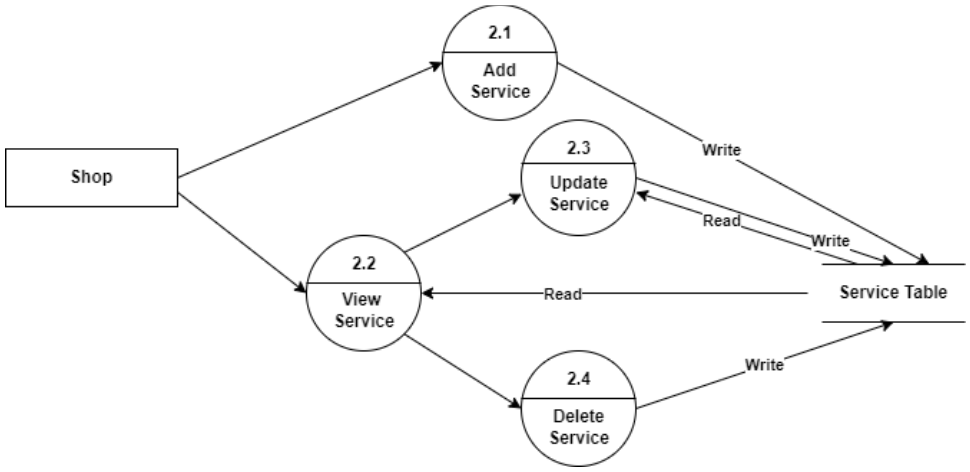


Fig.No: 3.8 – Shop Manage service DFD

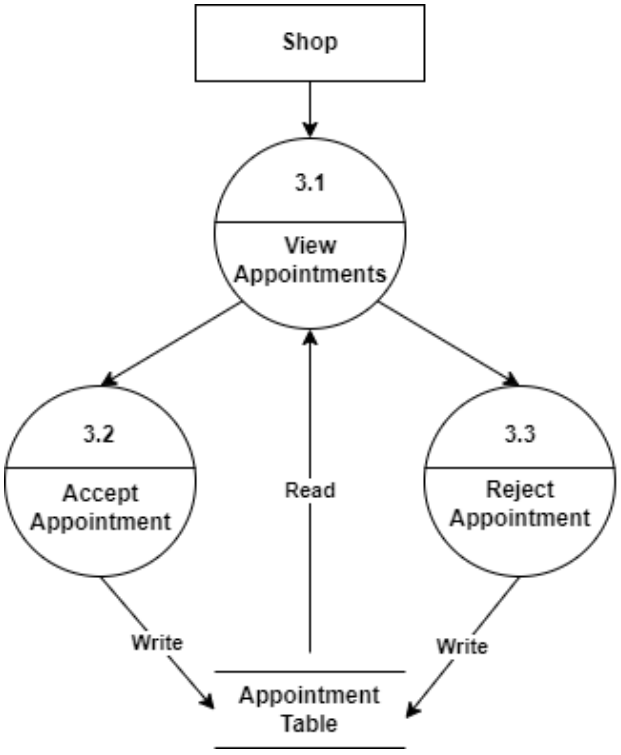


Fig.No: 3.9 – Shop Manage appointments DFD

3.4.2.3 Level 1 (User)

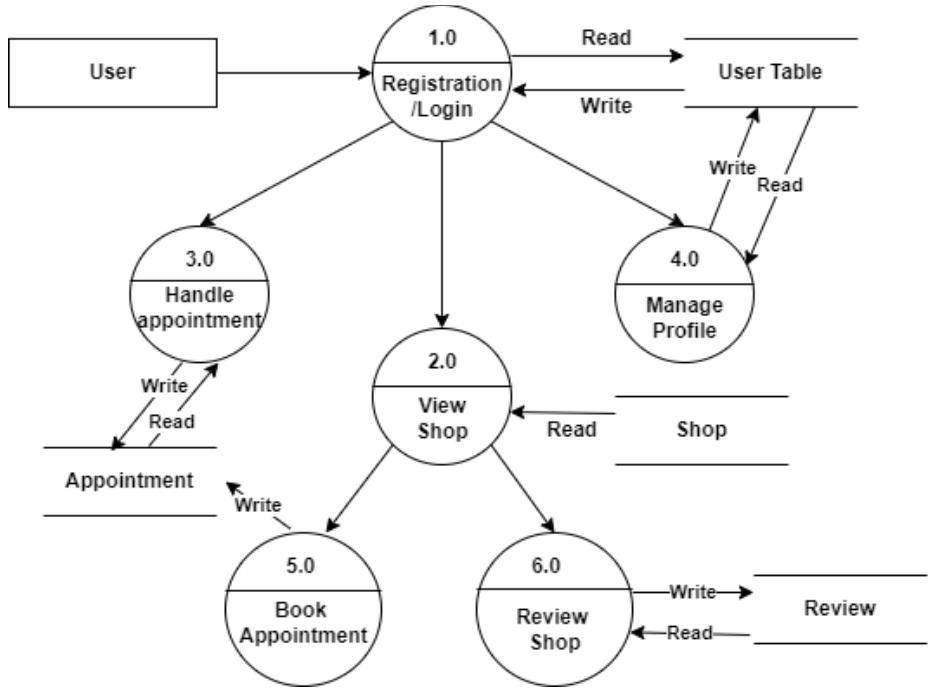


Fig.No: 3.10 – User DFD Level 1

3.4.2.3.1 Level 2 (User)

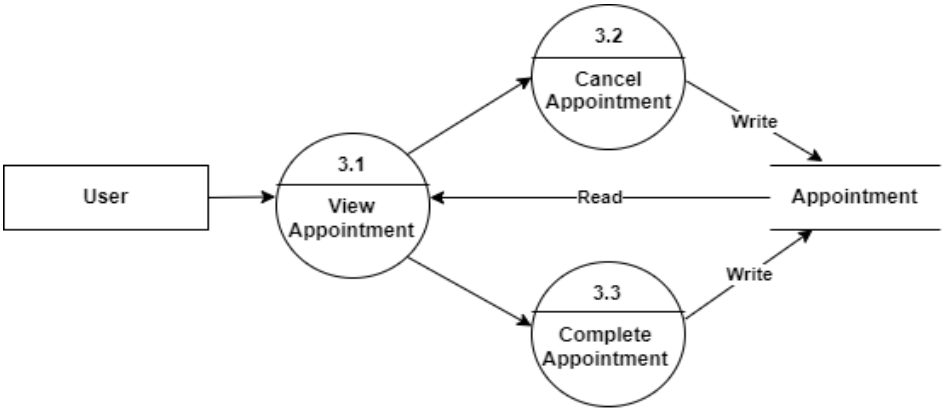


Fig.No: 3.10 – User handle appointment DFD

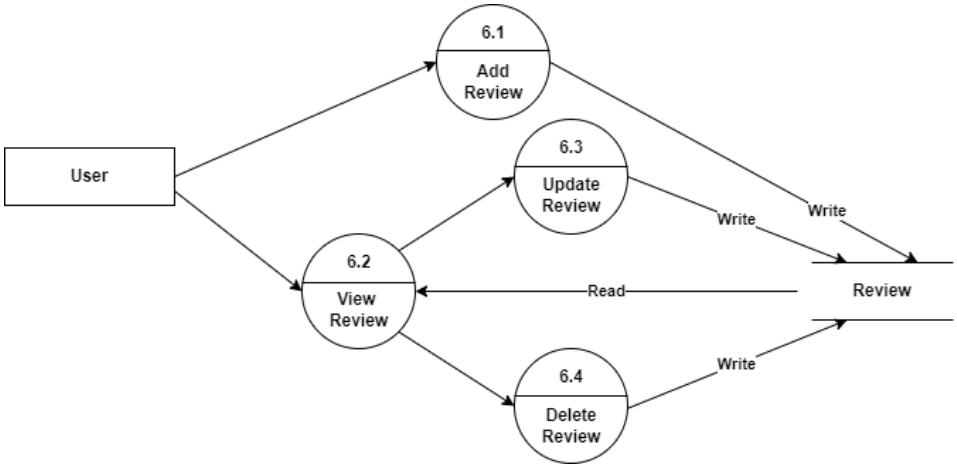


Fig.No: 3.11 – User give review DFD

3.5 Description of components

3.5.1 Registration/log in:

This module allows the user who is new to register to the software. & login who allow already registered and also can reset their password if they forget

3.5.2 Admin Module:

This is the master who can add categories. He has complete control over this application and can edit or can delete the details of the user and the shop.

3.5.3 Shop Module:

Shop owners can register here and add their services with the price and time so that they can get the appointment request from the customers who were registered here as a user.

3.5.4 User Module:

The user module is the main component of the application. In which users can see all the top-rated and nearby shops and also they can book an appointment and make a review

4. Database Design (or Data structure)

4.1 Introduction (brief write-up about Database design)

Introduction

Database is a collection of related data. Relational database stores data in a table or relations. The data stored in a relation are arranged in records. Each record consists of a set of attributes. Fields can be referred to as characteristics of records. This document describes the table that is used to design software, its attributes, data type, constraints, and relationship among those tables.

Through MongoDB Compass, this application uses MongoDB as a database design platform. Database System based on NoSQL.

Definition for NoSQL(Not only SQL):

NoSQL databases (Not only SQL) are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.

Definition for MongoDB:

MongoDB is a document database with the scalability and flexibility you want with the querying and indexing you need.

Definition for MongoDB Compass:

MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment.

The design process consists of the following steps in MongoDB:

- Design your schema according to user requirements.
- Combine objects into one document if you will use them together. Otherwise, separate them (but make sure there should not be a need for joins).
- Duplicate the data (but limited) because disk space is cheap as compared to compute time.
- Do joins while writing, not on reading.
- Optimize your schema for the most frequent use cases.
- Do complex aggregation in the schema.

4.2 Purpose and scope

4.2.1 Purpose

- **Avoid Redundantly**

  The table in the database should be constructed following standards and with utmost dedication. It should have different fields and minimize redundant data. The table should always have a unique id.

- **Faultless Information**

  The database should follow the standards and conventions and provide meaningful information useful to the organization.

- **Data Integrity**

  Integrity assists in guaranteeing that the values are valid and faultless. Data Integrity is set to tables, relationships, etc.

- **Modify**

  The database developed should be worked upon with the conventions and standards so that it can be easily modified whenever the need arises.

4.2.2 Scope

- Normalization of Database.
- Establishing the Relation between the tables.
- Accessing the data from multiple tables.

4.3 Database Identification

The identification of a database by a unique name given to the various database objects. The identifier is the name of the database object. The following are the various database objects.

- Tables
- Columns
- Views
- Sequences
- Indexes
- Stored Procedures

4.4 Schema information

Schema design for NoSQL usually involves designing Keys, Indexes & Denormalization of attributes, all of which are inter-dependent on the application queries & workflows. The query requirements elicitation should include the following specifications at a minimum: Business Data Entities.

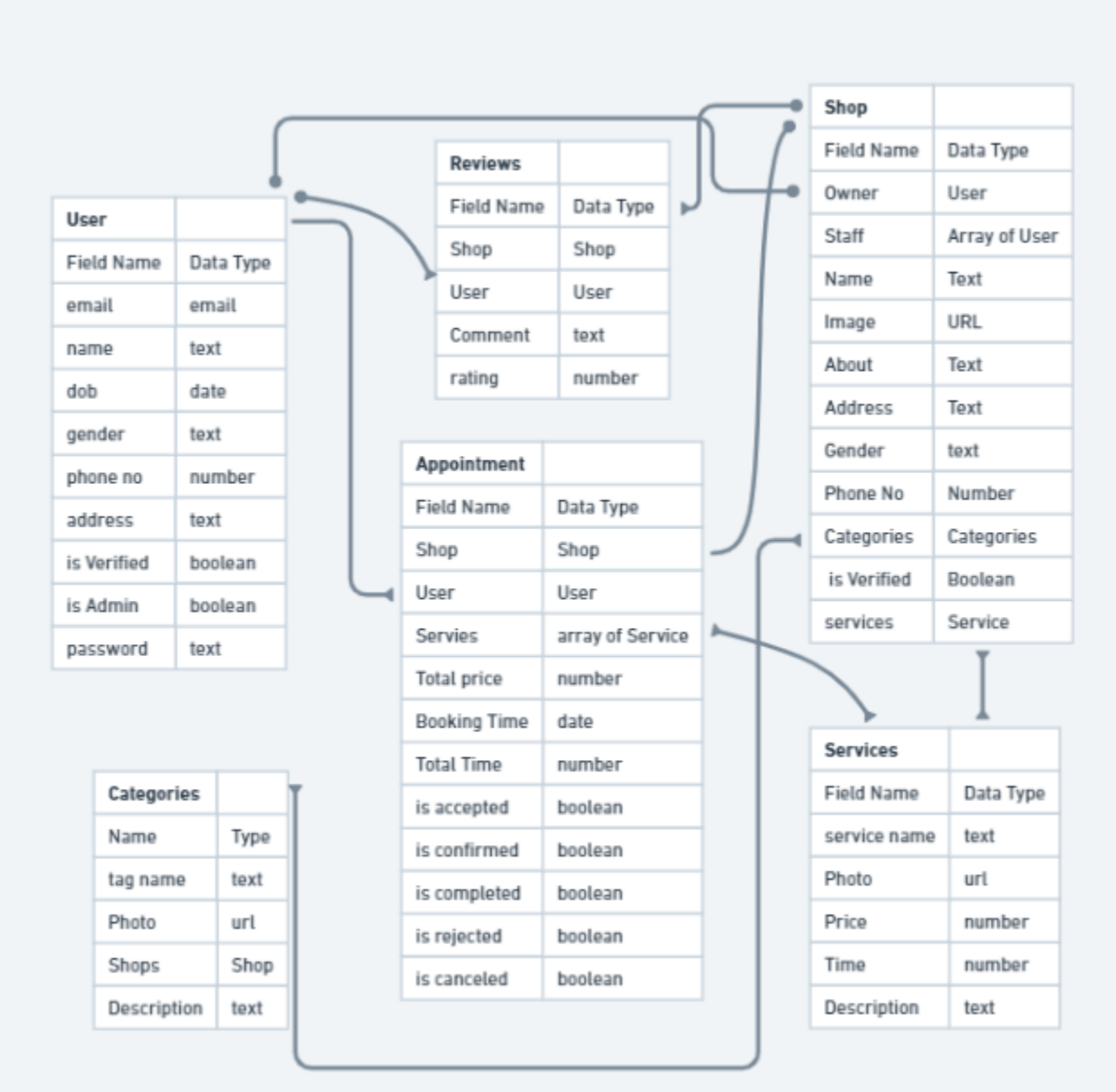


Fig.No: 4.1 – Database Schema

4.5 Table Definition

Instead of tables, a MongoDB database stores its data in collections. A collection holds one or more BSON documents. Documents are analogous to records or rows in a relational database table. Each document has one or more fields; fields are similar to the columns in a relational database table

4.5.1 User table

NO	Column Name	Data Type	Description	Required
4.5.1.1	Email	String	Email id of the user as primary key	True
4.5.1.2	Name	String	Name of the user	True
4.5.1.3	Phone	Number	Phone number of the user	True
4.5.1.4	DOB	Date	Date of Birth of user	False
4.5.1.5	Gender	String	Gender of the user	False
4.5.1.6	Roles	Role Schema	Roles of the user	True
4.5.1.7	PinCode	Number	Pin code of the user address	False
4.5.1.8	UserPic	String	Location path of the user pic	False
4.5.1.9	Password	String	Encrypted password	True
4.5.1.10	Verified	Boolean	To check is user email verified by default false	False
4.5.1.11	FavoriteShop	List	It contains the list of favorite shop id	False
4.5.1.12	Shop	List	It contains the shop id of the user	False
4.5.1.13	Appointment	List	It contains the appointment id of the user	False

4.5.1.6.1 Roles Schema

NO	Column Name	Data Type	Description	Required
4.5.1.6.1.1	IsAdmin	Boolean	To check he is admin by default false	False
4.5.1.6.1.2	IsUser	Boolean	To check he is a user by default true	False
4.5.1.6.1.3	IsShopOwner	Boolean	To check he is shop owner by default false	False
4.5.1.6.1.4	IsShopMember	Boolean	To check he is shop member by default false	False

4.5.2 Category table

No	Column Name	Data Type	Description	Required
4.5.2.1	CategoryName	String	Name of the category	True
4.5.2.2	CategoryPic	String	Picture location path of the category	False

4.5.2.3	Description	String	Description of the category	True
4.5.2.4	Shops	List	Shop id who were added this category to their shops	False
4.5.2.5	Services	List	Service id who were added this category to their services	False

4.5.3 Shop Table

No	Column Name	Data Type	Description	Required
4.5.3.1	Owner	String	User id of the shop owner	True
4.5.3.2	ShopId	String	It should be a unique as a primary key	True
4.5.3.3	ShopName	String	Name of the shop	True
4.5.3.4	ShopLogo	String	Picture location path of the shop logo	False
4.5.3.5	About	String	It is the description about the shop	False
4.5.3.6	Gender	String	Type of service provided to the gender	True
4.5.3.7	Contact	Contact Schema	It is table contains shop contacts	True
4.5.3.8	Likes	List	It contains user id who were added this shop to favorite list	False
4.5.3.9	Category	List	It contains the category id	False
4.5.3.10	Members	List	It contains the user id of the shop members	False
4.5.3.11	Services	List	It contains the services id of the shop	False
4.5.3.12	Appointments	List	It contains the appointment id of the shop	False
4.5.3.13	Rating	Double	Average rating of the shop	False
4.5.3.14	Reviews	List	It contains the review id of the shop	False
4.5.3.15	IsActive	Boolean	Is the shop active from long time	True
4.5.3.16	IsOpen	Boolean	Is shop open or not	True

4.5.3.7.1 Contact Schema.

No	Column Name	Data Type	Description	Required
4.5.3.7.1.1	Email	String	Email id of the shop owner	True

4.5.3.7.1.2	Website	String	URL of the shop website	False
4.5.3.7.1.3	Phone	Number	Phone number of the shop	True
4.5.3.7.1.4	Address	String	Address of the shop	True
4.5.3.7.1.5	PinCode	Number	Pin code of the shop address	True
4.5.3.7.1.6	Whatsapp	String	Whatsapp number of the shop	False
4.5.3.7.1.7	Instagram	String	Instagram profile URL of the shop	False
4.5.3.7.1.8	Twitter	String	Twitter profile link of the shop	False

4.5.4 Service table

No	Column Name	Data Type	Description	Required
4.5.4.1	ServiceName	String	Name of the service	True
4.5.4.2	Price	Number	Price of the service	False
4.5.4.3	Photo	String	Picture location path of the service	False
4.5.4.4	Time	String	Time duration require for the service	True
4.5.4..5	Description	String	Description about the service	True
4.5.4.6	Category	List	It contains the category id of the service	True

4.5.5 Appointment table

No	Column Name	Data Type	Description	Required
4.5.5.1	ShopId	String	Object id of the shop	True
4.5.5.2	MemberId	String	Id of the shop member	True
4.5.5.3	UserId	String	Id of the user	True
4.5.5.4	TotalPrice	Number	Total price of the services	True
4.5.5.5	TotalTime	Number	Total time required to complete the service	True
4.5.5.6	Services	List	Id of the services which is required	True
4.5.5.7	StartTime	Date	Start time of the service	True
4.5.5.8	EndTime	Date	End time of the service	True
4.5.5.9	AppointmentStatus	String	A status of the appointment by default pending	true

4.5.6 Review table

No	Column Name	Data Type	Description	Required
4.5.6.1	From	String	Id of the user who reviewed	True
4.5.6.2	To	String	Id of the shop	True
4.5.6.3	Model_Type	String	It contains type of review (shop or Service)	True
4.5.6.4	Photo	String	Location path of the photo	True
4.5.6.5	Comment	String	Comment about the review	False
4.5.6.6	Title	String	Title about the review	True
4.5.6.7	Rating	String	Rating about the service or shop	False

4.6 Physical design

The physical design is where you translate schemas into actual database structures

- Entity to collections
- Objects to fields
- ObjectID (Unique)

4.7 Data Dictionary

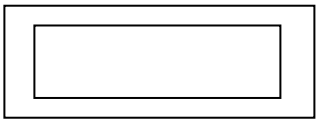
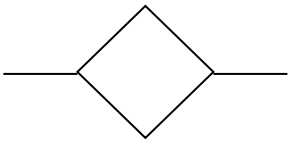
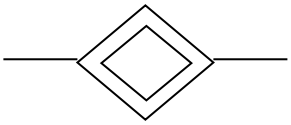
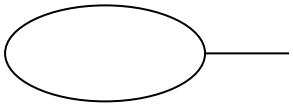
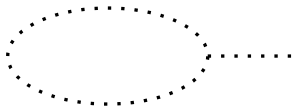
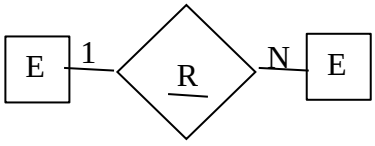
A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary holds records about other objects in the database, such as data ownership, data relationship to other objects, and other data.

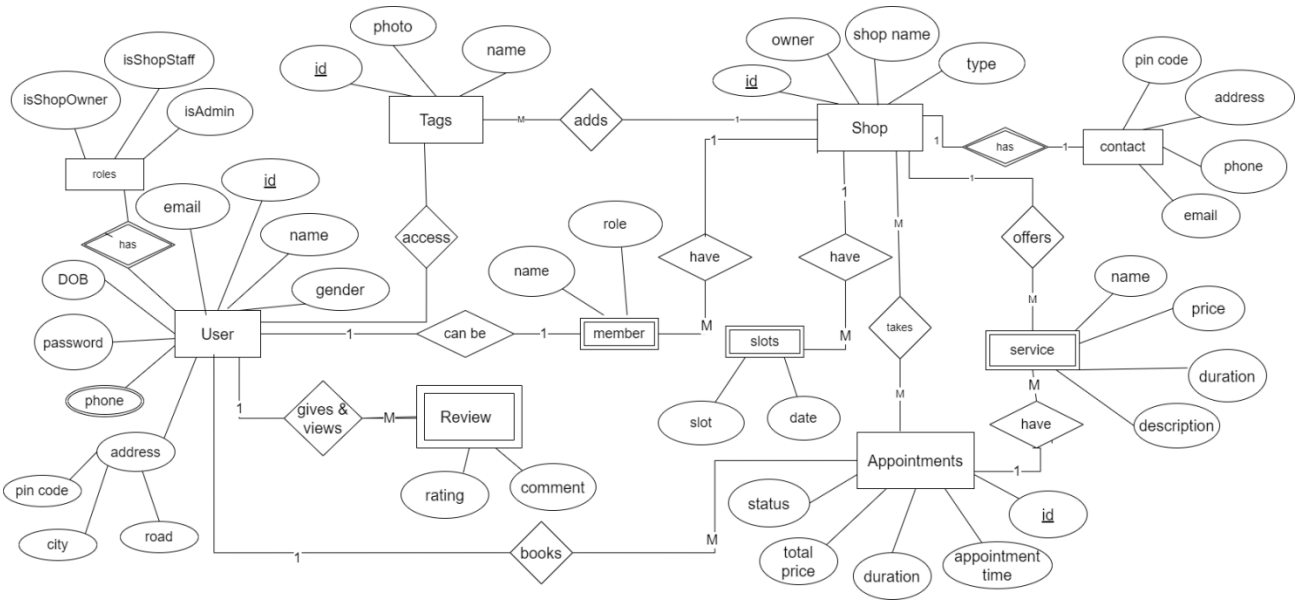
The data dictionary, in general, includes information about the following:

- Name of the data item
- Aliases
- Description/Purpose
- Related data items
- Range of values
- Data structure definition

4.8. ER diagram

ER- modelling is a data modelling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modelling method are called Entity-Relationship Diagrams or ER-diagrams or ERDs.

Symbols	Conversion
	Entity
	Weak entity
	Relation
	Identity Relation
	Attribute
	Derived Attribute
	Cardinality ratio 1: N from E1:E2 in R.



4.9 Database Administration

4.9.1 System information

A database is an organized collection of structured information, or data typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

In our case “MongoDB” is DBMS.

4.9.2 DBMS configuration

To configure DBMS through MongoDB compass is:

- Install Node JS
- Install the MongoDB Node JS Drive
- Create a MongoDB Atlas cluster and load the sample data.
- Next, you'll need a MongoDB database.
- Get your cluster's connection info.
- Import MongoClient.
- Create main function.
- List the databases in cluster.
- Save Your File.

4.9.3 Support software required

"MongoDB compass"

Software Requirements:

The following operating system are officially supported:

- Windows 7 (64-bit, Professional level or higher)
- Mac OS X 10.6.1+
- Ubuntu 9.10 (64bit)
- Ubuntu 8.04 (32bit/64bit)

4.9.4 Storage requirements

The minimum hardware requirements are:

- Hard Disk:1 GB Required 500GB(Recommended)
- CPU: Intel Core or Xeon 3GHz (or Dual Core 2GHz) or equal AMD CPU
- Cores: Single (Dual/Quad Core is recommended)
- RAM: 4 GB (6 GB recommended)
- Graphic Accelerators: NVidia or ATI with support of OpenGL 1.5 or higher
- Display Resolution: 1280×1024 is recommended, 1024×768 is minimum.

4.9.5 Backup and recovery

To back up our database

Step 1: Create Direct Backups Using Mongodump

You can run the **Mongodump** command using the below syntax from the system command line as follows:

Code:

mongodump <options> <connection-string>

Step 2: Backup a Remote MongoDB Instance

As mentioned in Step 1, you can customize a host and a port number with the **–uri** connection string using the following syntax:

Code:

mongodump --uri="mongodb://<host URL/IP>:<Port>" [additional options]

Step 3: Backup a Secure MongoDB Instance

MongoDB's Mongodump command allows you to implement access control mechanisms for your data backups. This will require you to provide a Username, Password, and specific Authentication options in the following syntax:

Code:

mongodump --authenticationDatabase=<Database> -u=<Username> -p=<Password> [additional options]

Step 4: Select Databases & Collections

In Step 3 you learned how to back up a remote database using Mongodump. This step involves the **–db** and **–collection** options which indicate a database and a collection that requires backing up. You can run the –db option in a standalone manner, but to execute a collection, you have to specify a database. Moreover, you can remove a collection from the backup process, by using the **–excludeCollection** option.

To select a particular database, use the following command:

Code:

mongodump --db=<Backup Target - Database> [additional options]

Step 5: Change the Backup Directory

You can leverage the **–out** option to specify the backup folder's location in the following way:

Code:

mongodump --out=<Directory Location> [additional options]

Step 6: Create an Archive File

The Mongodump utility provides you with a method to create an archive file. You can use the **–archive** option can for specifying the file to be archived. In case no specification is given, the output will be in the standard form (stdout).

Keep in mind that you can't use the **–archive** option along with the **–out** option.

Code:

mongodump --archive=<file> [additional options]

Step 7: Compress the MongoDB Backup

Now, since you know how to backup data using Mongodump, it's time to understand the process of compressing these files. You can use the **–gzip** option to compress the JSON and BSON files individually using the following command:

Code:

mongodump --gzip [additional options]

Step 8: Restore Database

Apart from excellent backups, MongoDB also provides you the facility to restore your data. The **Mongorestore** command will seamlessly load data from the Mongodumb backups and restore your Mongo database. The mongorestore

command, however, cannot overwrite documents present in the database if the id for the document already exists. Otherwise, Mongorestore will generate a new database or will add data to the existing one.

The Mongorestore command requires you to mention the path to your dump directory and the syntax is follows:

Code:

```
mongorestore dump/
```

To restore database

In order to restore data into a remote MongoDB instance, you need to establish the connection. The connection to a database can be specified using either:

- The URI connection string
- The host option
- The host and port option

1. Connecting using the URI option:

   Code:

   ```
   mongorestore [additional options] --uri="mongodb://<host URL/IP>:<Port>" [restore directory/file]
   ```

2. Connecting using the host option:

   Code:

   ```
   mongorestore [additional options] --host="<host URL/IP>:<Port>" [restore directory/file]
   ```

3. Connecting using host and port options:

   Code:

   ```
   mongorestore [additional options] --host="<host URL/IP>" --port=<Port> [restore directory/file]
   ```

5. Detailed Design (Logic design of modules)

5.1 Introduction (brief write-up about Database design)

During detailed design, the internal logic of each module specified in system design is decided. During this phase, further details of the modules are decided. The design of each of the modules is usually specified in a high-level description language which is independent of the language in which software eventually be implemented.

5.2 Structure of the software package (structure chart)

Structure chart:

A structure chart is a top-down modular design, consisting of squares representing different models in a system and lines. The structure chart shows how the program has been partitioned into manageable modules hierarchy and organization of those modules and communicational interface.

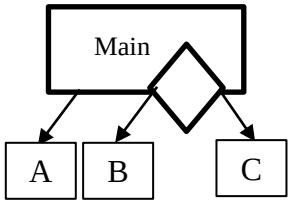
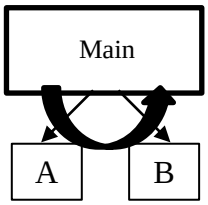
Symbol	Name	Process
	Data flow	Show the direction flow of data.
	Control flow	Shows the direction of flow control.
	Processing	Shows manipulation calculation and processing.
	Module Invocation	It represents a subordinate module being invoked by a superior ordinate module.
	Condition invocation	It indicates the invocation of subordinates. Modules depend on the evaluation of a condition.
	Iteration	It represents the iteration

Fig No: 5.1 – Structure chart

5.2.1 Admin

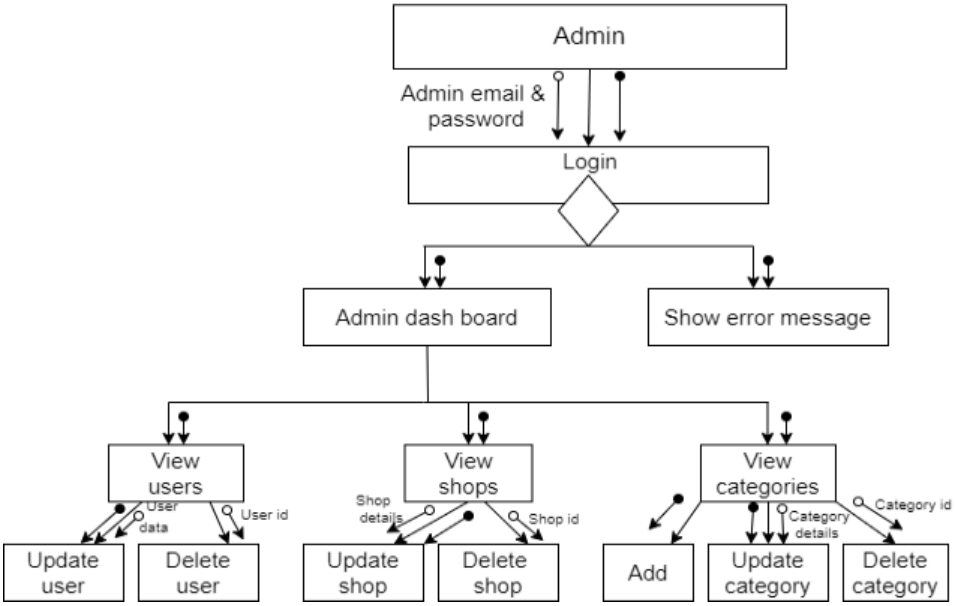


Fig No: 5.2 – Admin structure chart

5.2.2 Shop

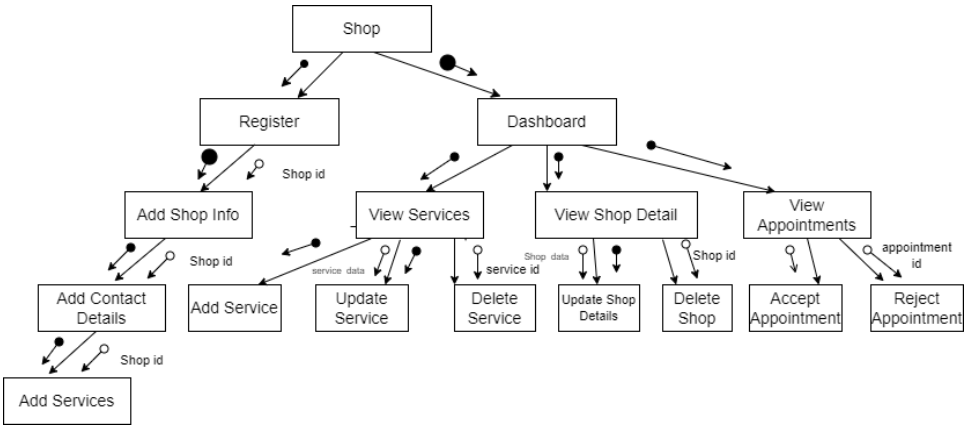


Fig No: 5.3 – Shop structure chart

5.2.2 User

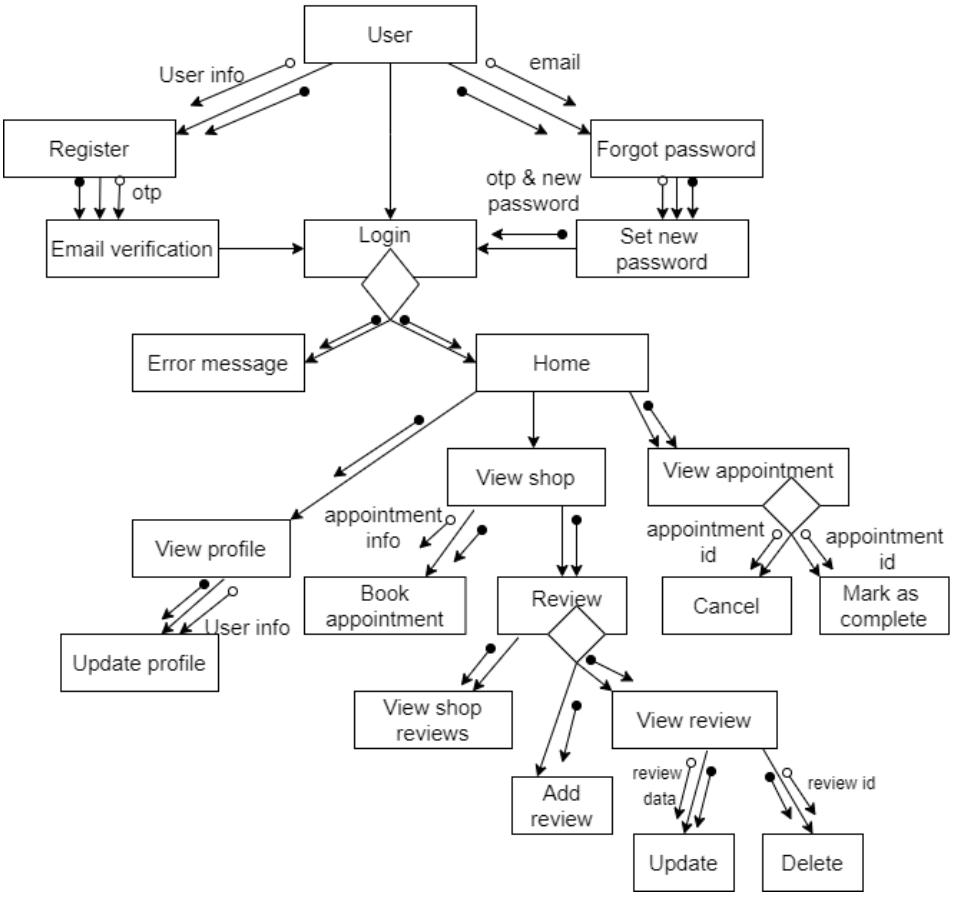


Fig No: 5.2 – User structure chart

5.3 Modular decomposition of the System

5.3.1 User Module

5.3.1.1 Signup/Register

5.3.1.1.1 Inputs

Email, Name, Phone, Password

5.3.1.1.2 Procedural details

Validate the input data and user information is saved in database.

5.3.1.1.3 File I/O interfaces

Saving the data in User table

5.3.1.1.4. Outputs

Message displayed. User registered successfully on valid data otherwise error messages.

5.3.1.1.5. Implementation aspects

from the frontend data is taken and request sent to API backend. Returned response will be showed.

5.3.1.2 Login

5.3.1.2.1 Inputs

Email, Password

5.3.1.2.2 Procedural details

- Check is user exist or not
- if exist check the password
- and then login the user
- send the response

5.3.1.2.3 File I/O interfaces

Not applicable

5.3.1.2.4. Outputs

Message displayed. User Registered successfully on valid data otherwise error messages

5.3.1.2.5. Implementation aspects

from the frontend data is taken and request sent to API backend. returned response will be showed.

5.3.1.3 Forget Password

5.3.1.3.1 Inputs

Email, OTP, new password

5.3.1.3.2 Procedural details

- send otp to the email sent by the user
- verify otp entered by the user
- set the new password for the user

5.3.1.3.3 File I/O interfaces

Saving the data in User table

5.3.1.3.4 Outputs

Message displayed. User Password reset successfully on valid data otherwise error messages

5.3.1.4 View Shops

5.3.1.4.1 Inputs

pin code (optional) , address (optional), gender (optional).

5.3.1.4.2 Procedural details

retrieve the shops based on rating and or any parameters given.

5.3.1.4.3 File I/O interfaces

Retrieve the data from Shops Table

5.3.1.4.4. Outputs

All the shops are listed so user can select any shops

5.3.1.4.4. Implementation aspects

not applicable

5.3.1.5 Book Appointment

5.3.1.5.1 Inputs

User, services, shop, member, date

5.3.1.5.2 Procedural details

Book the appointment to the user based on the given user

5.3.1.5.3 File I/O interfaces

Store the necessary data in User, Shop and Appointments table.

5.3.1.5.4 Outputs

Appointment booked successfully on booked otherwise error messages

5.3.1.5.5 Implementation aspects

Show the only available slots to book

5.3.1.6 Give reviews

5.3.1.6.1 Inputs

User, shop, rating, comment

5.3.1.6.2 Procedural details

If the user has not already given review then save the review other allow them to update the review.

5.3.1.6.3 File I/O interfaces

Store the necessary data in the Review table.

5.3.1.6.4 Outputs

Review saved successfully on success otherwise error messages.

5.3.1.6.5 Implementation aspects

Not applicable

5.3.2 Admin Module

5.3.2.1 View Categories

5.3.2.1.1 Inputs

not required

5.3.2.1.2 Procedural details

List all the categories in the app

5.3.2.1.3 File I/O interfaces

Retrieving the data in Category table

5.3.2.1.4 Outputs

List of categories

5.3.2.1.5 Implementation aspects

not applicable

5.3.2.2 Add or Update Categories

5.3.2.2.1 Inputs

Category name, category image, description

5.3.2.2.2 Procedural details

Add a new category or update the existing one

5.3.2.2.3 File I/O interfaces

Add or update the data in Category table

5.3.2.2.4 Outputs

Message displayed. category added or updated successfully on valid data otherwise error messages

5.3.2.3 View all shops

5.3.2.3.1 Inputs

Not required

5.3.2.3.2 Procedural details

List all the shops in the app

5.3.2.3.3 File I/O interfaces

Retrieving the data in Shop table

5.3.2.3.4 Outputs

List of shops

5.3.2.3.5 Implementation aspects

not applicable

5.3.2.4 Add or update shop information

5.3.2.4.1 Inputs

shop name, description, shop photo, adding categories

5.3.2.4.2 Procedural details

save shop information in corresponding shop

5.3.2.4.3 File I/O interfaces

Update User table

5.3.2.4.4 Outputs

Message displayed. Shop information Update successfully on valid data otherwise error messages

5.3.2.4.5 Implementation aspects

from the frontend data is taken and request sent to API backend. returned response will be showed.

5.3.2.5 Add or Update Shop Contact details

5.3.2.5.1 Inputs

Phone, address, pin code, social media links, and website

5.3.2.5.2 Procedural details

Find the shop an update the contact details

5.3.2.5.3 File I/O interfaces

Updating the data in Shop table

5.3.2.5.4 Outputs

Message displayed. shop updated successfully on valid data otherwise error messages

5.3.2.6 View Users

5.3.2.6.1 Inputs

not required

5.3.2.6.2 Procedural details

List all the users in the app

5.3.2.6.3 File I/O interfaces

Retrieving the data in User table

5.3.2.6.4 Outputs

List of users

5.3.2.6.5 Implementation aspects

Not applicable

5.3.2.7 Add or Update Categories

5.3.2.7.1 Inputs

user name, address, dob, profile pic, isAdmin, isShopOwner, isVerified, isShopMember

5.3.2.7.2 Procedural details

Update the existing user

5.3.2.7.3 File I/O interfaces

Add or update the data in User table

5.3.2.7.4 Outputs

Message displayed. user updated successfully on valid data otherwise error messages

5.3.3 Shop Module

5.3.3.1 Register

5.3.3.1.1 Inputs

Email, shop Name, Phone

5.3.3.1.2 Procedural details

Check if the shop with the same name and phone already exists. otherwise register the shop

5.3.3.1.3 File I/O interfaces

Saving the data in Shop table

5.3.3.1.4 Outputs

Message displayed. Shop registered successfully on valid data otherwise error messages

5.3.3.1.5 Implementation aspects

from the frontend data is taken and request sent to API backend. returned response will be showed.

5.3.3.2 Add or update shop information

5.3.3.2.1 Inputs

shop name, description, shop photo, adding categories

5.3.3.2.2 Procedural details

save shop information in corresponding shop

5.3.3.2.3 File I/O interfaces

Update User table

5.3.3.2.4 Outputs

Message displayed. Shop information Update successfully on valid data otherwise error messages

5.3.3.2.5 Implementation aspects

from the frontend data is taken and request sent to API backend. returned response will be showed.

5.3.3.3 Add or Update Shop Contact details

5.3.3.3.1 Inputs

Phone, address, pin code, social media links and website

5.3.3.3.2 Procedural details

Find the shop an update the contact details

5.3.3.3.3 File I/O interfaces

Updating the data in Shop table

5.3.3.3.4 Outputs

Message displayed. Shop updated successfully on valid data otherwise error messages

5.3.3.4 Add or Update Shop Services

5.3.3.4.1 Inputs

service name, Price, Duration, service image, tags

5.3.3.4.2 Procedural details

Find the shop and add or update the services

5.3.3.4.3 File I/O interfaces

Updating the data in Shop table

5.3.3.4.4 Outputs

Message displayed. Service added or updated successfully on valid data otherwise error messages

5.3.3.4 Accept or Reject Appointment

5.3.3.4.1 Inputs

Appointment id

5.3.3.4.2 Procedural details

update appointment status

5.3.3.4.3 File I/O interfaces

Update the data from Appointment Table

5.3.3.4.4 Outputs

Success message is shown on completion

5.3.3.4.5 Implementation aspects

Not applicable

6. Program code listing

6.1 Database connection

DB Connection

```
const database = database_url || dbConfig.db;
mongoose.Promise = global.Promise;
mongoose
  .connect(database, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(
    () => {
      console.log("Database connected");
    },
    (error) => {
      console.log("Database can't be connected: " + error);
    }
  );
```

6.2. Authorization / Authentication

6.2.1 Register

```
async function register(params, callback) {
  const { email } = params;
  // Check whether user already exists or not
  const user = await User.findOne({ email }).exec();
  // if user exist then return error
  if (user != null) {
    return callback({
      status: 400,
      message: "User already exists!",
    });
  }
  // if user does not exist then create new user
  const otp = otpGenerator.generate(6, {
    alphabets: false,
    upperCase: false,
    specialChars: false,
  });
  // Send OTP to user through email
  let msg =
    "Thank you for choosing Expertis. Use the following OTP to verify your account. OTP is valid for 5 minutes";
  sendOTPMail(email, otp, msg)
    .then((response) => {
      // Create new user in the database
      const user = new User(params);
      user
        .save()
        .then((response) => {
          const ttl = 5 * 60 * 1000; //5 Minutes in milliseconds
          const expires = Date.now() + ttl; //timestamp to 5 minutes in the future
          const data = `${user._id}.${otp}.${expires}`; // phone.otp.expiry_timestamp
          const hash = crypto
            .createHmac("sha256", key)
            .update(data)
            .digest("hex"); // creating SHA256 hash of the data
          const fullHash = `${hash}.${expires}`; // Hash.expires, format to send to the user
          // Send hash to user for verification
          return callback(null, {
            hash: fullHash,
            email: user.email,
            id: user._id,
          });
        })
        .catch((error) => {
          return callback(error);
        });
    })
    .catch((err) => {
      return callback({ status: 400, message: "Email can't be sent" });
    });
}
```

6.2.2 Verify OTP

```
async function verifyOTP(id, otp, hash, callback) {
  // Separate Hash value and expires from the hash returned from the user
  let [hashValue, expires] = hash.split(".");
  // Check if expiry time has passed
  let now = Date.now();
  if (now > parseInt(expires))
    return callback({ status: 400, message: "OTP Expired" });
  // Calculate new hash with the same key and the same algorithm
  let data = `${id}.${otp}.${expires}`;
  let newCalculatedHash = crypto
    .createHmac("sha256", key)
    .update(data)
    .digest("hex");
  // Match the hashes

  if (newCalculatedHash === hashValue) {
    // Make user verified
    let doc = await User.findByIdAndUpdate(
      id,
      { verified: true },
      { useFindAndModify: true, new: true }
    );
    if (!doc)
      callback(
        `Cannot update Profile with id=${id}. Maybe user was not found!`
      );
    else {
      const user = await User.findById(id);
      const token = auth.generateAccessToken({
        id: user._id,
        email: user.email,
        name: user.name,
        isAdmin: user.roles.isAdmin,
        isShopMember: user.roles.isShopMember,
        isShopOwner: user.roles.isShopOwner,
        isVerified: user.verified,
        dob: user.dob,
        phone: user.phone,
        gender: user.gender,
      });
      return callback(null, { ...doc.toJSON(), token });
    }
  } else {
    return callback("Invalid OTP");
  }
}
```

6.2.3 Login

```
async function login(params, callback) {
  const { email, password } = params;
  // Find user by email
  const user = await User.findOne({ email }).populate({
    path: "shop",
    populate: { path: "services" },
  });
  // If user found
  if (user != null) {
    // Check if password is correct
    if (bcrypt.compareSync(password, user.password)) {
      // If user is not verified send OTP to verify user
      if (user.verified == false) {
        const otp = otpGenerator.generate(6, {
          alphabets: false,
          upperCase: false,
          specialChars: false,
        });
        const ttl = 5 * 60 * 1000; //5 Minutes in miliseconds
        const expires = Date.now() + ttl; //timestamp to 5 minutes in the future
        const data = `${user._id}.${otp}.${expires}`; // phone.otp.expiry_timestamp
        const hash = crypto
          .createHmac("sha256", key)
          .update(data)
          .digest("hex"); // creating SHA256 hash of the data
        const fullHash = `${hash}.${expires}`; // Hash.expires, format to send to the user
        let msg =
          "Your email is not verified. Please verify your email by entering the OTP sent below. OTP is valid
for 5 minutes";
        sendOTPMail(email, otp, msg)
          .then((response) => {
            return callback({
              status: 300,
              data: {
                hash: fullHash,
                email: user.email,
                id: user._id,
                message: "Verify your email, OTP sent to your email",
              },
            });
          })
          .catch((err) => {
            return callback({ status: 400, message: "Email can't be sent" });
          });
      } else {
        // If user is verified, generate token and return it
        const token = auth.generateAccessToken({
          id: user._id,
          email: user.email,
          name: user.name,
          isAdmin: user.roles.isAdmin,
          isShopMember: user.roles.isShopMember,
          isShopOwner: user.roles.isShopOwner,
          isVerified: user.verified,
          dob: user.dob,
          phone: user.phone,
          gender: user.gender,
        });
        return callback(null, {
          ...user.toJSON(),
          token,
          message: "Login Successful",
        });
      }
    } else {
      return callback({
        status: 400,
        message: "Invalid Password!",
      });
    }
  } else {
    return callback({
      status: 400,
      message: "User does not exist!",
    });
  }
}
```

6.2.4 Forget Password

```
async function forgetPassword(email, callback) {
  const user = await User.findOne({ email });
  if (user != null) {
    const otp = otpGenerator.generate(6, {
      alphabets: false,
      upperCase: false,
      specialChars: false,
    });
    const ttl = 5 * 60 * 1000; //5 Minutes in miliseconds
    const expires = Date.now() + ttl; //timestamp to 5 minutes in the future
    const data = `${email}.${otp}.${expires}`; // phone.otp.expiry_timestamp
    const hash = crypto.createHmac("sha256", key).update(data).digest("hex"); // creating SHA256 hash of the
data
    const fullHash = `${hash}.${expires}`; // Hash.expires, format to send to the user
    let msg =
      "Thank you for choosing Expertis. Use the following OTP to reset password pocess. OTP is valid for 5
minutes";
    sendOTPMail(email, otp, msg)
      .then((par) => {
        return callback(null, { hash: fullHash, email: user.email });
      })
      .catch((e) => {
        return callback("Email Not Sent");
      });
  } else {
    return callback({
      message: "Email Not Registered",
    });
  }
}

```

6.2.5 Set New Password

```
async function changePassword(params, callback) {
  User.findOneAndUpdate({ email: params.email }, params, {
    useFindAndModify: true,
  })
    .then((response) => {
      if (!response)
        callback(
          `Cannot update Profile with id=${params.id}. Maybe user was not found!`
        );
      else callback(null, response);
    })
    .catch((error) => {
      return callback(error);
    });
}
```

6.2.6 Updated Profile

```
async function updateUser(userData, callback) {
  const userId = userData.id;

  let doc = await User.findByIdAndUpdate(userId, userData, {
    useFindAndModify: true,
    new: true,
  }).populate({ path: "shop", populate: { path: "services" } });
  if (!doc) {
    return callback({
      status: 400,
      message: "User does not exists",
    });
  }
  return callback(null, { ...doc.toJSON() });
}
```

6.2.7 Delete User

```
async function deleteUser(req, res, callback) {
  let { id } = req.params;
  const user = await User.findById(id);
  const req_user = await User.findById(req.user.id);
  // Check user authorization
  if (!user) {
    return res.status(404).send({ message: "User not found" });
  }
  if (user.id.toString() !== req.user.id) {
    if (req_user.roles.isAdmin === false) {
      return callback({ status: 400, message: "You are not authorized" });
    }
  }

  if (user.shop.length > 0) {
    const id = user.shop[0].toString();
    //delete the shop services
    await Services.deleteMany({ shop: id })
      .then(console.log("delete the services"))
      .catch((e) => {
        return callback(e);
      });
    //delete the shop reviews
    await Reviews.deleteMany({ to: id })
      .then(console.log("delete the shop review"))
      .catch((e) => {
        return callback(e);
      });
    //delete the shop appointments
    await Appointment.deleteMany({ shopId: id })
      .then(console.log("delete the appointments of the shop"))
      .catch((e) => {
        return callback(e);
      });
    //delete the shop SlotBooking
    await SlotBooking.deleteMany({ shopId: id })
      .then(console.log("delete the slot bookings of the shop"))
      .catch((e) => {
        return callback(e);
      });
    //delete the shop
    Shop.findByIdAndRemove(id)
      .then((response) => {
        console.log("delete the shop");
      })
      .catch((error) => {
        return callback(error);
      });
  }
  //delete the appointments of the user
  await Appointment.deleteMany({ userId: user.id.toString() })
    .then(console.log("delete appointments of the user"))
    .catch((error) => {
      return callback(error);
    });
  //delete the reviews of the user
  await Reviews.deleteMany({ from: user.id.toString() })
    .then(console.log("delete reviews of the user"))
    .catch((error) => {
      console.log(error);
    });
  //delete the user
  await user.remove();
  return callback(null, {
    status: 200,
    message: "User deleted successfully",
  });
}
```

6.3 Data store / retrieval / update

6.3.1 Admin

6.3.1.1 Categories

6.3.1.1.1 Create New Category

```
async function createTag(data, callback) {
  Tags.create(data)
    .then((res) => {
      return callback(null, res);
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.3.1.1.2 Update Category

```
async function updateTag(data, callback) {
  const { id } = data;
  await Tags.findByIdAndUpdate(id, data, {
    useFindAndModify: true,
    new: true,
  })
    .then((res) => {
      if (res == null || res == undefined) {
        return callback({ status: 404, message: "Tag not found!" });
      }
      return callback(null, res);
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.3.1.1.3 Get All Categories

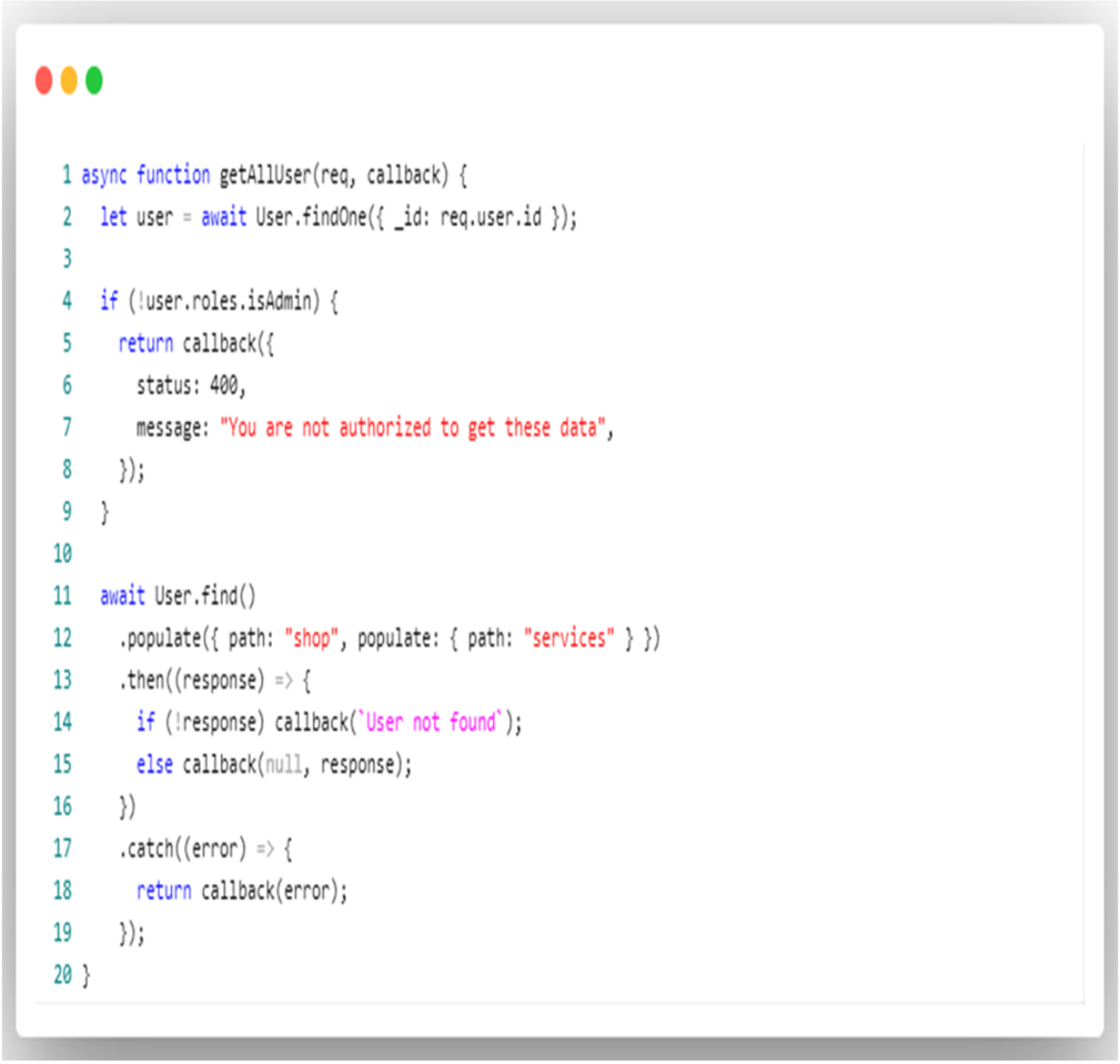
```
async function getTags(params, callback) {
  const tags = await Tags.find({});
  return callback(null, tags);
}
```

6.3.1.1.3 Delete Category

```
async function deleteTag(params, callback) {
  const { id } = params;
  const tags = await Tags.findByIdAndDelete(id);
  if (tags) {
    return callback(null, tags);
  } else {
    return callback({ status: 404, message: "Tag not found!" });
  }
}
```

6.3.1.2 Users

6.3.1.2.1 View all users



```
async function getAllUser(req, callback) {
  let user = await User.findOne({ _id: req.user.id });

  if (!user.roles.isAdmin) {
    return callback({
      status: 400,
      message: "You are not authorized to get these data",
    });
  }

  await User.find()
    .populate({ path: "shop", populate: { path: "services" } })
    .then((response) => {
      if (!response) callback(`User not found`);
      else callback(null, response);
    })
    .catch((error) => {
      return callback(error);
    });
}
```

Note: Admin have all the below functionalities of shop and user

6.3.2 Shop

6.3.2.1 Create Shop

```
async function createShop(shopData, callback) {
  try {
    const { owner } = shopData;
    const { email, phone } = shopData.contact;
    console.log(email, phone);
    const user = await User.findById(owner).exec();
    if (user == null) {
      return callback({
        status: 400,
        message: "Invalid User",
      });
    }
    // Set the shop owner as a member of the shop
    member = {
      member: user._id,
      role: "owner",
      name: user.name,
      pic: user.userPic,
    };
    shopData.members = [member];
    console.log(shopData);
    // Create the shop and save it in the database
    const shop = new Shop(shopData);
    shop
      .save()
      .then((response) => {
        // Add the shop to the user's shop list and update his role to be a shop owner
        User.findByIdAndUpdate(
          owner,
          {
            $set: {
              "roles.isShopOwner": true,
              "roles.isShopMember": true,
            },
            $push: {
              shop: response._id,
            },
          },
          { new: true }
        )
          .then((res) => {
            if (res == null) {
              return callback("Document not found");
            } else {
              return callback(null, response);
            }
          })
          .catch((err) => {
            return callback(err);
          });
      })
      .catch((error) => {
        if ("contact.phone" in error.errors) {
          return callback({
            status: 400,
            message: "Shop with phone number already exists",
          });
        }
        if ("shopId" in error.errors) {
          return callback({
            status: 400,
            message: "Shop id already in use",
          });
        }
        return callback(error);
      });
  } catch (error) {
    return callback(error);
  }
}
```

6.3.2.2 Update Shop

```
async function updateShop(shopData, callback) {
  const shopId = shopData.id;
  Shop.findByIdAndUpdate(shopId, shopData, {
    useFindAndModify: false,
    new: true,
  })
    .populate("services")
    .then((response) => {
      if (!response)
        callback(
          `Cannot update Shop with id=${shopId}. Maybe Shop was not found!`
        );
      else callback(null, response);
    })
    .catch((error) => {
      return callback(error);
    });
}
```

6.3.2.3 Services

6.3.2.3.1 Add Service

```
async function addService(params, callback) {
  const { id } = params;
  // Creating the service and saving it in the database
  Services.create({ ...params.service_data, shop: id })
    .then((document) => {
      // Add the service to the shop's services list
      Shop.findByIdAndUpdate(
        id,
        {
          $push: {
            services: document._id,
          },
        },
        { new: true }
      )
        .then((res) => {
          if (res == null) {
            return callback("Shop not found");
          } else {
            return callback(null, res);
          }
        })
        .catch((err) => {
          return callback(err);
        });
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.3.2.3.2 Update Service

```
async function updateService(params, callback) {
  const { id } = params;
  Services.findByIdAndUpdate(id, params.service_data, {
    useFindAndModify: true,
    new: true,
  })
    .then((response) => {
      if (!response)
        callback(
          `Cannot update service with id=${id}. Maybe service was not found!`
        );
      else callback(null, response);
    })
    .catch((error) => {
      return callback(error);
    });
}
```

6.3.2.3.3 Delete Service

```
async function deleteService(id, callback) {
  Services.findByIdAndRemove(id)
    .then(async (response) => {
      if (!response)
        callback(
          `Cannot delete Service with id=${id}. Maybe Service was not found!`
        );
      else {
        // delete the service from the shop's services list
        let updatedShop = await Shop.findByIdAndUpdate(
          response.shop,
          {
            $pull: { services: id },
          },
          { new: true }
        );
        console.log(updatedShop);

        callback(null, updatedShop);
      }
    })
    .catch((error) => {
      return callback(error);
    });
}
```

6.3.2.4 Appointments

6.3.2.4.1 View Shop Appointments

```
async function getShopAppointments(req, res, callback) {
  try {
    const { id } = req.params;
    if (!ObjectId.isValid(id)) {
      return callback({
        status: 400,
        message: "Invalid shop id",
      });
    }
    let filter = {
      shopId: id,
    };
    const { upcoming } = req.query;
    // if upcoming is true then only upcoming appointments are returned with the following appointment status
    let appointmentStatus = ["PENDING", "CONFIRMED", "ACCEPTED"];
    if (upcoming !== undefined && upcoming == "true") {
      filter.endTime = { $gt: new Date() };
      filter.appointmentStatus = { $in: appointmentStatus };
    }
    const appointments = await Appointment.find(filter)
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    return callback(null, appointments);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.2.4.2 Accept Appointment

accept_appointment

```
async function acceptAppointment(req, res, callback) {
  try {
    const { id } = req.params;
    const appointment = await Appointment.findById(id);
    if (!appointment) return callback("Appointment not found");
    // check if user is authorized to accept the appointment
    if (
      appointment.memberId.toString() !== req.user.id &&
      req.user.isAdmin == false
    ) {
      return callback("User not authorized");
    }
    // Update the appointment status to accepted
    const updatedAppointment = await Appointment.findByIdAndUpdate(
      { _id: id },
      {
        appointmentStatus: "ACCEPTED",
      },
      { new: true }
    )
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    if (!updatedAppointment) return callback("Operation failed");
    return callback(null, updatedAppointment);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.2.4.3 Reject Appointment

reject_appointment

```
async function rejectAppointment(req, res, callback) {
  try {
    const { id } = req.params;
    const appointment = await Appointment.findById(id);

    if (!appointment) return callback("Appointment not found");
    // check if user is authorized to reject the appointment
    if (
      appointment.memberId.toString() !== req.user.id &&
      req.user.isAdmin == false
    ) {
      return callback("User not authorized");
    }
    // Remover the appointment from slots booked
    const updateSlotsBooked = await SlotsBooked.findOneAndUpdate(
      {
        date: getDDMMMYYYYDate(appointment.startTime),
        shopId: appointment.shopId,
        memberId: appointment.memberId,
      },
      {
        $pullAll: {
          slots: appointment.slots,
        },
      },
      { new: true }
    );

    // Update the appointment status to rejected
    const updatedAppointment = await Appointment.findByIdAndUpdate(
      { _id: id },
      {
        appointmentStatus: "REJECTED",
      },
      { new: true }
    )
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    if (!updatedAppointment) return callback("Operation failed");
    return callback(null, updatedAppointment);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.2.5 Delete Shop

delete_shop

```
async function deleteShop(req, callback) {
  try {
    const id = req.params.id;
    let shop = await Shop.findById(id).exec();
    if (!shop) {
      return callback({
        status: 400,
        message: "Shop id does not exits",
      });
    }
    // Check is user is authorized to delete the shop
    if (req.user.id != shop.owner) {
      if (!req.user.isAdmin)
        return callback({
          status: 401,
          message: `You are not authorized to delete this shop`,
        });
    }
    //delete the shop services
    await Services.deleteMany({ shop: id });
    //delete the shop reviews
    await Reviews.deleteMany({ to: id });
    //delete the shop appointments
    await Appointments.deleteMany({ shopId: id });
    //delete the shop SlotBooking
    await SlotBooking.deleteMany({ shopId: id });
    //delete the shop
    Shop.findByIdAndRemove(id)
      .then(async (response) => {
        if (!response)
          return callback(
            `Cannot delete Shop with id=${id}. Maybe Shop was not found!`
          );
        // delete shop from user's shops list
        let updatedUser = await User.findByIdAndUpdate(shop.owner, {
          $pull: { shop: id },
        });
        return callback(null, updatedUser);
      })
      .catch((error) => {
        return callback(error);
      });
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3 User

6.3.3.1 Shops

6.3.3.1.1 View Shops Based on personal info and ratings

get_shops

```
async function getShops(req, callback) {
  try {
    // Create query object to hold search criteria
    let query;
    const pinCode = req.query.pinCode;
    let city = req.query.city;
    let gender = req.query.gender;
    let pattern = [];
    if (pinCode !== undefined && pinCode !== null) {
      pattern.push({ "contact.pinCode": pinCode });
    }
    if (gender !== undefined && gender !== null) {
      if (gender.toLowerCase() == "male")
        pattern.push({ gender: { $ne: "WOMEN" } });
      else if (gender.toLowerCase() == "female")
        pattern.push({ gender: { $ne: "MEN" } });
    }

    if (city !== undefined && city !== null) {
      city = new RegExp(city, "i");
      pattern.push({ "contact.address": city });
    }

    if (pattern.length > 0) {
      query = { $or: pattern };
    }
    // Find shops with the given query and sort by rating
    const shops = await Shop.find(query)
      .sort({ "rating.totalMembers": -1, "rating.avg": -1 })
      .limit(10)
      .populate("services");

    return callback(null, shops);
  } catch (e) {
    return callback(e);
  }
}
```

6.3.3.1.2 View Shop

view_shop

```
async function getShopByShopId(shopId, callback) {
  // Find shop by either shopId or objectId
  let pattern = [];
  let query;
  if (shopId !== undefined && shopId !== null) {
    pattern.push({ shopId: shopId });
  }
  if (ObjectId.isValid(shopId)) {
    pattern.push({ _id: ObjectId(shopId) });
  }
  if (pattern.length > 0) {
    query = { $or: pattern };
  }
  Shop.findOne(query)
    .populate("services")
    .then((response) => {
      if (!response) callback("Shop not found with id " + shopId);
      else callback(null, response);
    })
    .catch((error) => {
      return callback(error);
    });
}
```

6.3.3.1.3 View shop rating and reviews

view_reviews

```
async function getReviews(params, callback) {
  const { id } = params;
  Reviews.find({ to: id })
    .populate("from", "name email roles userPic")
    .then((res) => {
      return callback(null, res);
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.3.3.2 Appointments

6.3.3.2.1 Book Appointments

book_appointment

```
async function bookAppointment(params, callback) {
  try {
    const { shopId, userId, memberId } = params;
    let startTime = new Date(params.startTime);
    if (startTime < new Date()) {
      return callback({
        status: 400,
        message: "Start time should be greater than current time",
      });
    }
    let user = await User.findById(userId);
    let shop = await Shop.findById(shopId);

    if (!shop) return callback("Shop not found");
    if (!user) return callback("User not found");

    let memberFound = false;
    // check if member is present in the shop
    shop.members.forEach((member) => {
      if (member.member.toString() == memberId) {
        memberFound = true;
        console.log(`member found ${memberId}`);
      }
    });
    if (!memberFound) return callback("Member not found");

    servicesIds = params.services;
    let services = [];
    let totalPrice = 0;
    let totalTime = 0;
    // Calculate total price and total time for the services
    for (let i = 0; i < servicesIds.length; i++) {
      if (shop.services.includes(servicesIds[i])) {
        const service = await Services.findById(servicesIds[i]);
        if (!service) return callback("Service not found");
        totalPrice += parseFloat(service.price);
        totalTime += parseInt(service.time);
        services.push(service);
      } else {
        return callback("Service not found in shop with id: " + servicesIds[i]);
      }
    }
    // Calculate end time for the appointment by adding total time to start time
    let endTime = moment(startTime).add(totalTime, "minutes").toDate();
    // Get the slots between start time and end time
    let slots = getSlots(startTime, endTime);
    if (slots.length === 0) {
      return callback("Slots time is not available today");
    }
    // Covert start time to DD-MMM-YYYY format
    let bookingDate = getDDMMMYYYYDate(startTime);
    // Check if slots are already booked for the date
    const preBookedSlots = await SlotsBooked.find({
      shopId: shopId,
      memberId: memberId,
      date: bookingDate,
      slots: { $in: slots },
    });
    if (preBookedSlots.length > 0) {
      return callback("Slots are already booked");
    }
    // Create appointment object
    const appointment = await Appointment.create({
      shopId: shopId,
      userId: userId,
      memberId: memberId,
      totalPrice: totalPrice,
      totalTime: totalTime,
      services: services,
      slots: slots,
      startTime: startTime,
      endTime: endTime,
    });
    if (!appointment) return callback("Appointment not created");
    // Add appointment to user appointments
    const updatedUser = await User.findOneAndUpdate(
      { _id: userId },
      {
        $push: {
          appointments: appointment._id,
        },
      },
      { new: true, upsert: true }
    );
    if (!updatedUser) return callback("User not found");
    // Create or update slots booked for the date, shop and member
    const slotBooked = await SlotsBooked.findOneAndUpdate(
      { date: bookingDate, shopId: shopId },
      {
        shopId: shopId,
        memberId: memberId,
        date: bookingDate,
        $addToSet: { slots: slots },
      },
      { new: true, upsert: true }
    );
    if (!slotBooked) {
      return callback("Slot not booked");
    }
    // Add appointment and slots booked to shop appointments
    const updatedShop = await Shop.findOneAndUpdate(
      { _id: shopId },
      {
        $addToSet: {
          slotsBooked: slotBooked._id,
          appointments: appointment._id,
        },
      },
      { new: true }
    );
    if (!updatedShop) return callback("Shop not found");
    const bookingData = await Appointment.findById(appointment._id)
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    return callback(null, bookingData);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3.2.2 Get user appointments

get_user_appointments

```
async function getUserAppointments(req, res, callback) {
  try {
    const { id } = req.params;
    const { past } = req.query;
    let appointmentStatus = ["PENDING", "ACCEPTED", "CONFIRMED", "CANCELLED", "COMPLETED", "REJECTED"];
    let filter = {
      userId: id,
      appointmentStatus: { $in: appointmentStatus },
      endTime: { $gt: new Date() },
    };
    // if past is true then get past appointments
    if (past !== undefined && past == "true") {
      filter.endTime = { $lt: new Date() };
    }
    if (!ObjectId.isValid(id)) {
      return callback({
        status: 400,
        message: "Invalid user id",
      });
    }
    // Check is user authorized to get appointments
    if (id != req.user.id && req.user.isAdmin == false) {
      return callback({
        status: 401,
        message: "Unauthorized",
      });
    }
    const appointments = await Appointment.find(filter)
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    return callback(null, appointments);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3.2.3 View appointments

view_appointment

```
async function getAppointment(req, callback) {
  try {
    const { id } = req.params;
    if (!ObjectId.isValid(id)) {
      return callback({
        status: 400,
        message: "Invalid appointment id",
      });
    }
    const appointment = await Appointment.findById(id)
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    if (!appointment) return callback("Appointment not found");
    return callback(null, appointment);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3.2.4 Cancel appointments

cancel_appointments

```
async function cancelAppointment(req, res, callback) {
  try {
    const { id } = req.params;
    const appointment = await Appointment.findById(id);
    if (!appointment) return callback("Appointment not found");
    if (appointment.appointmentStatus == "COMPLETED") {
      return callback("Appointment is already completed");
    }
    if (appointment.appointmentStatus == "CANCELLED") {
      return callback("Appointment is already cancelled");
    }
    // Check if user is authorized to cancel appointment
    if (
      req.user.id !== appointment.userId.toString() &&
      req.user.isAdmin == false
    ) {
      return callback("User not authorized");
    }
    // Remove booked slots from slots booked collection
    const updateSlotsBooked = await SlotsBooked.findOneAndUpdate(
      {
        date: getDDMMMYYYYDate(appointment.startTime),
        shopId: appointment.shopId,
        memberId: appointment.memberId,
      },
      {
        $pullAll: {
          slots: appointment.slots,
        },
      },
      { new: true }
    );
    if (!updateSlotsBooked) return callback("Slots not found");
    // Update appointment status to cancelled
    const updatedAppointment = await Appointment.findByIdAndUpdate(
      { _id: id },
      {
        appointmentStatus: "CANCELLED",
      },
      { new: true }
    )
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    if (!updatedAppointment) return callback("Operation failed");
    return callback(null, updatedAppointment);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3.2.5 Complete appointments

complete_appointment

```
async function completeAppointment(req, res, callback) {
  try {
    const { id } = req.params;
    const appointment = await Appointment.findById(id);
    if (!appointment) return callback("Appointment not found");
    if (appointment.appointmentStatus == "COMPLETED") {
      return callback("Appointment is already completed");
    }
    if (appointment.appointmentStatus == "CANCELLED") {
      return callback("Appointment is already cancelled");
    }
    // check if user is authorized to complete the appointment
    if (
      req.user.id !== appointment.userId.toString() &&
      req.user.isAdmin == false
    ) {
      return callback("User not authorized");
    }
    // Update the appointment status to completed
    const updatedAppointment = await Appointment.findByIdAndUpdate(
      { _id: id },
      {
        appointmentStatus: "COMPLETED",
      },
      { new: true }
    )
      .populate("shopId", "owner shopId shopName shopLogo contact members")
      .populate("userId", "name gender roles userPic favoriteShops address");
    if (!updatedAppointment) return callback("Operation failed");
    return callback(null, updatedAppointment);
  } catch (error) {
    return callback(error);
  }
}
```

6.3.3.3 Reviews

6.3.3.3.1 Give review

review_shop

```
async function addOrUpdateReview(params, callback) {
  const { from, to, model_type, comment, rating } = params;
  let shop = await Shop.findById(to);
  let user = await User.findById(from);
  if (shop == null) {
    return callback({ status: 404, message: "Shop not found"});
  }
  if (user == null) {
    return callback({ status: 404, message: "User not found"});
  }
  // Check is rating is valid
  if (rating > 5 || rating < 0.5) {
    return callback({
      status: 400,
      message: "Invalid Rating",
    });
  }
  let shopRating = shop.rating;
  // Find user old review
  let review = await Reviews.findOne({ from, to });
  if (review !== null) {
    // if user has already reviewed the shop then decrement the old review
    let rating = parseFloat(review.rating);
    shopRating = removeShopRatingInfo(shopRating, rating);
  }
  // Add new review
  shopRating = addShopRatingInfo(shopRating, rating);
  // Calculate new rating of shop
  let ratingSum = getRatingSum(shopRating);
  shopRating.avg = ratingSum / shopRating.totalMembers;
  // Create or update review
  Reviews.findOneAndUpdate({ from, to, model_type }, params, {
    new: true,
    upsert: true,
  })
    .populate("from", "name email roles userPic")
    .then(async (res) => {
      let shopId = res.to;
      // Update shop rating
      let updatedShop = await Shop.findByIdAndUpdate(
        shopId,
        {
          rating: shopRating,
          $addToSet: {
            reviews: res._id,
          },
        },
        { new: true }
      );
      return callback(null, updatedShop);
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.3.3.3.2 Delete review

delete_review

```
async function deleteReview(req, callback) {
  const reviewId = req.params.reviewId;

  Reviews.findByIdAndDelete(reviewId)
    .populate("from", "name email roles userPic")
    .then(async (res) => {
      if (res == null) {
        return callback({ status: 404, message: "Review not found" });
      }
      // Update shop rating
      let shop = await Shop.findById(res.to);
      let shopRating = shop.rating;
      let rating = parseFloat(res.rating);
      shopRating = removeShopRatingInfo(shopRating, rating);
      let ratingSum = getRatingSum(shopRating);
      shopRating.avg = parseInt(ratingSum / shopRating.totalMembers) || 3;
      let updatedShop = await Shop.findByIdAndUpdate(
        res.to,
        {
          rating: shopRating,
          $pull: {
            reviews: res._id,
          },
        },
        { new: true }
      );
      return callback(null, updatedShop);
    })
    .catch((e) => {
      return callback(e);
    });
}
```

6.4 Data validation

6.4.1 Login form validation

login_form_validation

```
const userLoginValidationRules = () => {
  return [
    // username must be an email
    body("email").isEmail().withMessage("Email must be valid"),
    // password must be at least 5 chars long
    body("password")
      .isLength({ min: 6 })
      .withMessage("Password must be at least 6 chars long"),
  ];
};
```

6.4.2 Register form validation

register_form_validation

```
const userRegisterValidationRules = () => {
  return [
    body("email").isEmail().withMessage("Email must be valid"),
    body("name")
      .not()
      .isEmpty()
      .trim()
      .escape()
      .isLength({ min: 3, max: 20 })
      .withMessage("Name must be between 3 and 20 characters"),
    body("phone")
      .isLength({ min: 10, max: 10 })
      .withMessage("Phone number must be 10 digit")
      .isMobilePhone()
      .withMessage("Phone must be valid"),
    // password must be at least 5 chars long
    body("password")
      .isLength({ min: 6 })
      .withMessage("Password must be at least 6 chars long"),
  ];
};
```

6.4.3 Shop register form validation

shop_register_form_validation

```
const shopRegisterValidationRules = () => {
  return [
    body("email").isEmail().withMessage("Email must be valid"),
    body("name")
      .not()
      .isEmpty()
      .trim()
      .escape()
      .isLength({ min: 3, max: 20 })
      .withMessage("Shop name must be between 3 and 20 characters"),
    body("phone")
      .isLength({ min: 10, max: 10 })
      .withMessage("Phone number must be 10 digit")
      .isMobilePhone()
      .withMessage("Phone must be valid"),

  ];
};
```

6.4.4 Service form validation

service_form_validation

```
const serviceValidationRules = () => {
  return [
    body("name")
      .not()
      .isEmpty()
      .trim()
      .escape()
      .isLength({ min: 3, max: 20 })
      .withMessage("Shop name must be between 3 and 20 characters"),
    body("price")
      .isLength({ min: 0 })
      .withMessage("Price must be at least 0"),
    body("duration")
      .isLength({ min: 0 })
      .withMessage("Duration must be at greater than 0 minutes")

  ];
};
```

6.4.4 Category form validation

category_form_validation

```
const categoryValidationRules = () => {
  return [
    body("name")
      .not()
      .isEmpty()
      .trim()
      .escape()
      .isLength({ min: 3, max: 20 })
      .withMessage("Category name must be between 3 and 20 characters"),
    body("image").not().isEmpty().withMessage("Category image is required"),
  ];
};
```

6.5 Named procedures / functions

6.5.1 Getting slot

get_slot

```
function getSlot(date) {
  let hours = date.getHours();
  let minutes = date.getMinutes();
  let slot = hours * 2;
  slot = Math.round(slot);
  if (minutes > 30) {
    slot += 1;
  }
  return slot;
}

function getSlots(startTime, endTime) {
  let slots = [];
  let start = getSlot(startTime);
  //console.log("start ", start);
  let end = getSlot(endTime);
  //console.log("end ", end);
  for (let i = start; i <= end; i++) {
    slots.push(i);
  }
  return slots;
}
```

6.5.1 Sending email

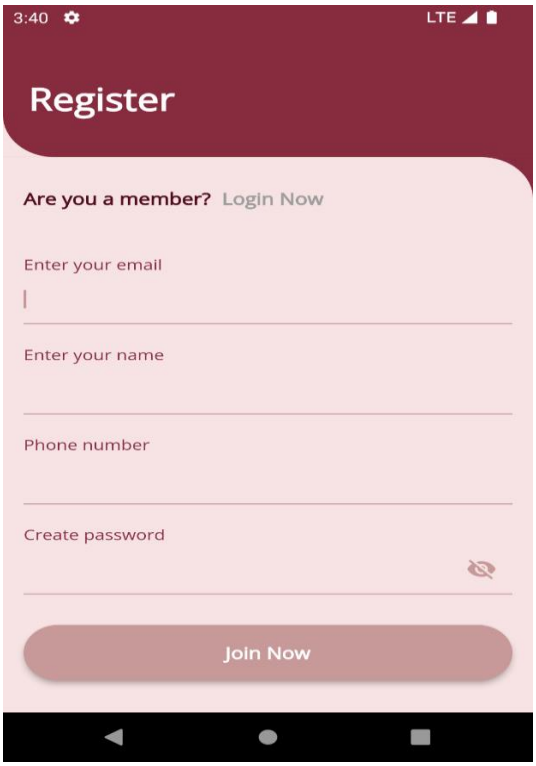
sending_otp_mail

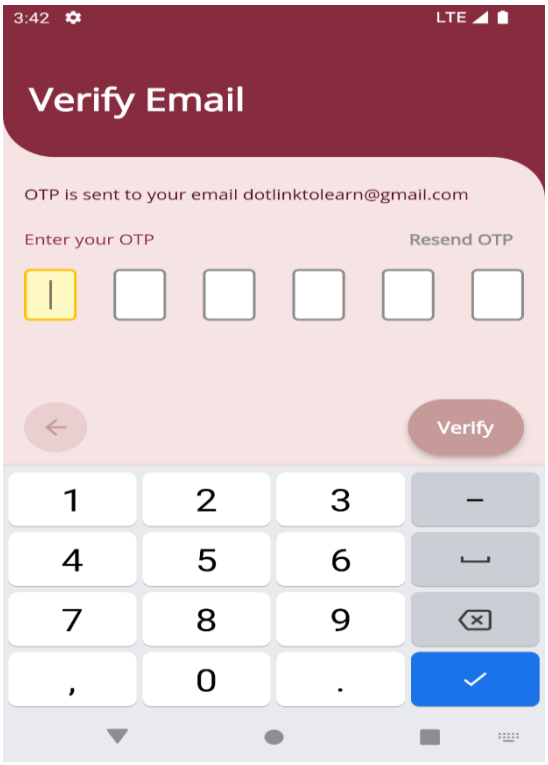
```
exports.sendOTPMail =  (email, otp, msg) => {
    return new Promise((resolve, reject) => {
        const mailOptions = {
            from: process.env.EMAIL_USERNAME,
            to: email,
            subject: "OTP for verification",
            html: `<div style="font-family: Helvetica,Arial,sans-serif;min-width:1000px;overflow:auto;line-
height:2">
                    <div style="margin:50px auto;width:70%;padding:20px 0">
                      <div style="border-bottom:1px solid #eee">
                        <a href="" style="font-size:1.4em;color: #00466a;text-decoration:none;font-
weight:600">Expertis Inc</a>
                      </div>
                      <p style="font-size:1.1em">Verify,</p>
                      <p>${msg}</p>
                      <h2 style="background: #00466a;margin: 0 auto;width: max-content;padding: 0 10px;color:
#fff;border-radius: 4px;">${otp}</h2>
                      <p style="font-size:0.9em;">Regards,<br />Expertis</p>
                      <hr style="border:none;border-top:1px solid #eee" />
                      <div style="float:right;padding:8px 0;color:#aaa;font-size:0.8em;line-height:1;font-
weight:300">
                        <p>Expertis Inc</p>
                        <p>Find your best</p>
                        <p>India</p>
                      </div>
                    </div>
                  </div>`,
        };
        transporter.sendMail(mailOptions, (err, info) => {
            if (err) {
              reject(err);
            } else {
              resolve(info);
            }
        } );
    } );
}
```

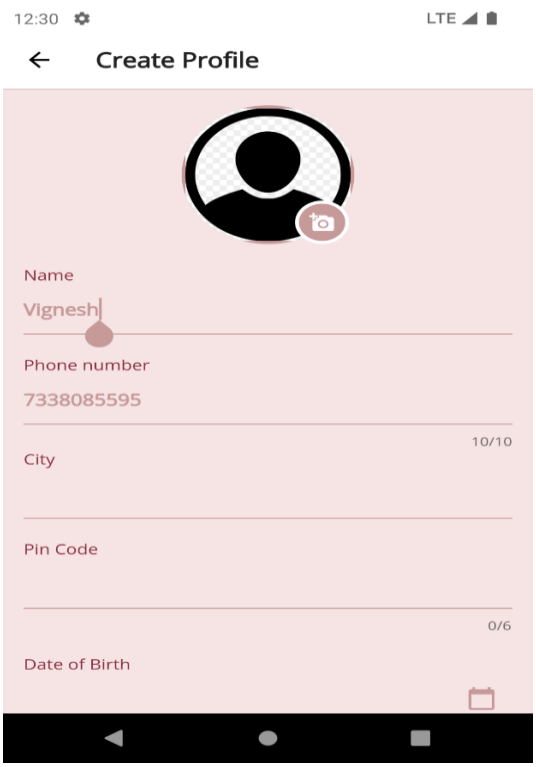
7. User Interface (Screens and Reports)

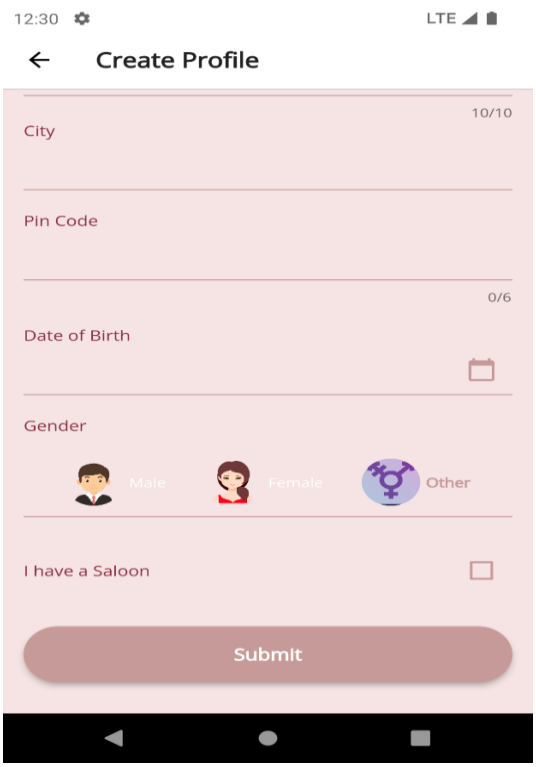
7.1 Register & Login

7.1.1 Register

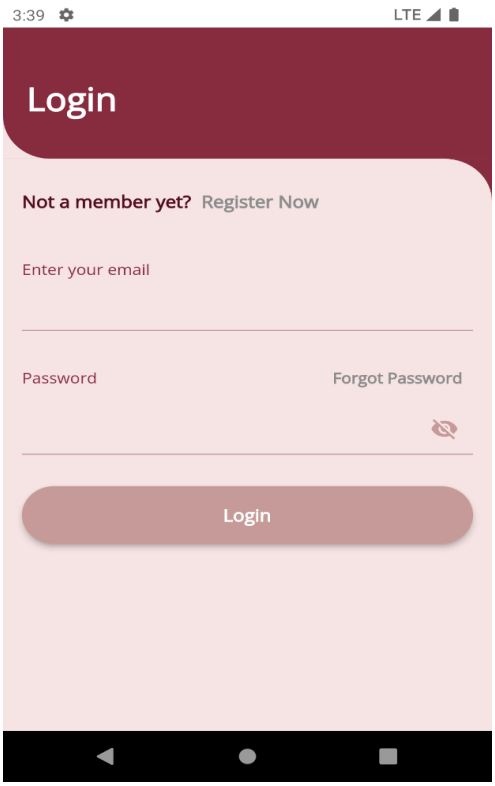




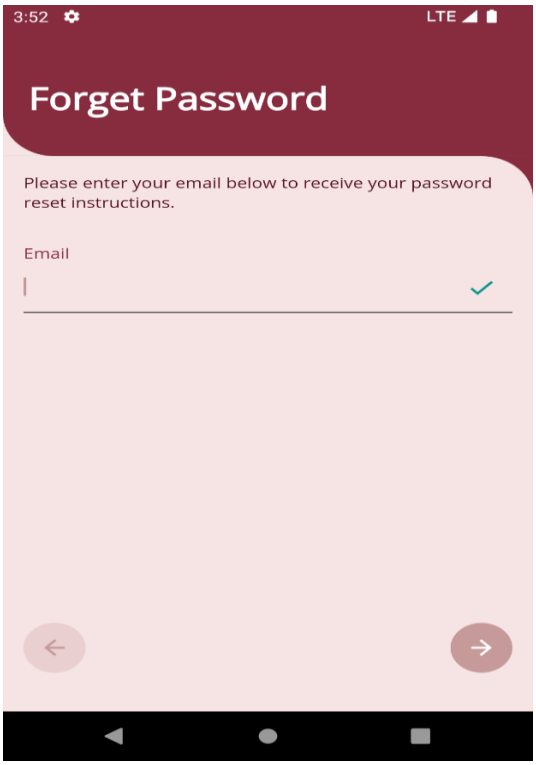

12:30
LTE
Create Profile
Name
Vignesh
Phone number
7338085595
10/10
City
Pin Code
0/6
Date of Birth

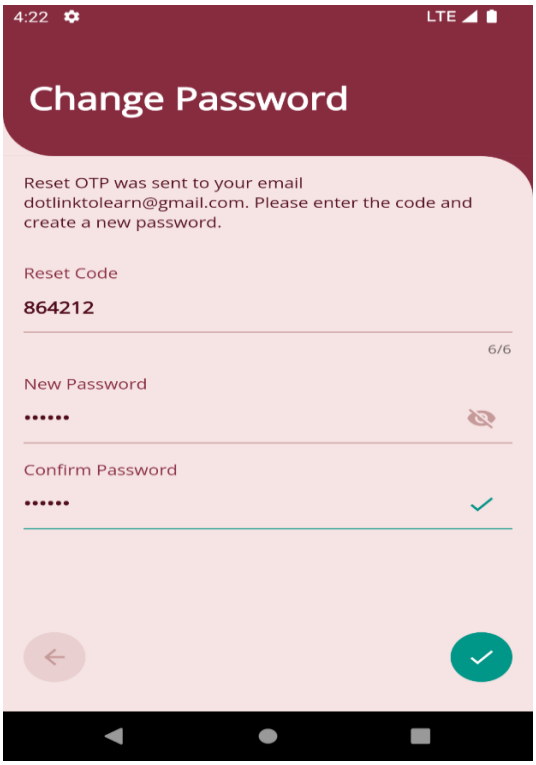

12:30
LTE
Create Profile
10/10
City
Pin Code
0/6
Date of Birth
Gender
Male
Female
Other
I have a Saloon
Submit

7.1.2 Login

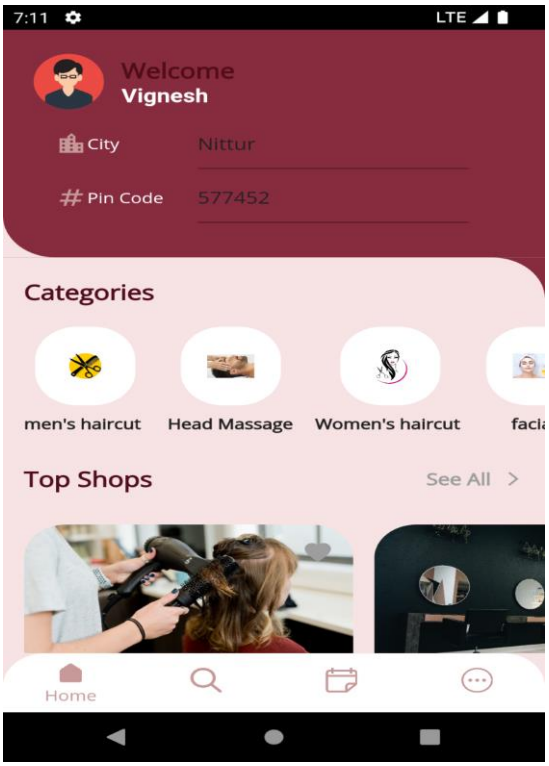

3:39
LTE
Login
Not a member yet? Register Now
Enter your email
Password
Forgot Password
Login

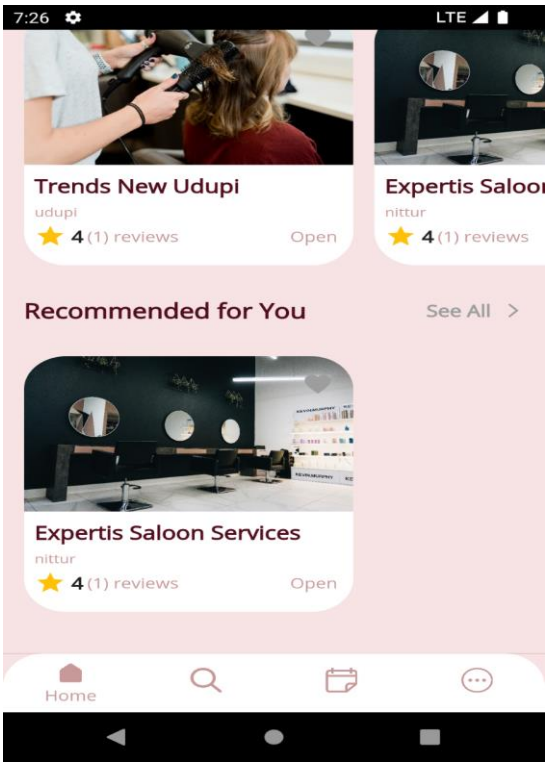
7.1.3. Forget Password



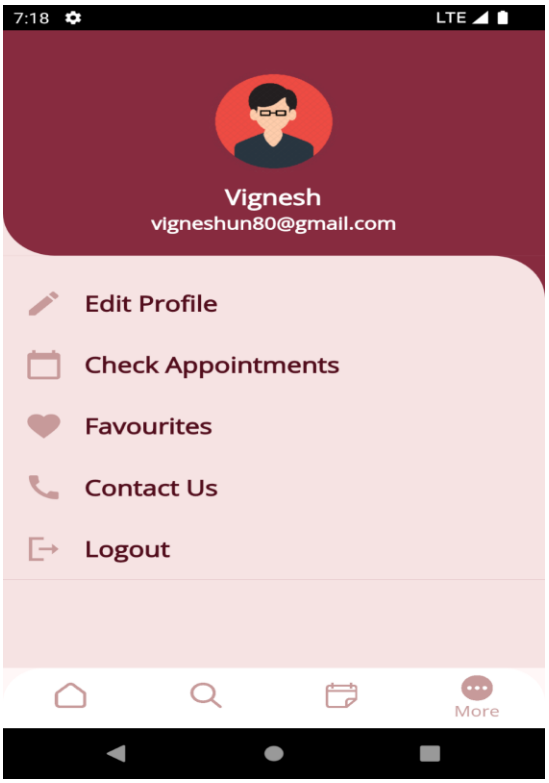


7.2 Main Screen / Home page





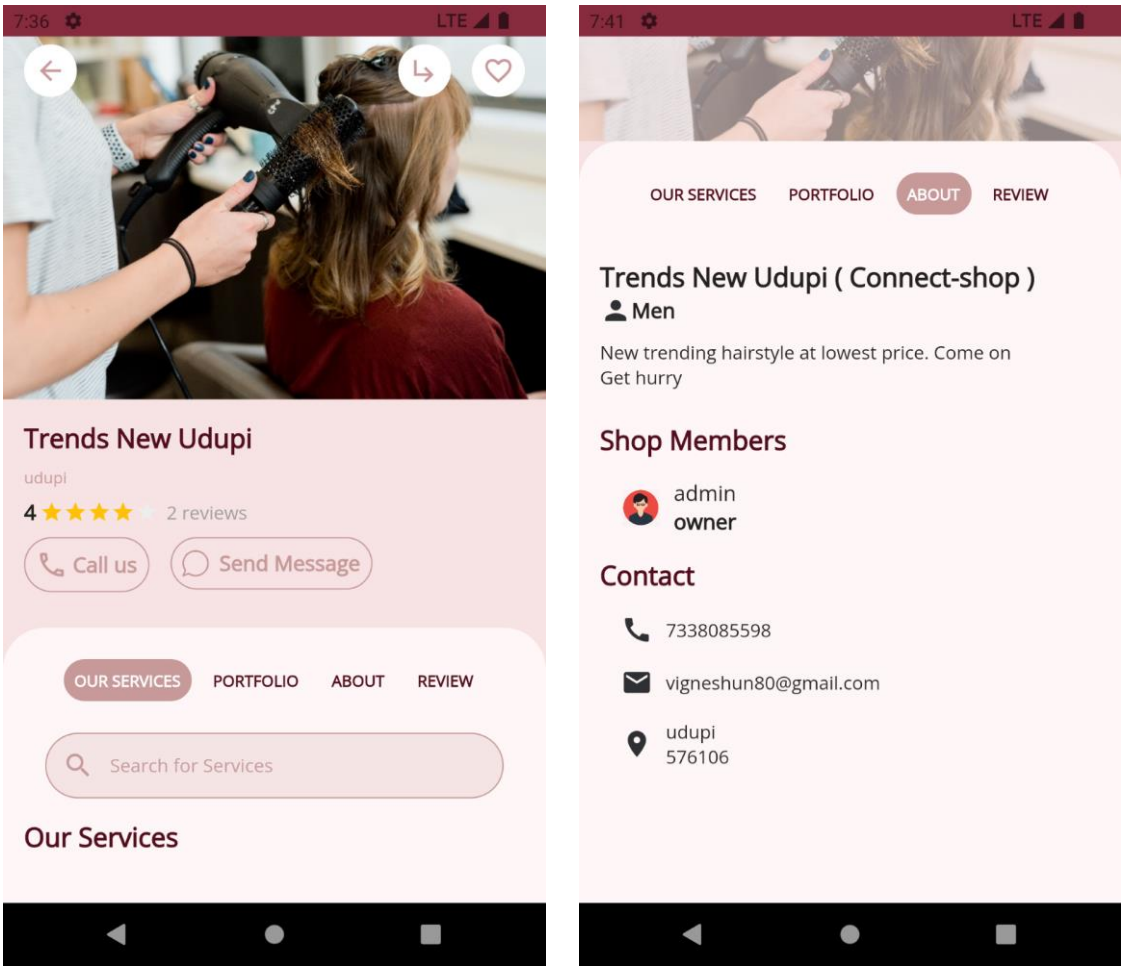
7.3 Menu



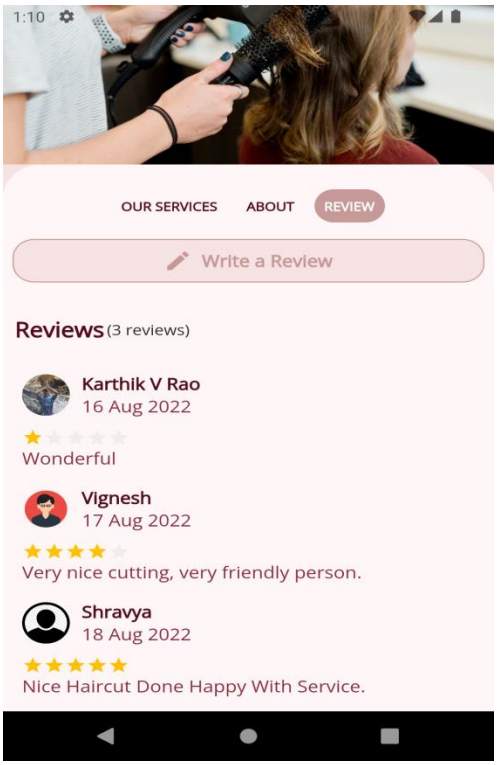
7.4 Data store / retrieval / update

7.4.1 User

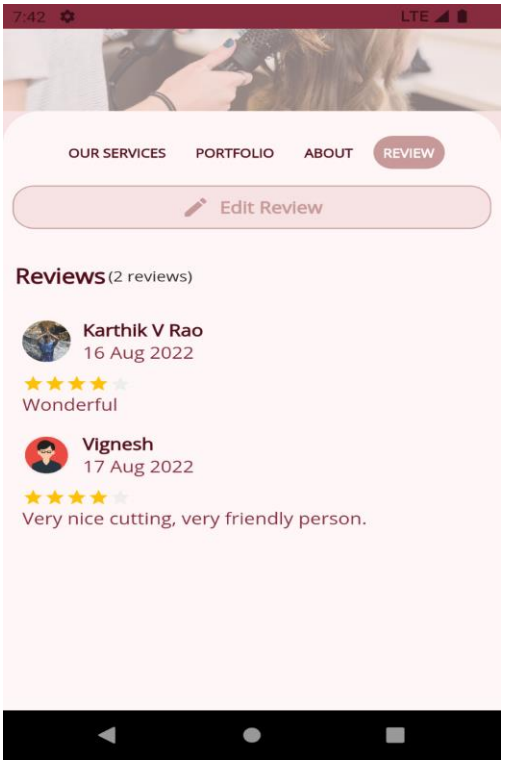
7.4.1.1 View Shop



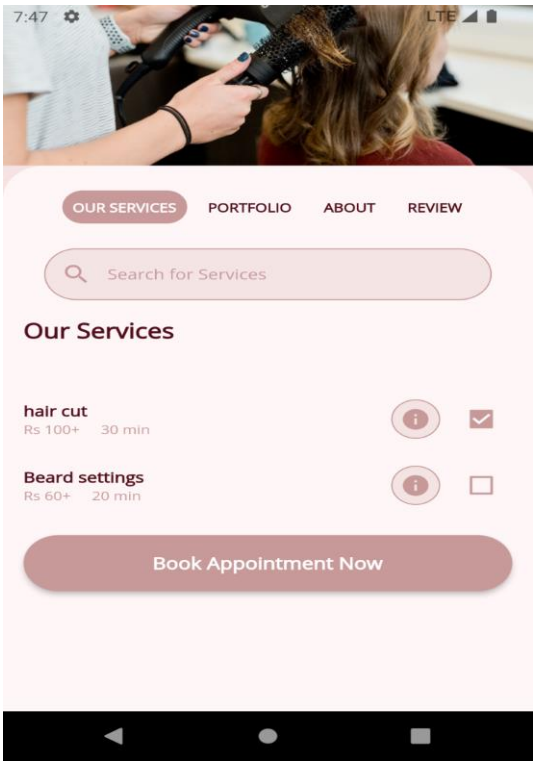
7.4.1.2 Write review

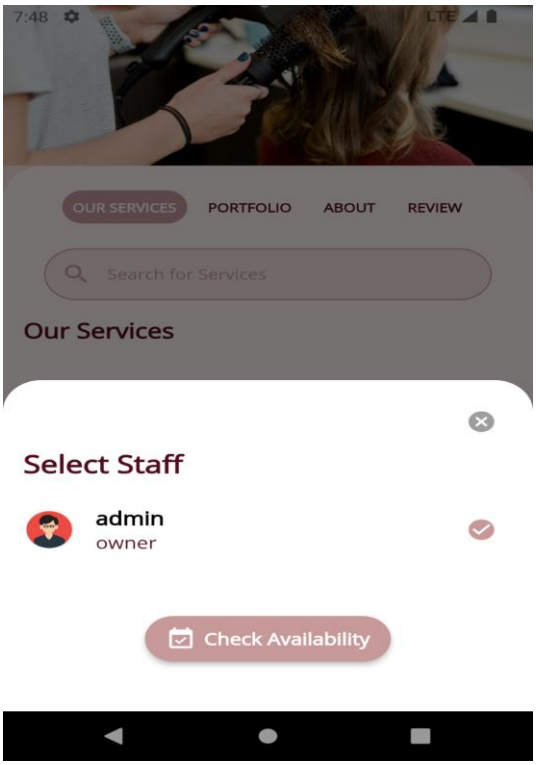


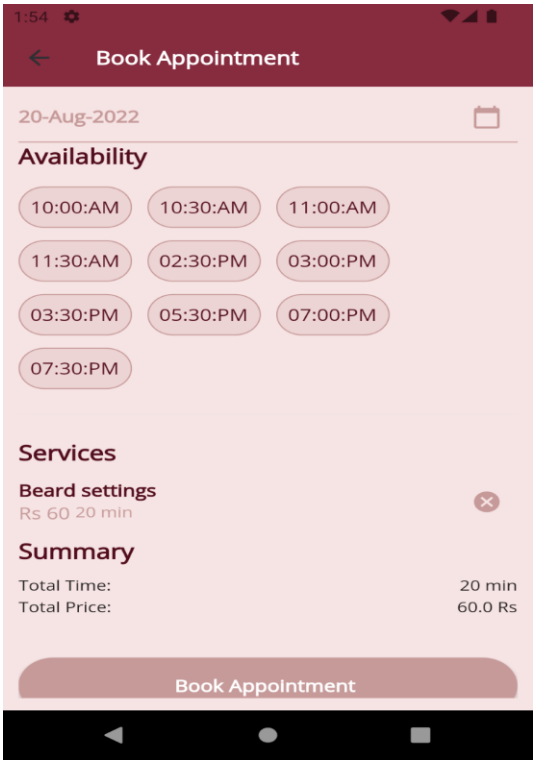
7.4.1.3 Write review



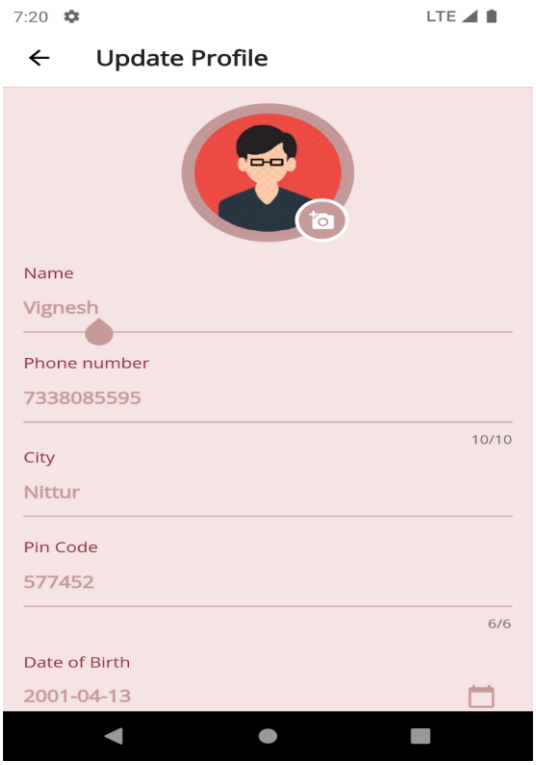
7.4.1.4 Book Appointment

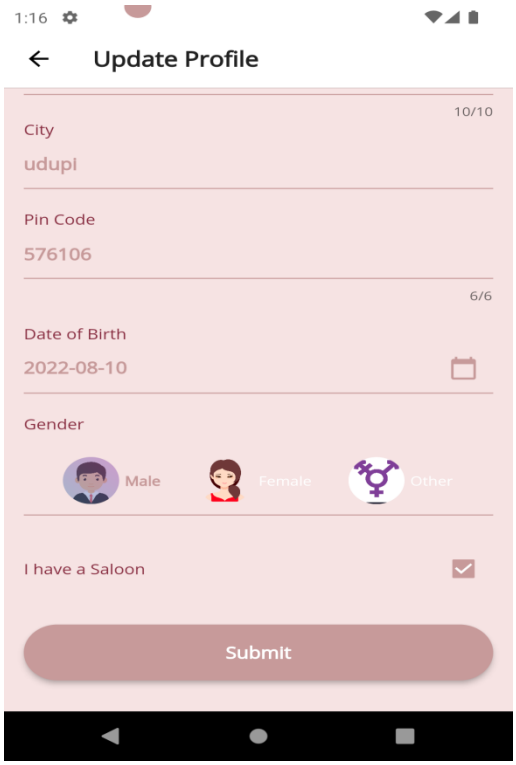




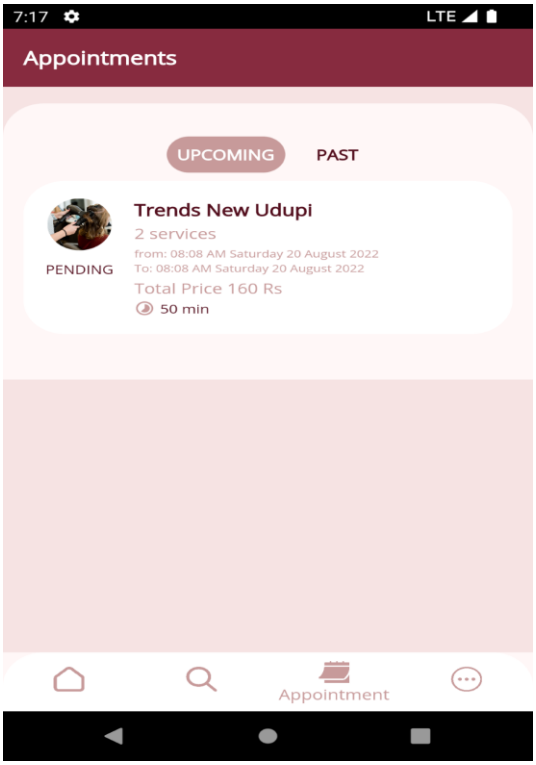


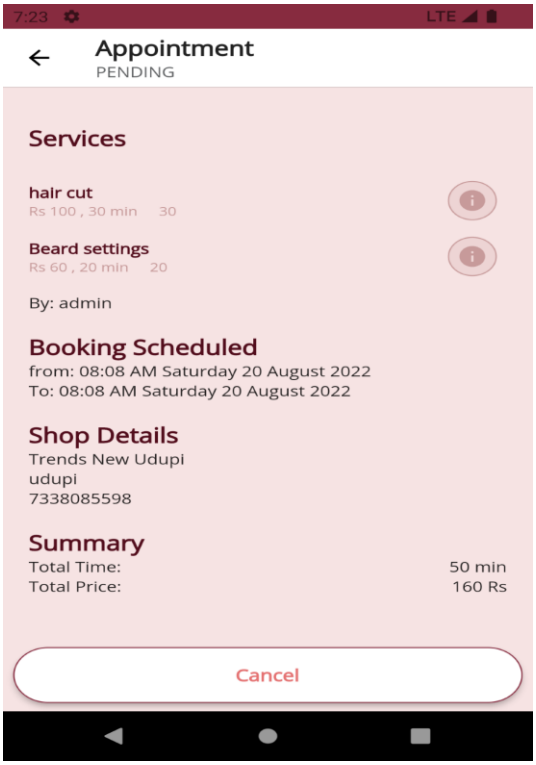
7.4.1.5 Manage profile



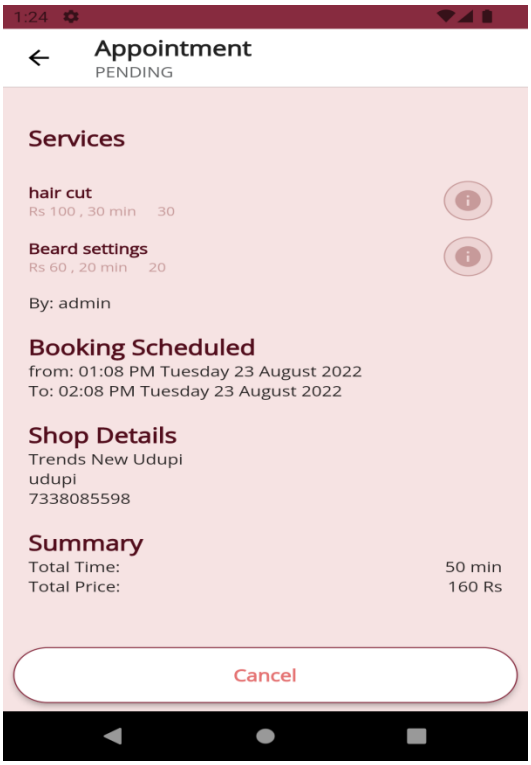


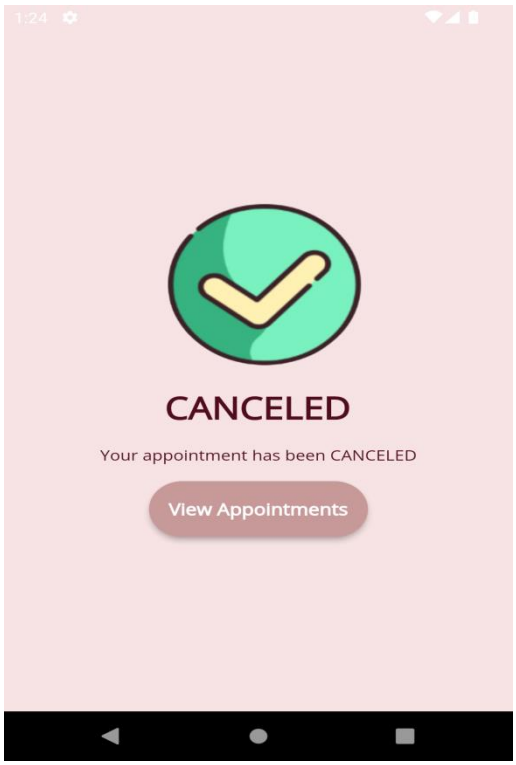
7.4.1.5 View appointments



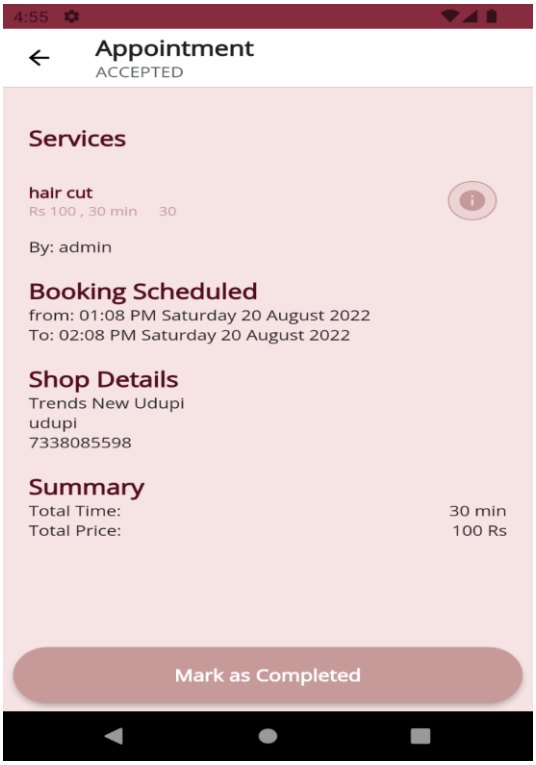


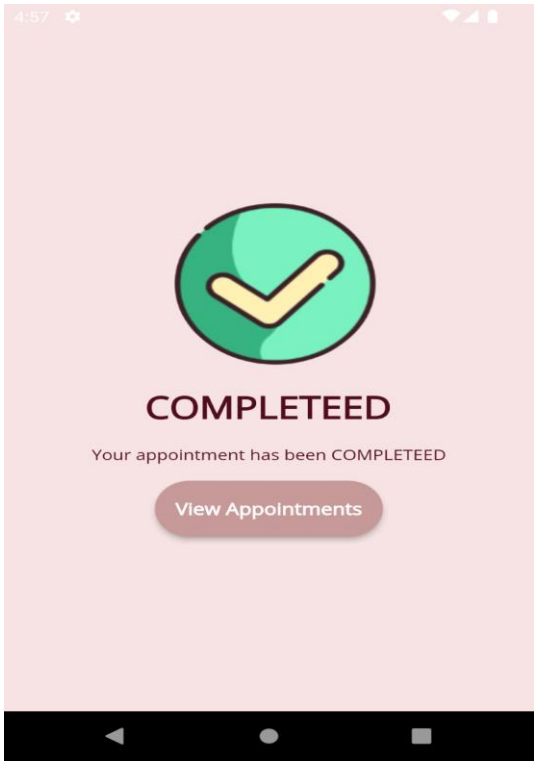
7.4.1.6.1 Cancel Appointment





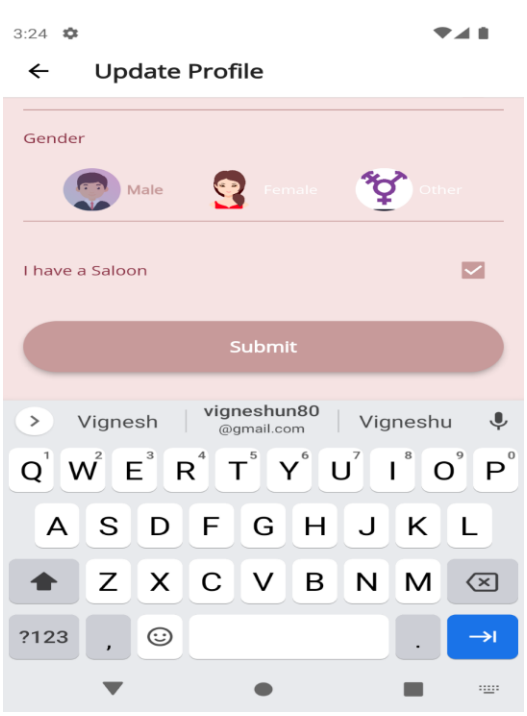
7.4.1.6.1 Complete Appointment

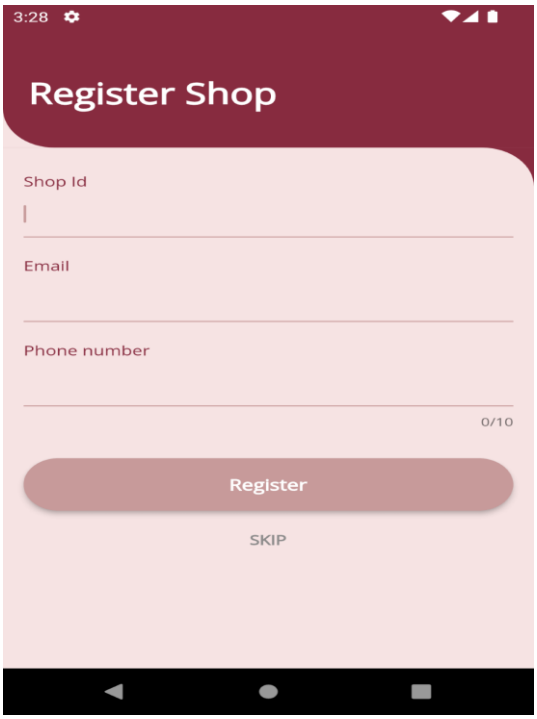




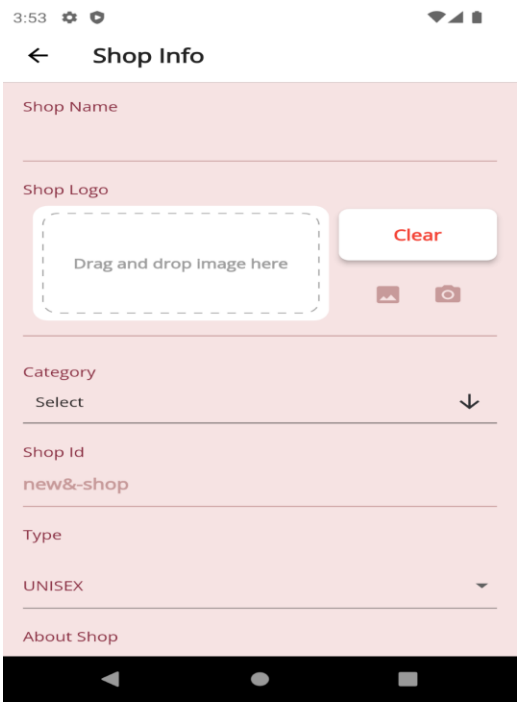
7.4.2 Shop

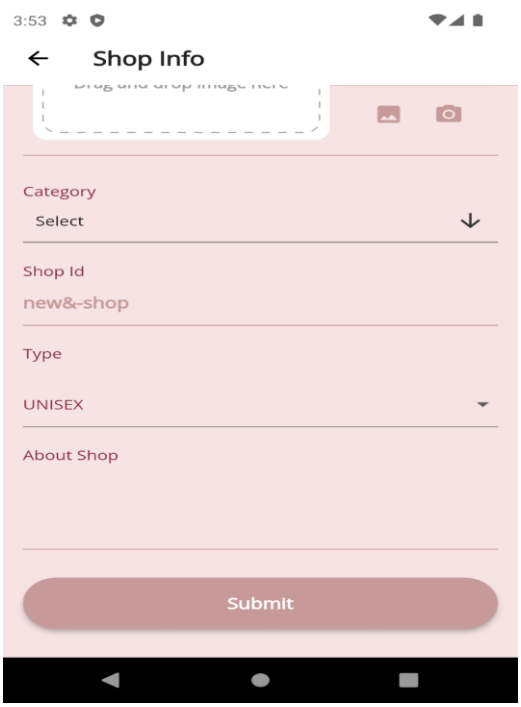
7.4.2.1 Create shop



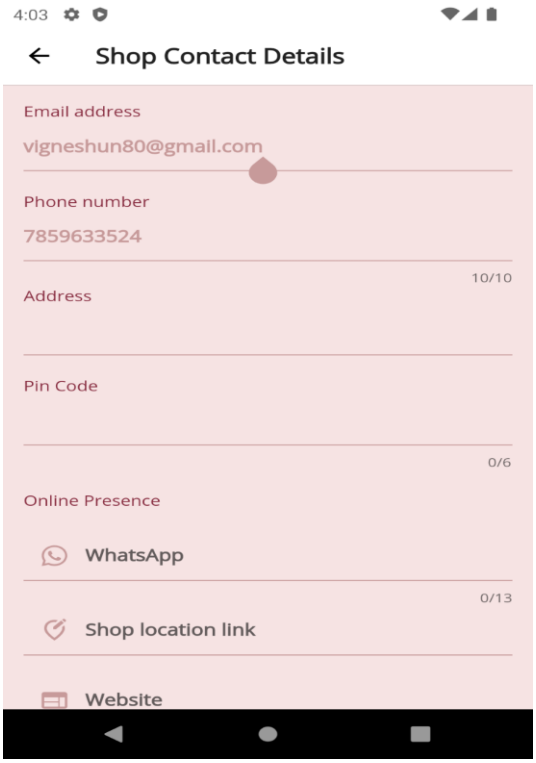


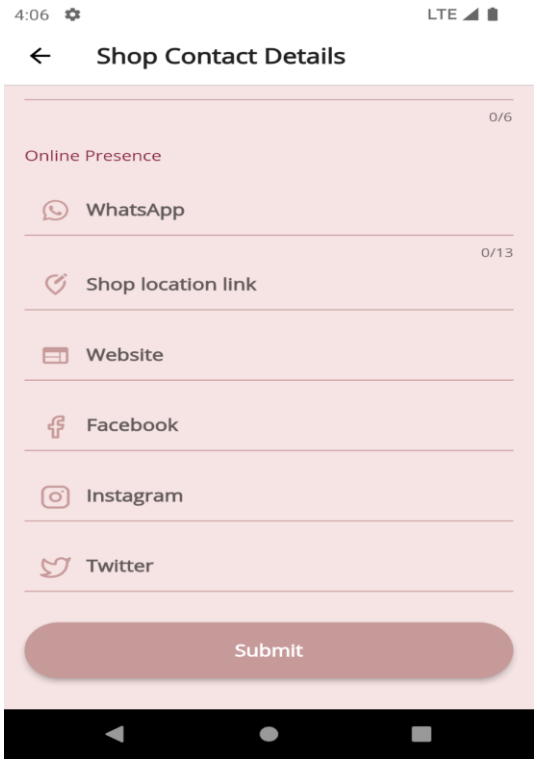
7.4.2.1 Add or update shop information



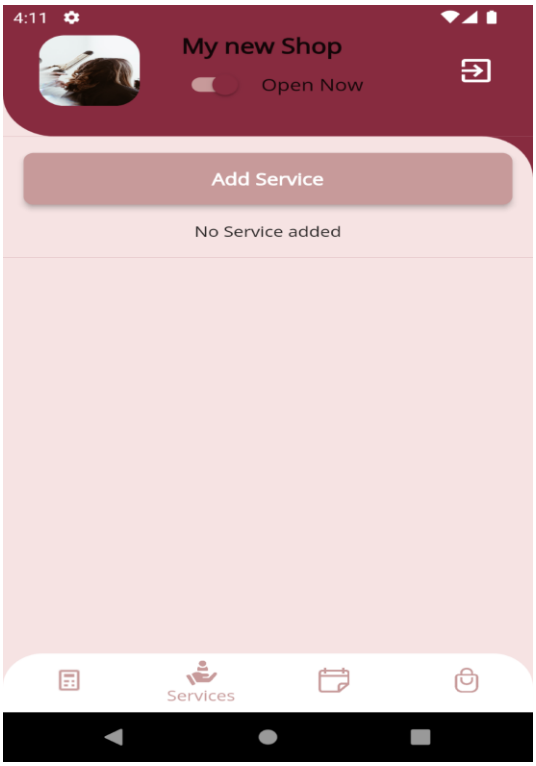


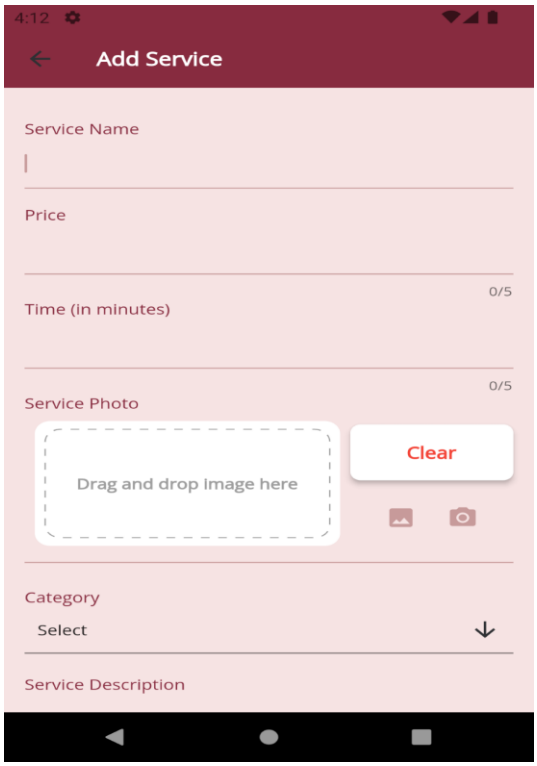
7.4.2.1 Add or update shop contact detail



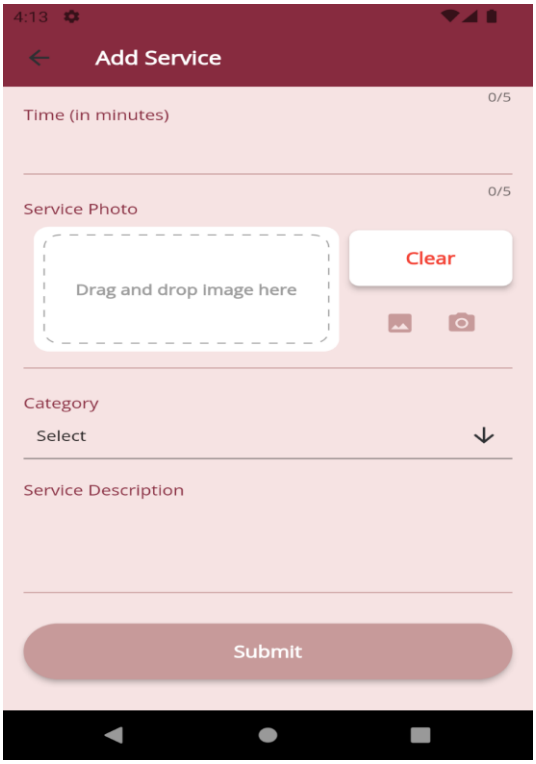


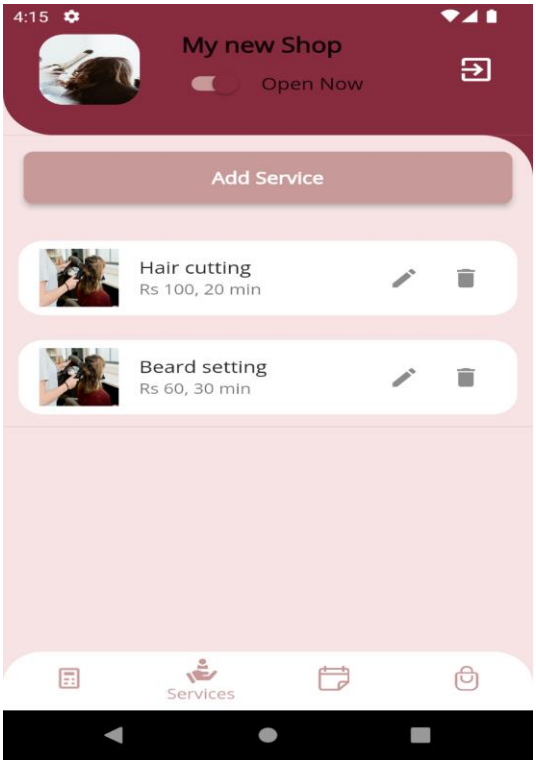
7.4.2.2 Shop Services handling



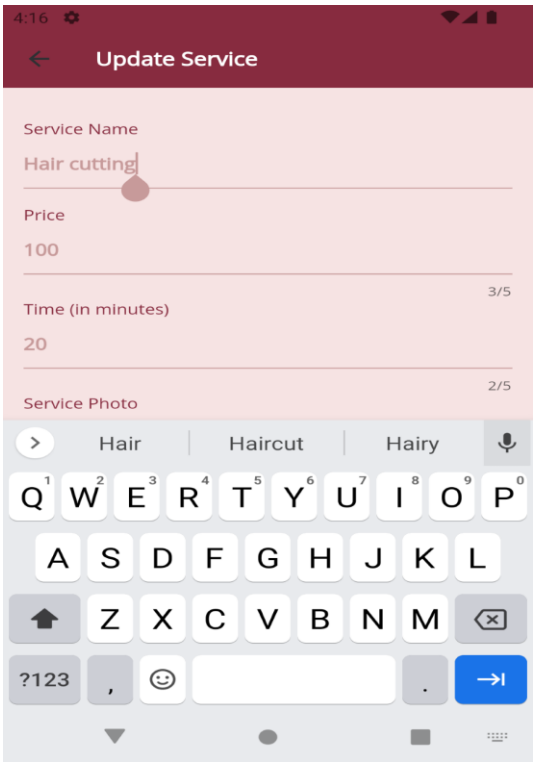


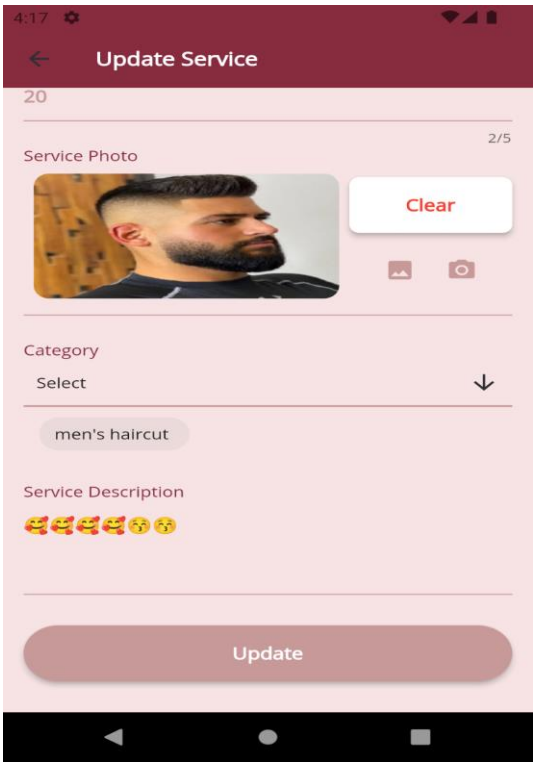
7.4.2.2.1 Adding Services handling



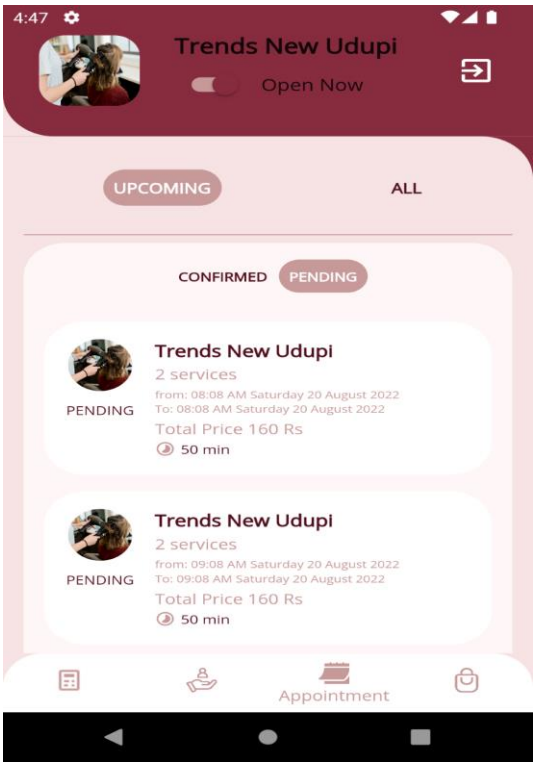


7.4.2.2.2 Updating Services handling

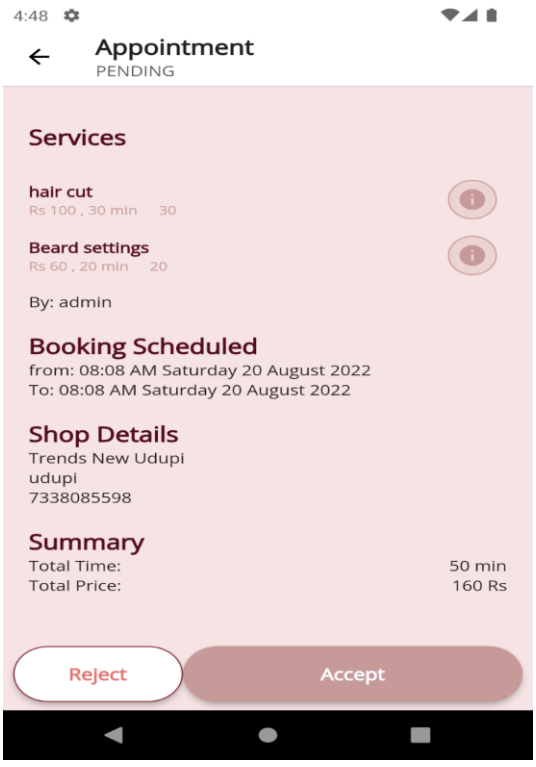


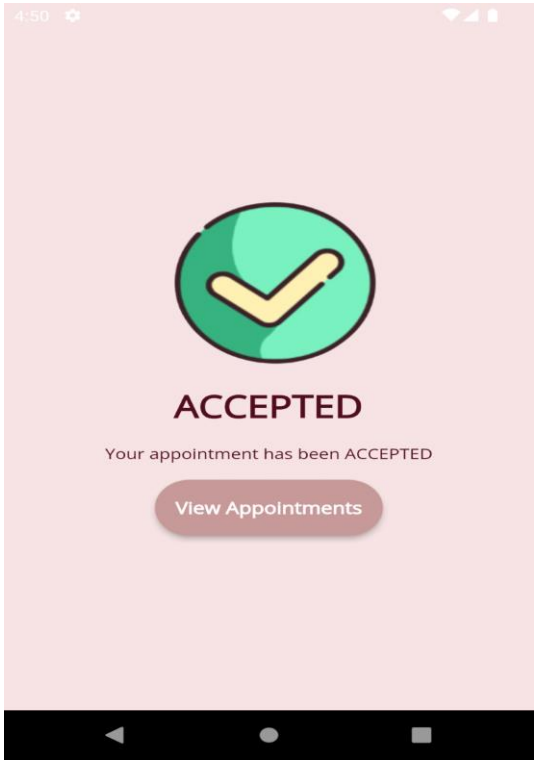


7.4.2.3 Shop appointments

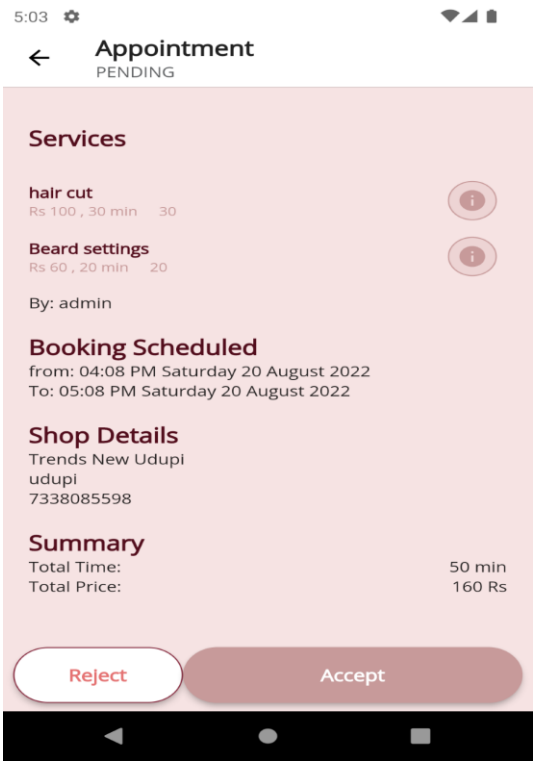


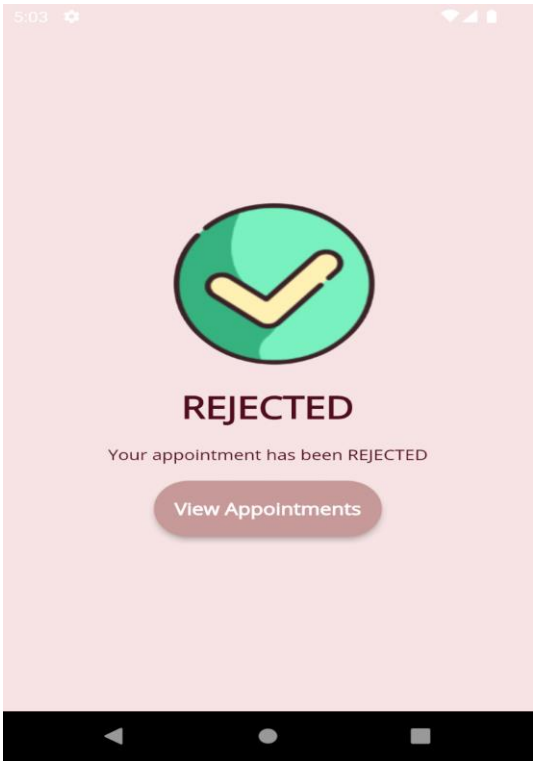
7.4.2.3.1 Accept appointment

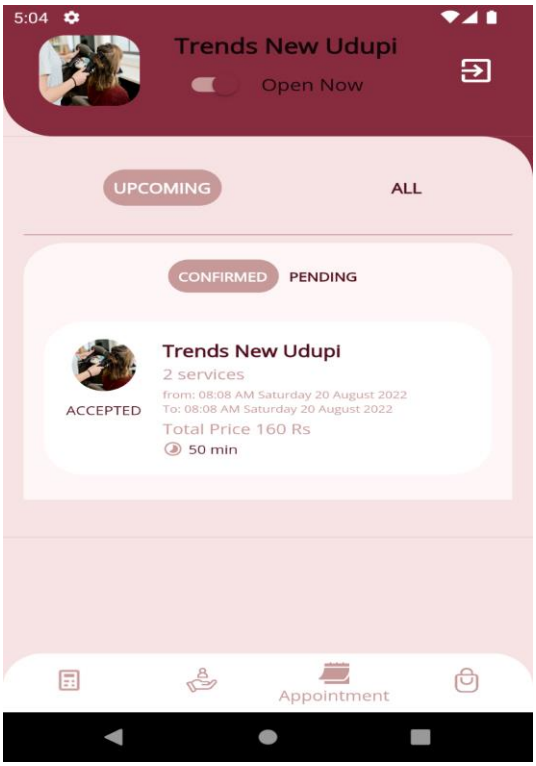


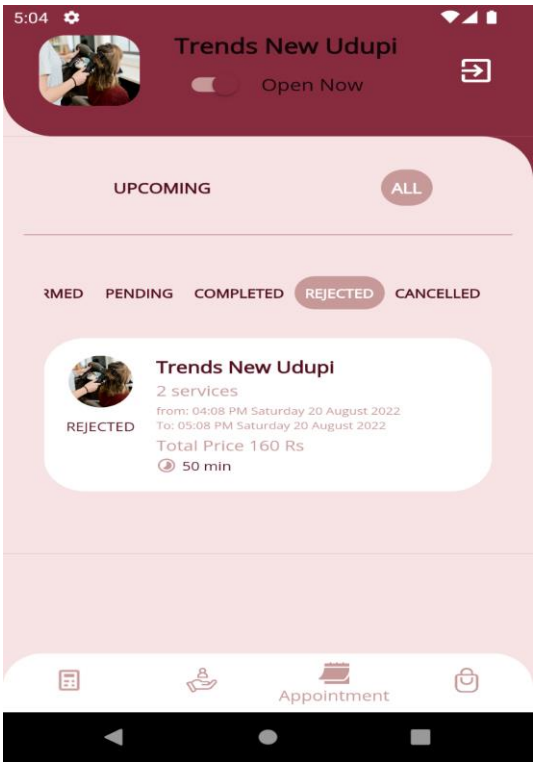


7.4.2.3.2 Reject appointment

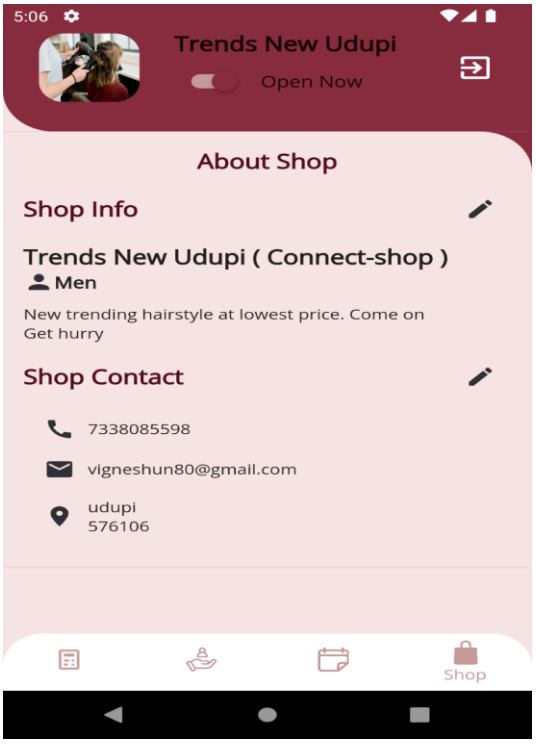


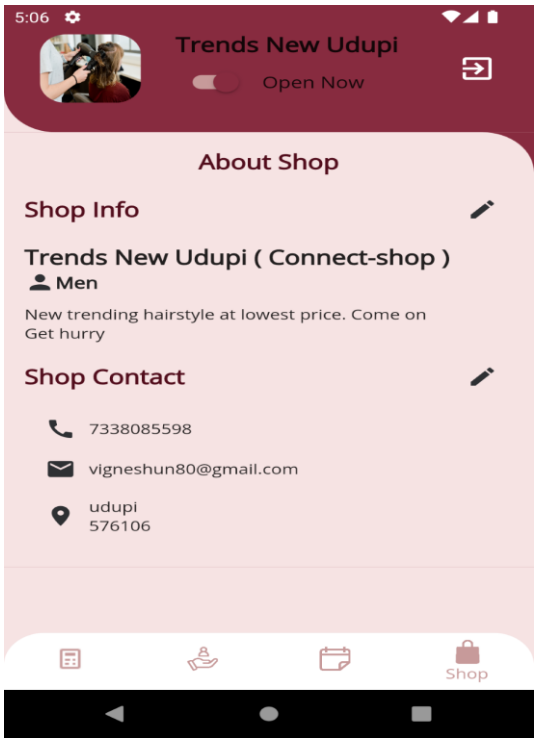






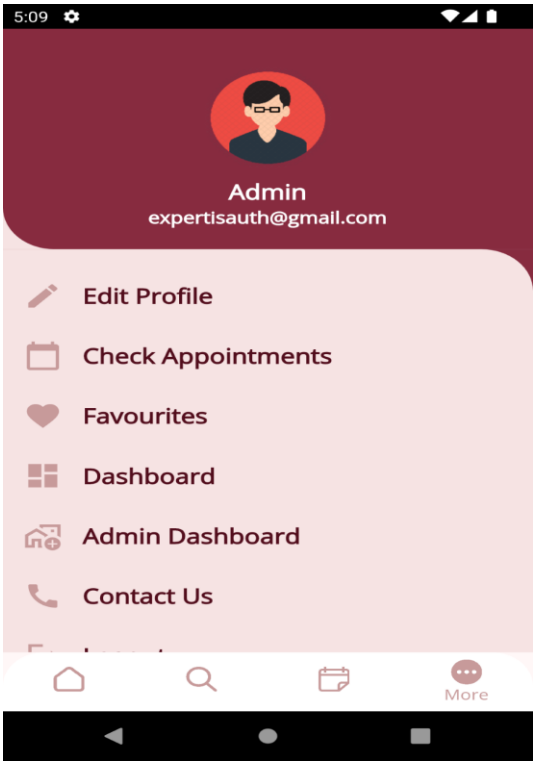
7.4.2.4 Update Shop Information

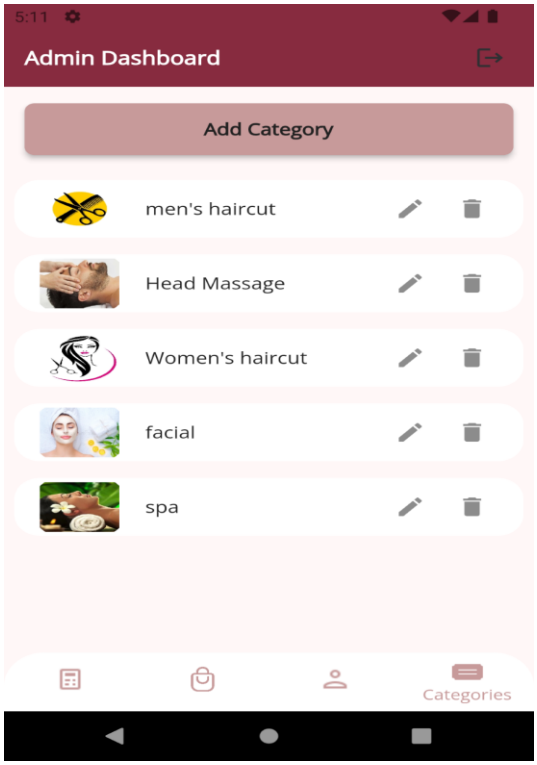




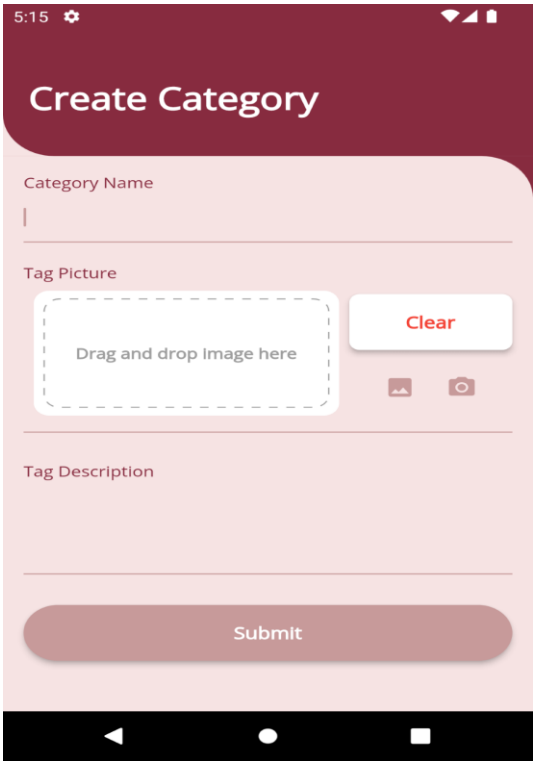
7.4.3. Admin

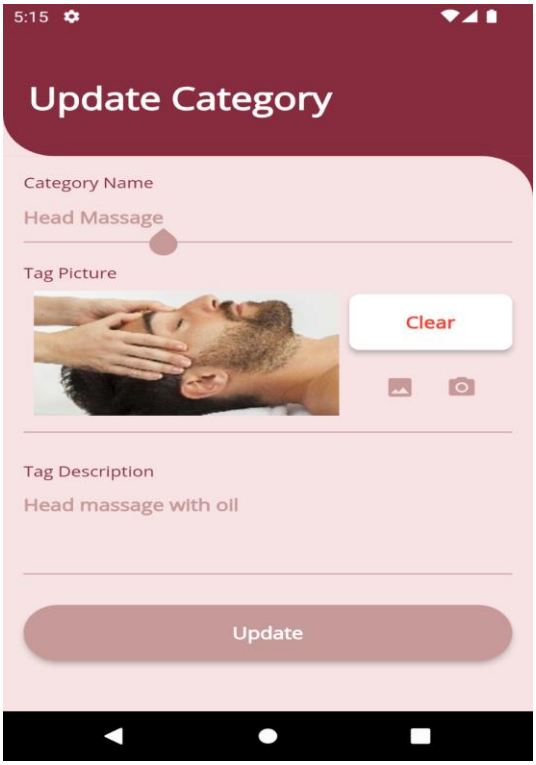
7.4.3.1 Manage category



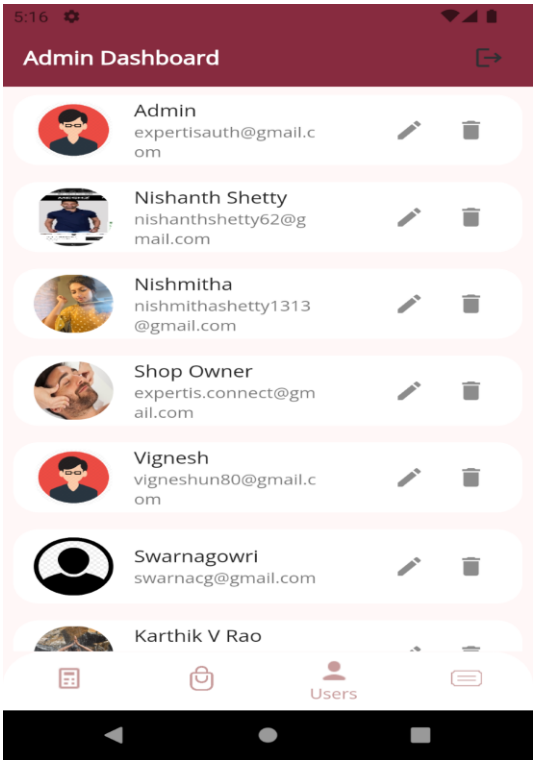


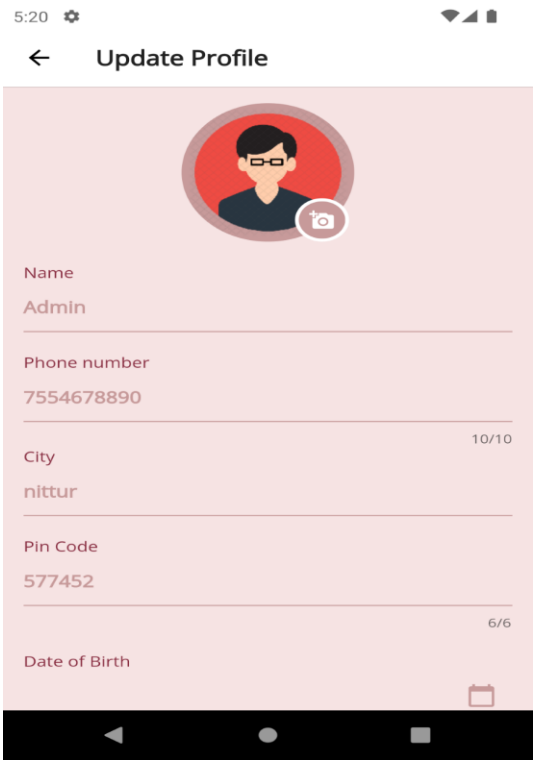
7.4.3.1 Add & update category



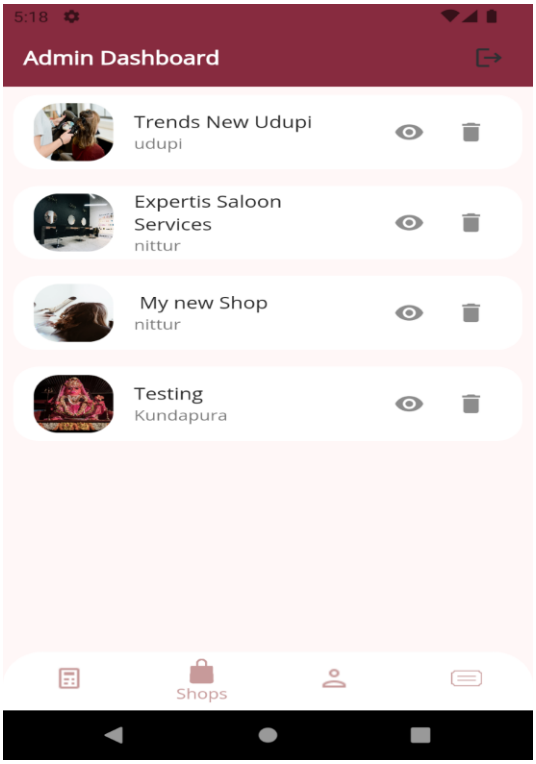


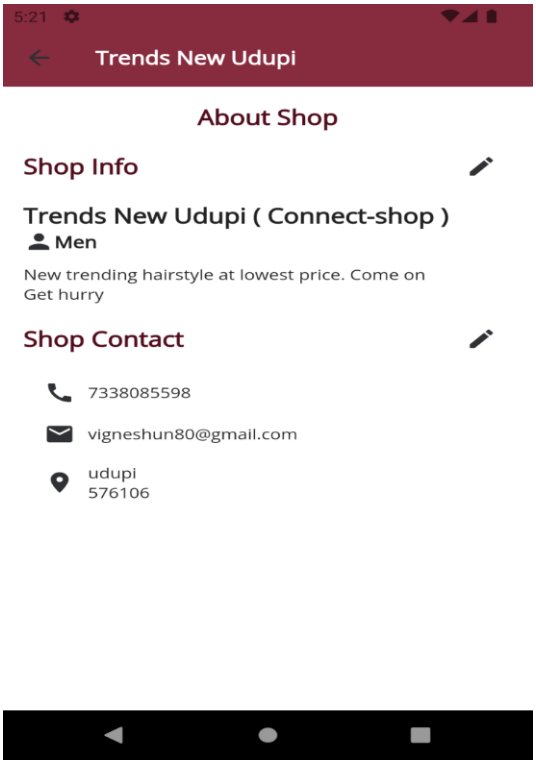
7.4.3.2 Manage user





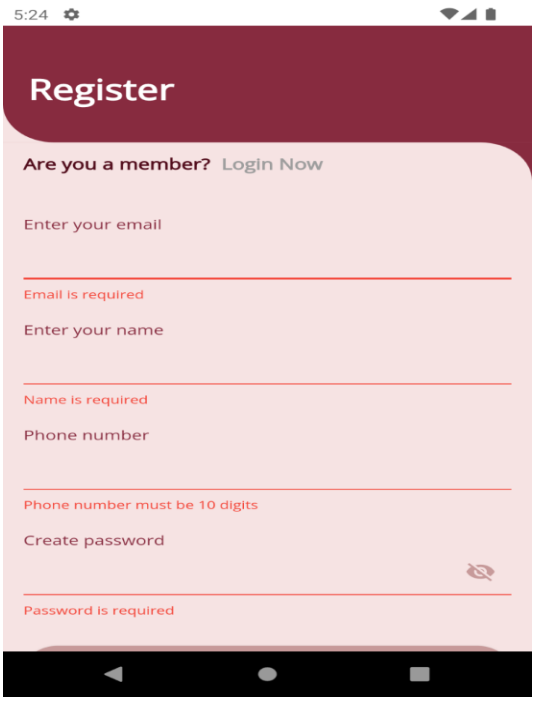
7.4.3.3 Manage shop



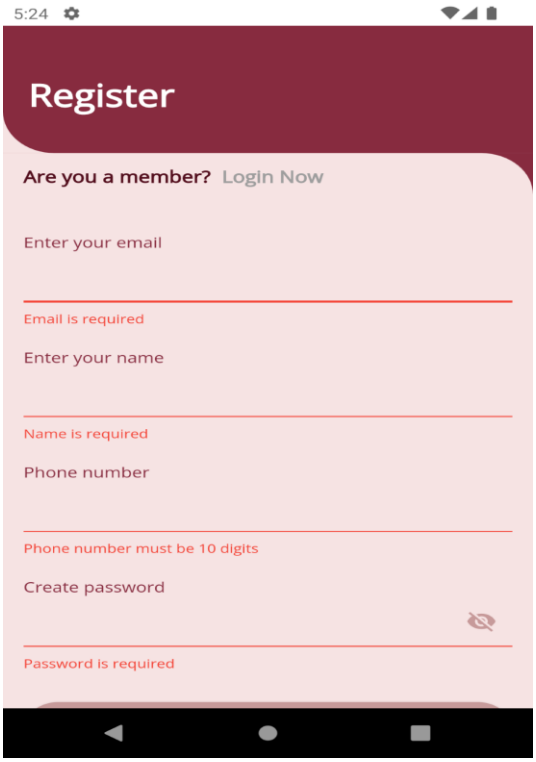


7.5 Validation

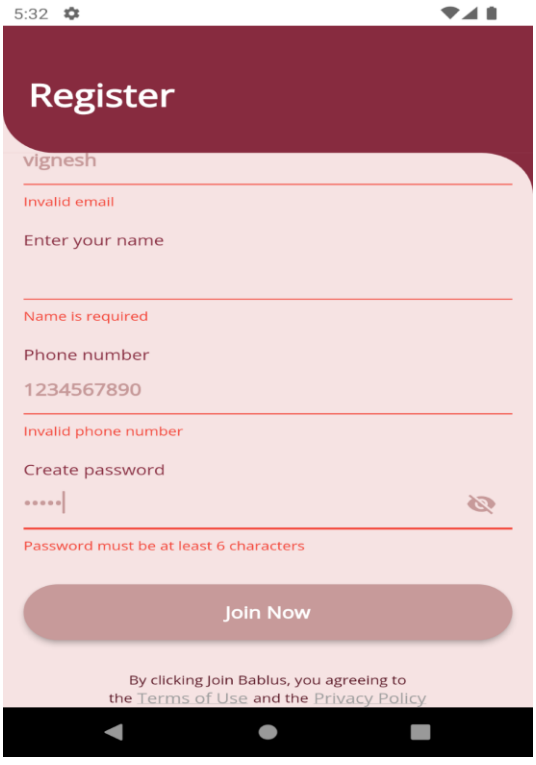
7.5.1 User Register required fields



7.5.2 Invalid mail and phone

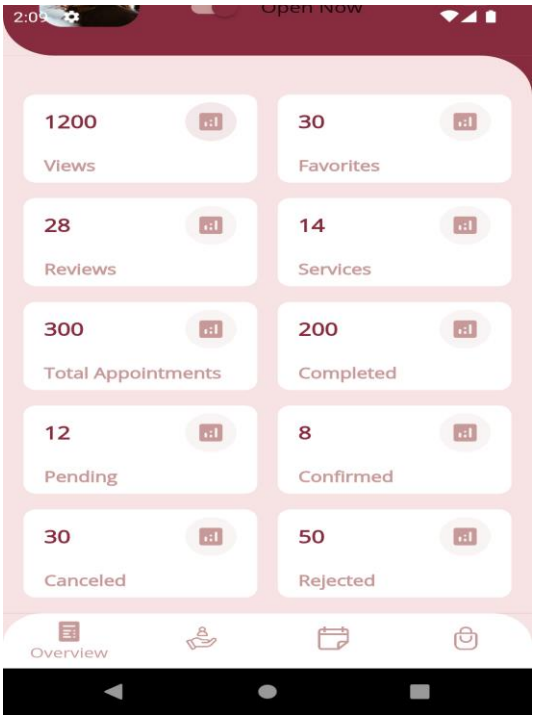


7.5.3 Password validation

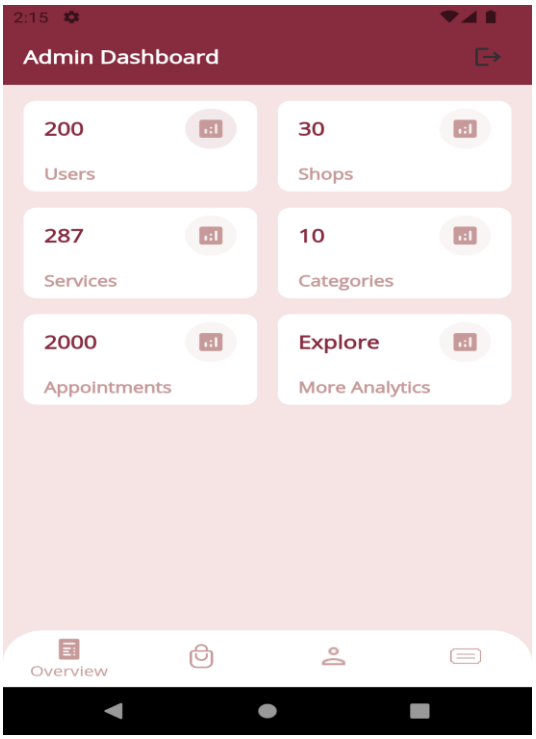


7.6 On-screen reports

7.6.1 Shop reports

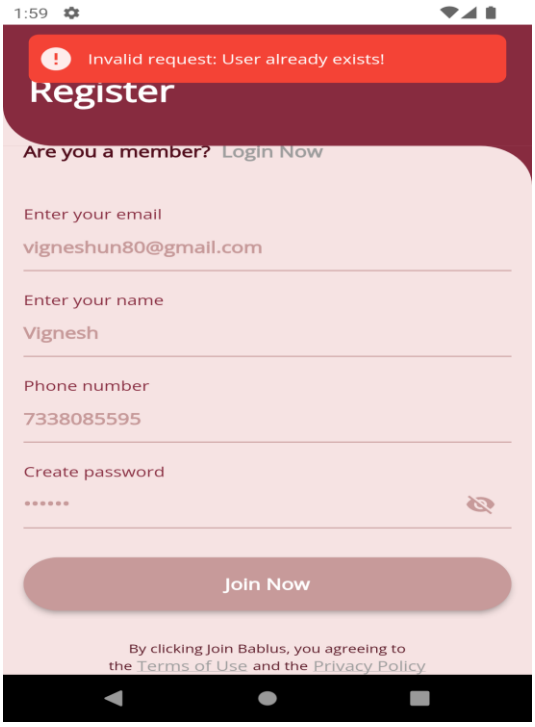


7.6.2 Admin reports

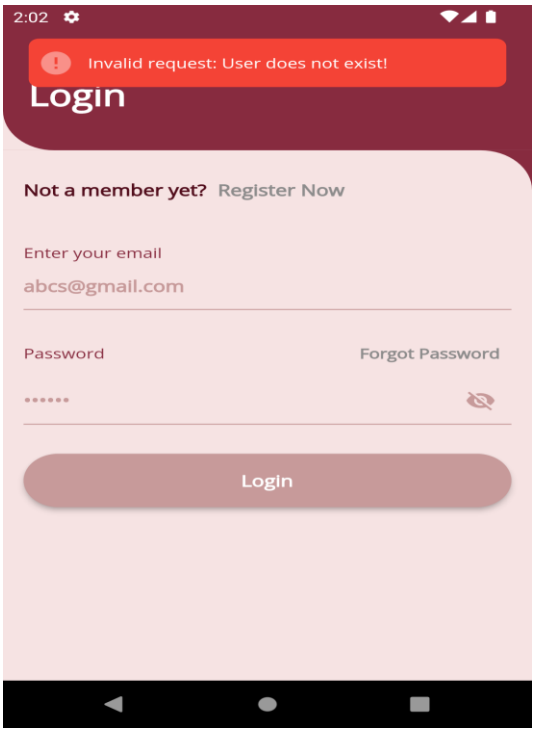


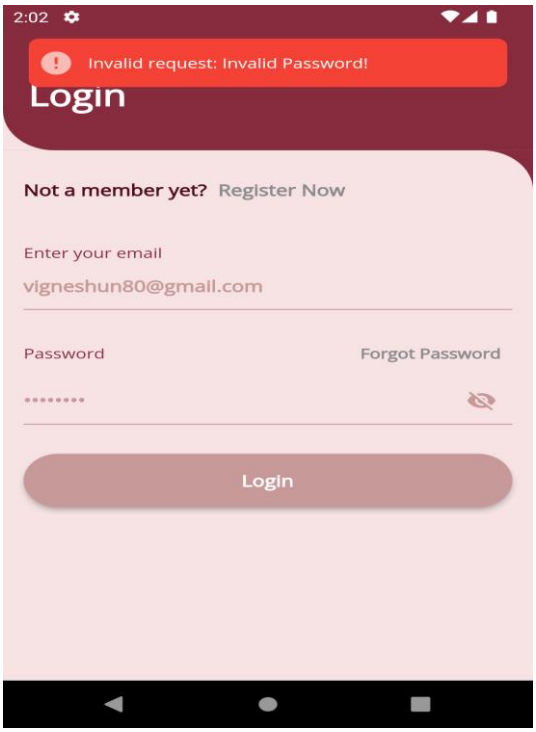
7.7 Error messages

7.7.1 User already exist

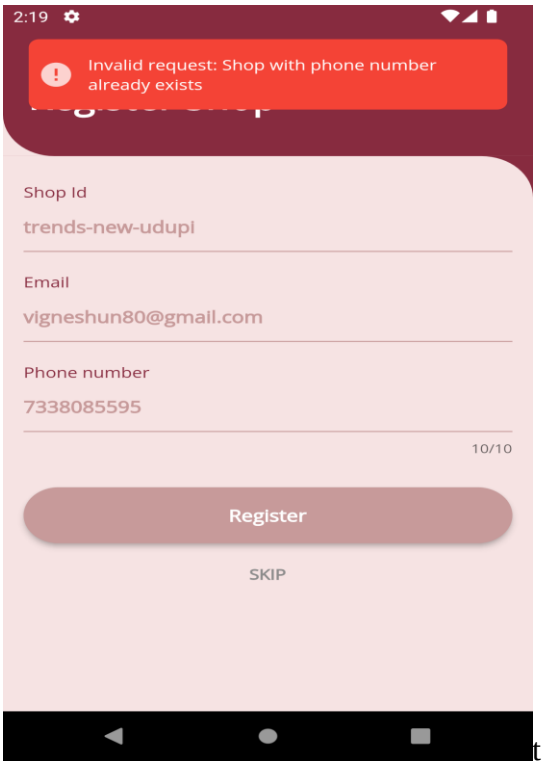


7.7.2 User not exist or invalid password while login





7.7.3 Shop already exist



t

8. Testing

8.1 Introduction (brief write-up about Software Testing)

Testing is a process of detecting errors. Testing is a very important role in quality assurance and for ensuring the reliability of the software. The result of testing is mainly used during maintenance.

8.2 Test Reports

8.2.1 Unit Testing

Unit testing is the smallest unit of testing that focuses on verification. Using concept of detailed design, the process of specification testing is carried out to uncover the errors within the boundary of the modules. All modules must be successful in the testing before the integration test begins.

Test case ID	**01**
Title	User Login
Purpose	Testing Login
Test data	E -mail address(Valid email address ,invalid email address ,empty field) Password (Valid password ,invalid password ,empty field)
Expected output	1.sucessfull login 2.login unsuccessful
Validation	1 Invalid email address.

	2. Password must have atleast 8 character.
Sample Test data	

E-mail address	Result
msupreetha2@gmail.com	Taken as valid email
Sh	Invalid email address.
Empty field	Invalid email address

Password	Result
9164141168	Taken has valid password
918478	Password must have at least 8 characters.
Empty field	Password must have at least 8 characters.

Test case ID	**02**
Title	User Registration
Purpose	Testing Registration
Test data	E -mail address(Valid email address ,invalid email address ,empty field) Name (valid name, invalid name, empty field) Phone (valid phone, invalid phone, empty field) Password (Valid password ,invalid password ,empty field)
Expected output	1.Registration successful 2. Registration unsuccessful
Validation	1 Invalid email address. 2.Invalid phone number. 3.Email already exits. 4.Phone number already exits. 5. Password must have at least 6 characters. 6.Name should have at lest 3 characters.
Sample Test data	

E-mail address	Result
msupreetha2@gmail.com	Taken as valid email
Sh	Invalid email address.
Empty field	Invalid email address

Password	Result
9164141168	Taken has valid password
91847	Password must have at least 6 characters.
Empty field	Password must have at least 6 characters.

Phone	Result
9342607939	Taken has valid phone number
91847	Phone number should be 10 characters.

	Empty field: Phone number should be 10 characters. Name / Result Karthik: Taken has valid name na: Name must have at least 3 characters. Empty field: Name must have at least 3 characters.

Test case ID	**03**
Title	Send forget Password OTP
Purpose	Testing Send forget Password OTP
Test data	E -mail address(Valid email address ,invalid email address ,empty field)
Expected output	1.OTP sent Successfully 2.Email is not registered.
Validation	1. Enter valid email.
Sample Test data	E-mail address / Result msupreetha2@gmail.com: Taken as valid email Sh: Invalid email address. Empty field: Invalid email address

Test case ID	**04**
Title	Verify OTP and Change The Password
Purpose	Testing Forget Password
Test data	Rest Code (OTP) (valid Code, invalid Code, empty field). New Password (valid Password, invalid Password, empty field). Confirm Password (Valid password ,invalid password ,empty field).
Expected output	1.Password Changed successful 2. Invalid OTP. 3.OTP expired.
Validation	1 Password must have at least 6 characters. 2Reset Code must 6 characters. 3. Password is not Matching .

<table>
<tr><td>Sample Test data</td><td>

Reset Code (OTP)	Result
284567	Taken as valid OTP
23	Invalid email OTP
Empty field	Invalid email OTP

Password	Result
9164141168	Taken has valid password
91847	Password must have at least 6 characters.
Empty field	Password must have at least 6 characters.

Confirm Password	Result
9164141168	Taken has valid password
91847	Password must have at least 6 characters.
Empty field	Password must match.

</td></tr>
</table>

Test case ID	**05**
Title	Shop Registration
Purpose	Testing Shop Registration
Test data	E -mail address(Valid email address ,invalid email address ,empty field) ShopId (valid shopId, invalid shopId, empty field) Password (Valid password ,invalid password ,empty field)
Expected output	1.Redirect to shop contact detail screen. 2. Stay in that page only
Validation	1Invalid ShopId. 2.Invalid phone number. 3.Email already exits. 4.Phone number already exits. 5. Password must have at least 6 characters. 6.Shop name is required. 7.ShopId required.
Sample Test data	

E-mail address	Result
msupreetha2@gmail.com	Taken as valid email
Sh	Invalid email address.
Empty field	Invalid email address

Password	Result
9164141168	Taken has valid password
91847	Password must have at least 6 characters.
Empty field	Password must have at least 6 characters.

Phone	Result
9342607939	Taken has valid phone number
91847	Phone number should be 10 characters.
Empty field	Phone number should be 10 characters.

Shop Name	Result
New look	Taken has valid name
na	Name must have at least 3 characters.
Empty field	Name must have at least 3 characters.

Shop Id	Result
Newlook	Taken has valid name
New Look	Shop Id should not contain space
Empty field	Shop Id must have at least 3 characters.

Test case ID	**06**
Title	Shop contact details
Purpose	Testing Shop contact details
Test data	Address(Valid address ,invalid address ,empty field) Pin Code (valid Pin Code, invalid Pin Code, empty field)
Expected output	1.Shop Registered successful 2. Error Messages
Validation	1. Pin Code must have 6 characters. 2.Address should have at lest 3 characters.
Sample Test data	

Pin Code	Result
576211	Taken as valid Pin Code
23	Pin Code must have 6 digits
Empty field	Pin Code must have 6 digits

Address	Result
Udupi main bus stand	Taken has valid name
na	Address must have at least 3 characters.
Empty field	Address Required.

Test case ID	**07**
Title	Edit User Profile

Purpose	Testing User Profile Edit
Test data	Name (Valid name ,invalid name ,empty field) Phone (valid phone number, invalid phone number, empty field) City (Valid City ,invalid City ,empty field) Pin Code (Valid Pin Code, invalid Pin Code, empty field). Gender(Male, Female, Other)
Expected output	1.Redirect to user profile screen 2. Error Messages
Validation	1. Pin Code must have 6 characters. 2.Invalid phone number. 3.Phone number already exits. `
Sample Test data	

Pin Code	Result
576211	Taken as valid Pin Code
23	Pin Code must have 6 digits
Empty field	Pin Code must have 6 digits

City	Result
Udupi	Taken has valid name
Empty field	Address Required.

Phone	Result
9342607939	Taken has valid phone number
91847	Phone number should be 10 characters.
Empty field	Phone number should be 10 characters.

Name	Result
Karthik	Taken has valid name
na	Name must have at least 3 characters.
Empty field	Name must have at least 3 characters.

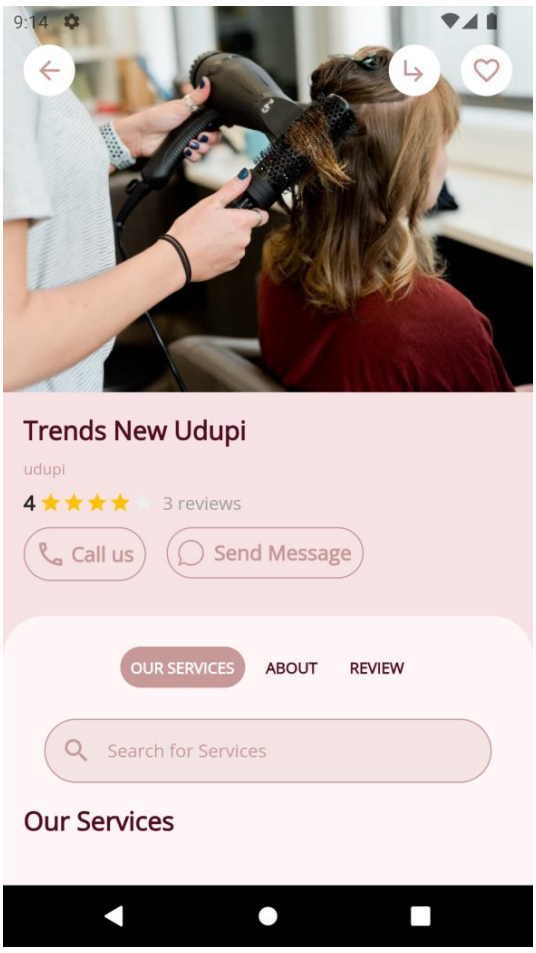
Test case ID	**08**
Title	Add or Update Category
Purpose	Testing Add or Update Category
Test data	Category Name(Valid Category Name, invalid Category Name, empty field)
Expected output	1.Redirect to Back 2. Error Messages
Validation	1. Category Name is required

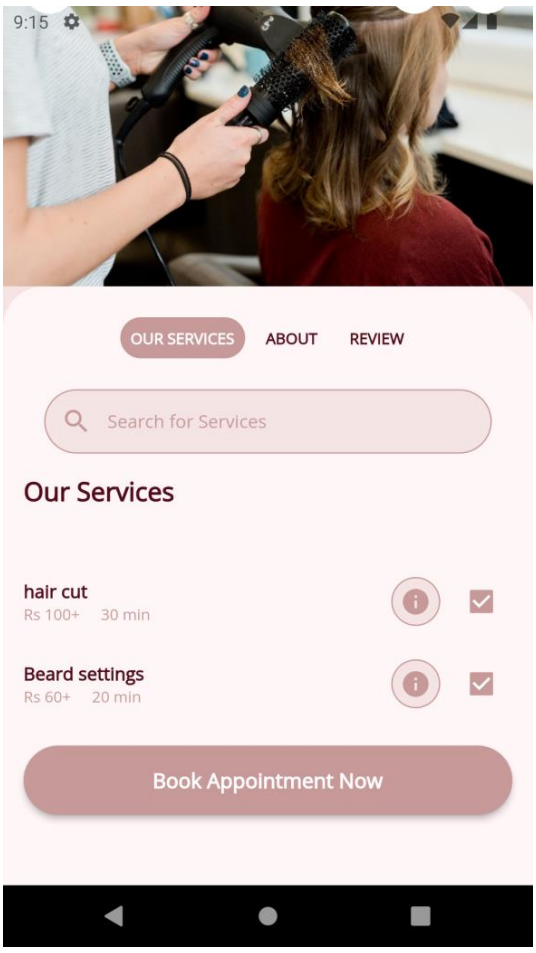
Sample Test data	<table><tr><td>City</td><td>Result</td></tr><tr><td>Beauty</td><td>Taken has valid name</td></tr><tr><td>Empty field</td><td>Category Name Required.</td></tr></table>

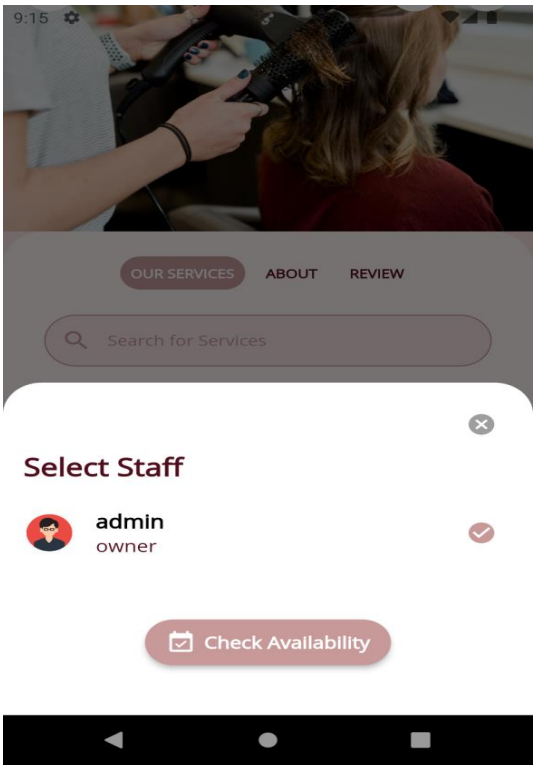
8.2.2 Integrate Testing

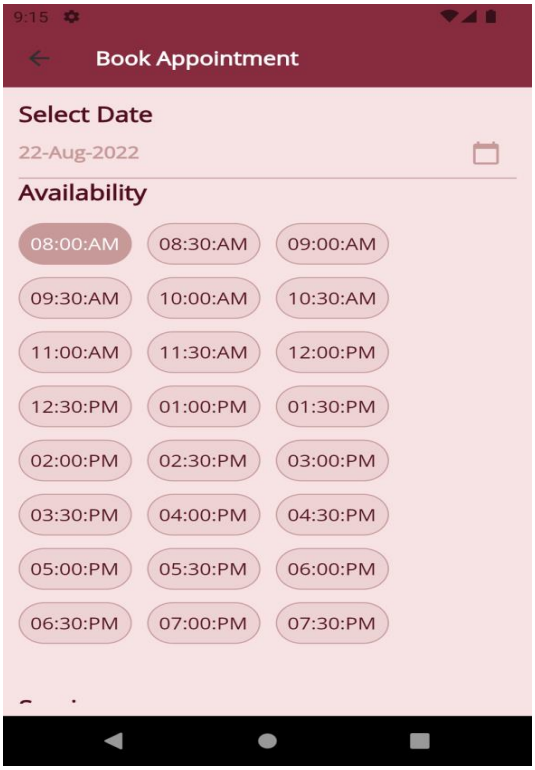
In this application developer test the programs as system. Software unit in a system are the modules and routines that are assembled and integrated to form a specific union.

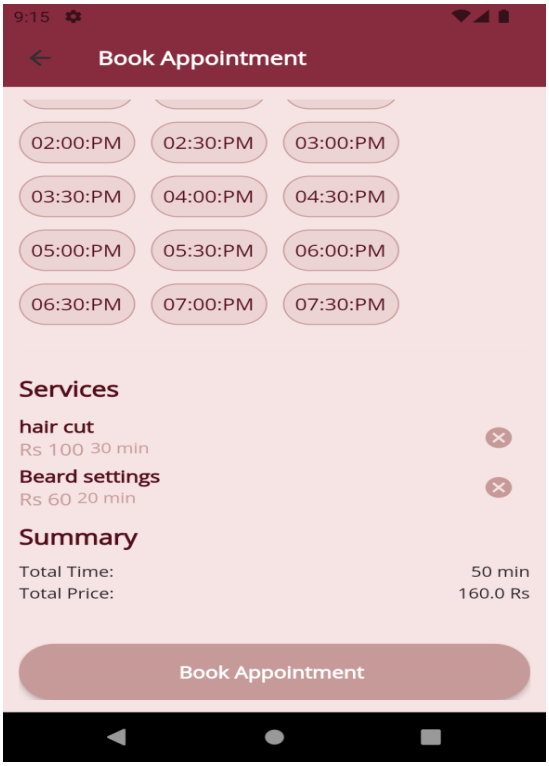
Booking Appointment

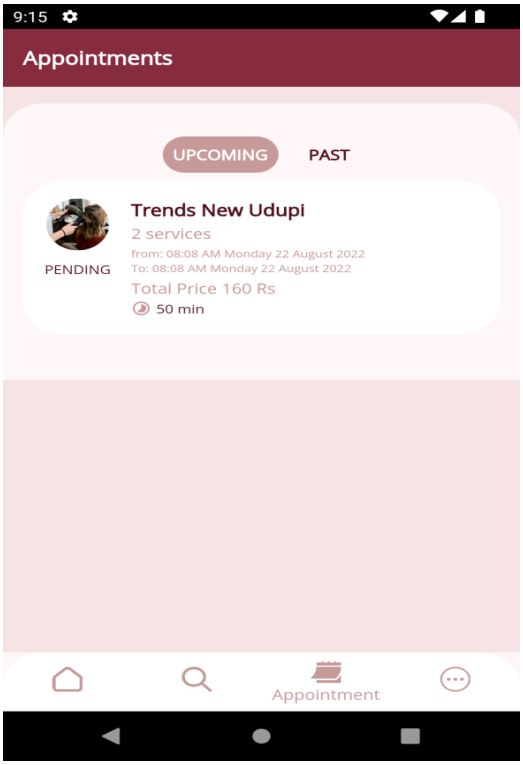





9:15
OUR SERVICES
ABOUT
REVIEW
Search for Services
Select Staff
admin
owner
Check Availability

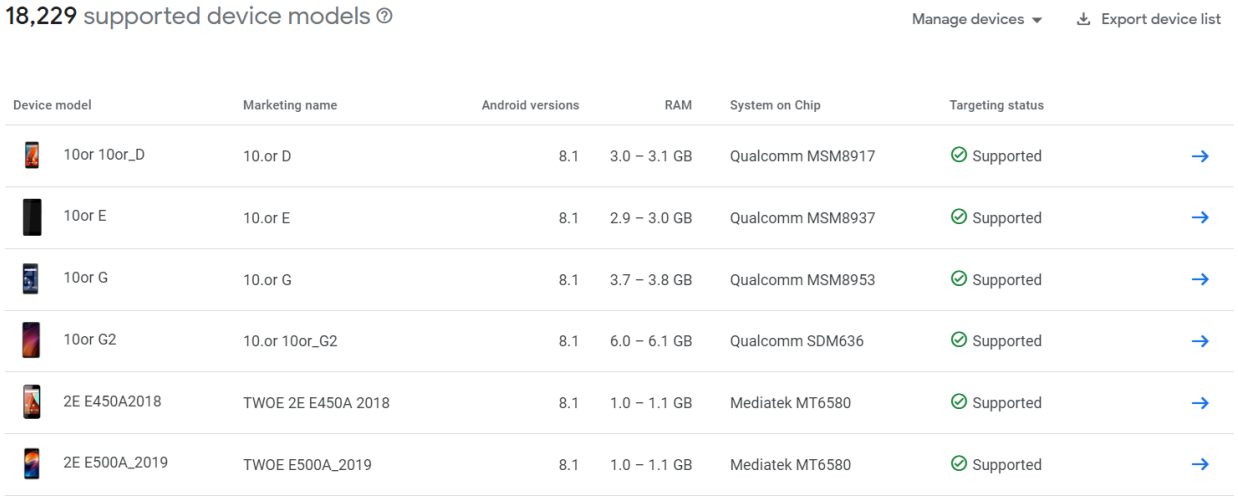

9:15
Book Appointment
Select Date
22-Aug-2022
Availability
08:00:AM
08:30:AM
09:00:AM
09:30:AM
10:00:AM
10:30:AM
11:00:AM
11:30:AM
12:00:PM
12:30:PM
01:00:PM
01:30:PM
02:00:PM
02:30:PM
03:00:PM
03:30:PM
04:00:PM
04:30:PM
05:00:PM
05:30:PM
06:00:PM
06:30:PM
07:00:PM
07:30:PM


9:15
Book Appointment
02:00:PM
02:30:PM
03:00:PM
03:30:PM
04:00:PM
04:30:PM
05:00:PM
05:30:PM
06:00:PM
06:30:PM
07:00:PM
07:30:PM
Services
hair cut
Rs 100 30 min
Beard settings
Rs 60 20 min
Summary
Total Time:
50 min
Total Price:
160.0 Rs
Book Appointment


9:15
Appointments
UPCOMING
PAST
Trends New Udupi
2 services
from: 08:08 AM Monday 22 August 2022
To: 08:08 AM Monday 22 August 2022
PENDING
Total Price 160 Rs
50 min
Appointment

8.2.3 System Testing

In system testing entire system is tested as a whole with all forms, code, modules and class modules. After the integration testing the whole of the system is tested in different environments and it is found that the system has performed well without giving any runtime error. Hence after the testing it is concluded that the system will work fine in all environment.



18,229 supported device models

Manage devices | Export device list

Device model	Marketing name	Android versions	RAM	System on Chip	Targeting status
10or 10or_D	10.or D	8.1	3.0 – 3.1 GB	Qualcomm MSM8917	Supported
10or E	10.or E	8.1	2.9 – 3.0 GB	Qualcomm MSM8937	Supported
10or G	10.or G	8.1	3.7 – 3.8 GB	Qualcomm MSM8953	Supported
10or G2	10.or 10or_G2	8.1	6.0 – 6.1 GB	Qualcomm SDM636	Supported
2E E450A2018	TWOE 2E E450A 2018	8.1	1.0 – 1.1 GB	Mediatek MT6580	Supported
2E E500A_2019	TWOE E500A_2019	8.1	1.0 – 1.1 GB	Mediatek MT6580	Supported

Conclusion

The project work title “Bablus – A online saloon booking App” has been successfully designed and developed and tested. User friendly interface are added to make it very easy user interactive application. The project is tested successfully and report is generated.

The modules have fulfilled by the entire objectives identified. The project have been developed in attractive user interactive fashion. So, the user with common development knowledge about the computer can handle it very easily. The book can be easily upload and enrolled to the app. So it is a user friendly app. The Project has produced all the reports required by the user. The module has fulfilled all the objectives identified and supports interactive user-friendly interface.

Limitations:

- Requires internet connectivity to use the application
- Shop owner should be active in the application

Scope for enhancement (future scope)

- Google map integration for calculating distance and direction
- Payment Gateway Integration using RazorPay
- Quick chat option between the user and shop

Abbreviations and Acronyms (list)

- ✓ SRS - Software Requirement Specification
- ✓ CFD - Context Flow Diagram
- ✓ DFD - Data Flow Diagram
- ✓ CFD - Control Flow Diagram
- ✓ ER diagram - Entity Relationship Diagram

Bibliography / References (list in specified format)

- An integrated approach to software engineering book- Pankaj Jalote
- https://docs.flutter.dev
- https://stackoverflow.com
- Youtube.com