

1. Introduction

1.1 Company Profile

Company Name: Lexicon Networks India Pvt Ltd

Address: 2nd Floor, Sanket Apartments, 3/9 Yashshree Colony,
Cummins College Road, Karvenagar, Pune 411062

Founded: 2008

Industry: Information Technology & Services

Official Website Link: www.lexiconnetworks.com/

1.2 Abstract

Rose Petals Beauty Salon is basically an Android application that is a major help to salons. Because they have many customers and many customers visit salons on different days, they cannot store their data manually or paperless. So, this application is helpful there. This application provides services of Salon to customers. Therefore, it creates interaction between Salons and customers. Also, by using this application we provide the best features to our users. In day to day our life is going to change mean it travel in digital way. Therefor we need some digital platform to easier our life. This application is one of the parts of this digitalization. Now a days every businessman stores their customers details and provides the best services. That one of the businesses is hair salon. I developed this app in android. This app provide service for hair salon business to store their customers details and also this is reminded customer to cut hair within his time period. This app helps to improve relations between customer and businessman.

Rose Petals Beauty Salon is a mobile-based salon app with appointment scheduling functionality. It connects clients, salons, and stylists in an online community allowing users to browse salons and stylists, and book or cancel appointments. Users can also write and read

reviews of products and stylists. Salons can specify the stylists that work at their salons, as well as the services they offer. Salons can also book appointments for customers and can view and print schedules in convenient formats.

My project will use Android and Firebase to back the interface with strong database functionality. For appointment scheduling, Beauty Hub app will integrate mobile calendar as a backend database for appointments as well as a front-end scheduling interface. This project will target the major play store app as the initial platform for the Beta version.

1.3 Existing System and Need for System

▪ Existing System

- In existing system some hair salon businessman cannot store their customers details therefore they cannot able to communicate between them this happens they lose some customers.
- Also, they cannot able to provide their best services to their customers.
- The existing system cannot provide the booking of time slots.
- When we need to book any time slot, we required to call that specific hair salon and then ask them to book my slot.
- Customers lose their valuable time
- Customer can not able to find their offers and discounts

▪ Need for System

Its customers do not have a proper way to make an appointment other than making a call or visiting the Salon premises. Salon owners, employees, and customers need to keep reminders on their mobiles over appointments. The salon owner and her employees maintain a diary to note down the appointment details. Service details of the salon are written on the paper; which always leads to misplacements. The owner needs to write all the service details on a new paper once it gets misplaced or updated.

Once payment has been made, the customer will receive a handwritten receipt (from the manual receipt book), and the cashier is keeping a copy of the same receipt. There is a higher risk of misplacing the receipt copies. The owner has no proper way to manage her employees and clients.

Generally, Salon hikes a lot in charges of their services. Even their offer is displayed for a limited number of days. Even there is a lot of rush in the salons, which consumes more time.

1.4 Scope of System

- Customers can book their time slots for hair cutting.
- Hair salon businessman can directly provide their best services to customers. For example, their best offers, Holidays, etc.
- Customers can reduce their time for calling hair salon and booking of slots.
- In future hair salon can able to sell their best product on this app.
- Providing the facility to registering Salon staff and maintaining their details.
- Providing the facility to registering regular Customers and maintaining them details.
- Facilitate appointment handling.
- View appointments leaves and holidays through an event calendar.
- Handling Salon Services along with their respective prices, hours etc.
- Providing Customer Payment handling option.
- Generating invoices through the system.
- Generating reports to support the higher managerial decisions.

1.5 Operating Environment - Hardware and Software

- **Server-side requirement**
 - Hardware Requirements:
 - Processor: Intel core i3 and above
 - RAM: 4 GB
 - HDD: 120GB
 - Software Requirements:
 - Operating System: Windows 10
 - Database: Firebase
 - Front End: Android
 - Server-Side Script: Java
 - Software Development Tool: Android Studio Code
- **Client-side requirement**
 - Hardware Requirements:
 - Android Version: above 7
 - RAM: 2 GB

1.6 Brief Description of Technology Used

- **Java**
 - For management of data.
 - To pass backend data to the frontend side.
 - For validation
- **Android**
 - For user interface (Frontend)
- **Android Studio**
 - To write a better code

- To manage the structure of files in project
- **XML**
 - For frontend design
 - To store values in projects like string, color, values, etc.
- **GitHub**
 - For storing or backup project files

1.6.1 Operating systems used (Windows or Unix)

- **Windows**
- **Android**

1.6.2 RDBMS/No SQL used to build database

- **Firebase**
- **SQLite**
- **JSON**

2. Proposed System

2.1 Study of Similar Systems (If required research paper can be included)

Title: Beauty and Aesthetics – A study of the Professional Hair Care Industry

Abstract

This paper describes about the Rose Petals Beauty Salon application developed using Android Studio new version. It also includes about the Single Platforms on which development of android platform application can be done. Rose Petals Beauty Salon Application is an Android Application which is builds in Android Studio 8.0.1. Android Studio is an official integrated development tool or environment for Google's Android operating system. It is builds on JetBrains' IntelliJ IDEA software. The main motive of building Cab Application is to provide employment and also make drivers, owners and customer's life easy. So basically, we are trying to connect peoples (customers and Owners of the Salon) can be mutually benefited. In this Application number of services available, so the customer can easily select the services, date of service, time of service etc.

Background: Beauty is a subject which is not easy to grasp especially as it is perceived differently. In advertising it is expressed through aesthetic messages and images which we relate to symbolic and social meanings. The professional hair care industry in Sweden serves as a good example where the creation of aesthetic experience influences consumer purchasing behaviour.

Purpose: The purpose of our thesis is to study how consumers' subjective view on beauty and aesthetics can be influenced by the professional hair care industry and how market is created for products which mainly satisfy emotional needs rather than fulfil utilitarian function.

Research Method: In our study we have applied an abductive research method approach. The empirical findings were based on 3 interviews with P&G Salon Professional representatives and 15 end consumers combined with a survey, conducted in 25 hair salons in the city of Pune.

Conclusion: Consumers act in a socially constructed world in which products are shaped around impulse and feeling rather than their rationality. When buying a professional hair care product people receive much more than the actual product itself. People improve not only physical appearance but they also feel beautiful from within. While the utilitarian function is basically the same in both professional hair care and retail products, the former contributes to higher degree of satisfaction.

Keywords: Beauty, aesthetics, hair care, purchasing behaviour, marketing

2.2 Feasibility Study

- **Technical Feasibility:**

- User Friendly
- We provide service to customers to book their times slots
- Anyone can create an account on this app and get the best services

- **Economic Feasibility:**

Salons can add their time periods means they can add their salon opening time and closing time, according to this time system automatically creates slots. We do not charge any fee from customers for slot booking.

- **Operational Feasibility:**

Salons can be able to add their worker's details on this app. Before logging into the app, it is mandatory to create an account first. By the creation of an account, the app asks users if it is a men's Salon or a women's Salon. We provide separate services for men's and women's salons.

2.3 Objectives of Proposed System

- Customers will not need to search for makeup artists for wedding ceremonies and can directly go to the app to find a good Salon and find a good makeup artist. And can easily book makeup artists.
- To enhance interaction between customers and salon.
- To get better service for customer from salon

2.4 Users of System

1. Salon Owner:

- Add their customers details in their database.
- Reminds customers for their hair cut and other services.
- Provide best offers to their customers.
- Can recommend a new hair style and beauty tips.
- Recommended new courses

2. Customer:

- Can book their time slot for haircut.
- Can able to compare price of haircut.
- Can able to pay their hair cut payment.
- Can give feedback to the system
- Can give review of the product, service

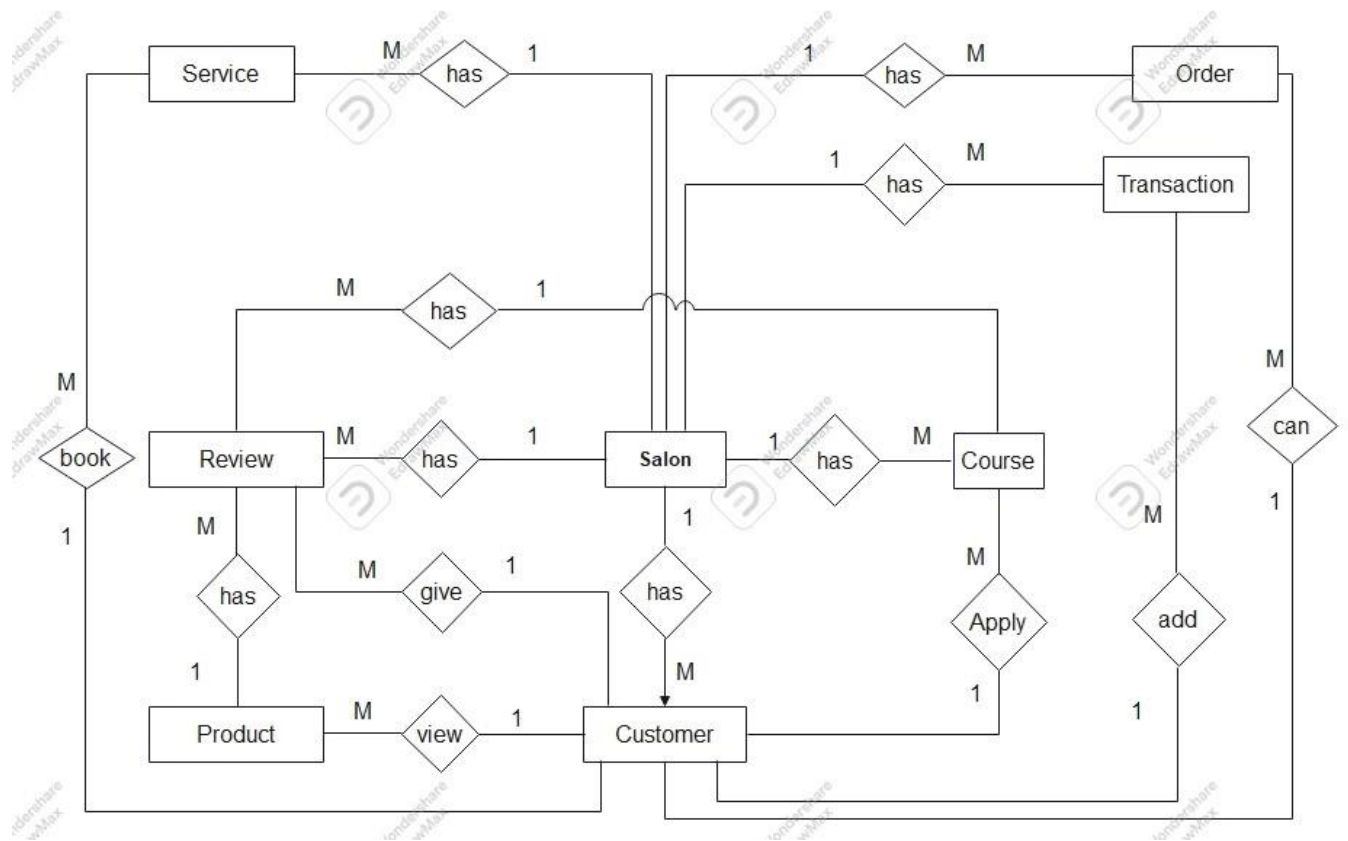
3. Analysis and Design

3.1 System Requirements (Functional and Non-Functional requirements)

- **Functional requirement**
 - Authentication
 - Business core
 - Transactions, checkouts
 - Authorizations
 - History data
 - Checking real time market price
 - Getting advertisement

- **Non-Functional requirements**
 - Loading speed
 - Time taken to deliver server response
 - User response time
 - Data consumption limits
 - Displaying Salons in the area
 - Displaying customers of Salons
 - Processing transactions
 - Refresh transactions and customers listings every 5 minutes
 - Refresh QR code every second

3.2 Entity Relationship Diagram (ERD)



3.3 Table Structure

Table Name: Salon

Column Name	Data Type	Constraint
key	String	Primary Key
Name	String	NOT NULL
Owner Name	String	NOT NULL
Mobile	String	NOT NULL
Email	String	Unique Key
Password	String	NOT NULL
Opening Time	String	NOT NULL
Closing Time	String	NOT NULL
Address	String	NOT NULL
Photo	String	NOT NULL
Token	String	NOT NULL

Table Name: Customer

Column Name	Data Type	Constraint
key	String	Primary Key
Name	String	NOT NULL
Mobile	String	NOT NULL
Email	String	Unique Key
Password	String	NOT NULL
Gender	String	NOT NULL
Token	String	NOT NULL

Table Name: Service

Column Name	Data Type	Constraint
key	String	Primary Key
Type	String	NOT NULL
Name	String	NOT NULL
Price	number	NOT NULL
Service Time Period	String	NOT NULL
Review	number	NOT NULL

Table Name: Transaction

Column Name	Data Type	Constraint
key	String	Primary Key
Customer Name	String	Foreign Key
Customer Mobile	String	NOT NULL
Services	ArrayList<String>	NOT NULL
Products	ArrayList<String>	NULL
Expense	number	NOT NULL
Date	String	NOT NULL
Bill	String	NOT NULL
Customer_token	String	NOT NULL

Table Name: Product

Column Name	Data Type	Constraint
key	String	Primary Key
Name	String	NOT NULL
Description	String	NOT NULL
Price	number	NOT NULL
Photos	ArrayList<String>	
Review	number	NOT NULL

Table Name: Course

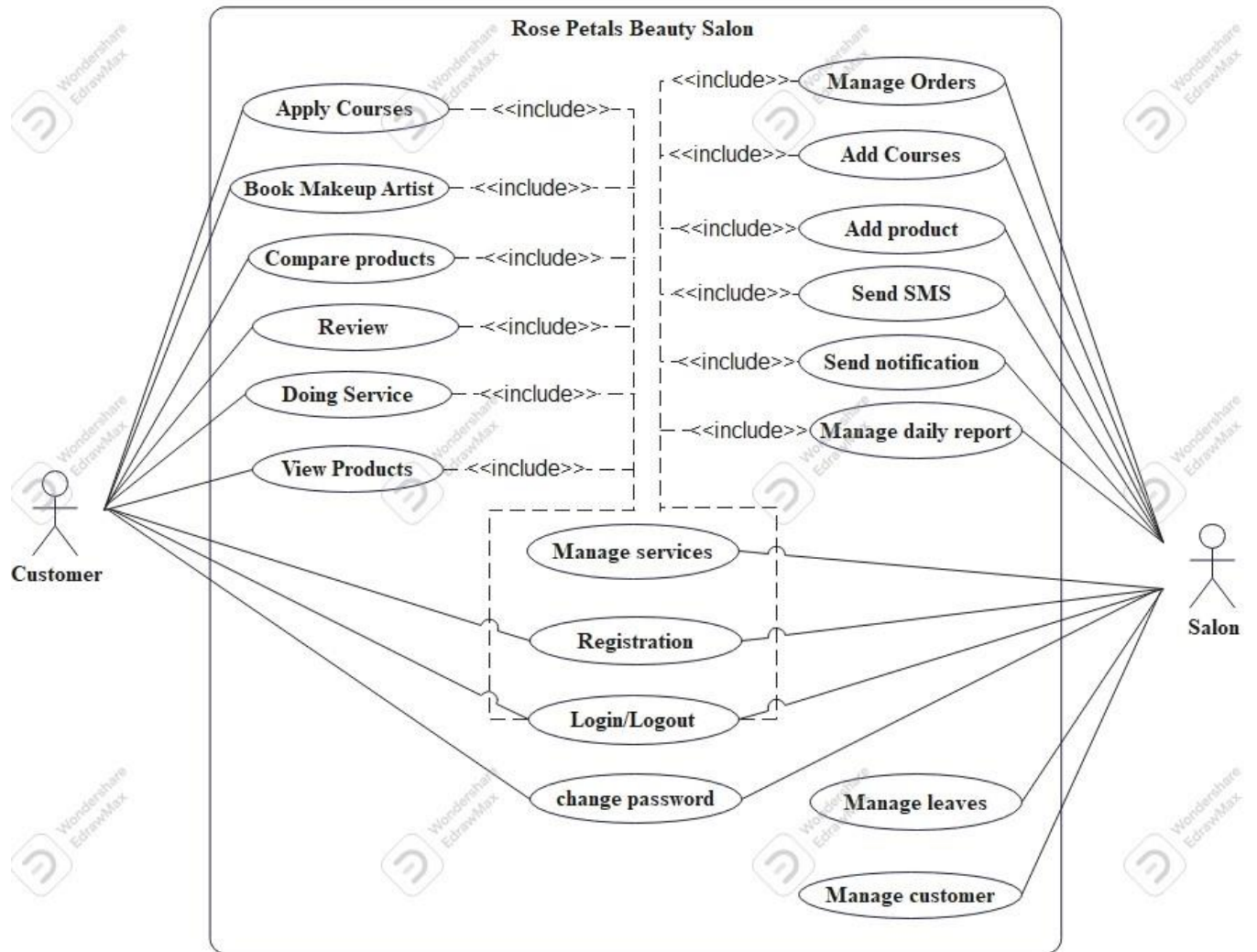
Column Name	Data Type	Constraint
key	String	Primary Key
Name	String	NOT NULL
Description	String	NOT NULL
Price	number	NOT NULL
Rating	number	NOT NULL

Table Name: Review

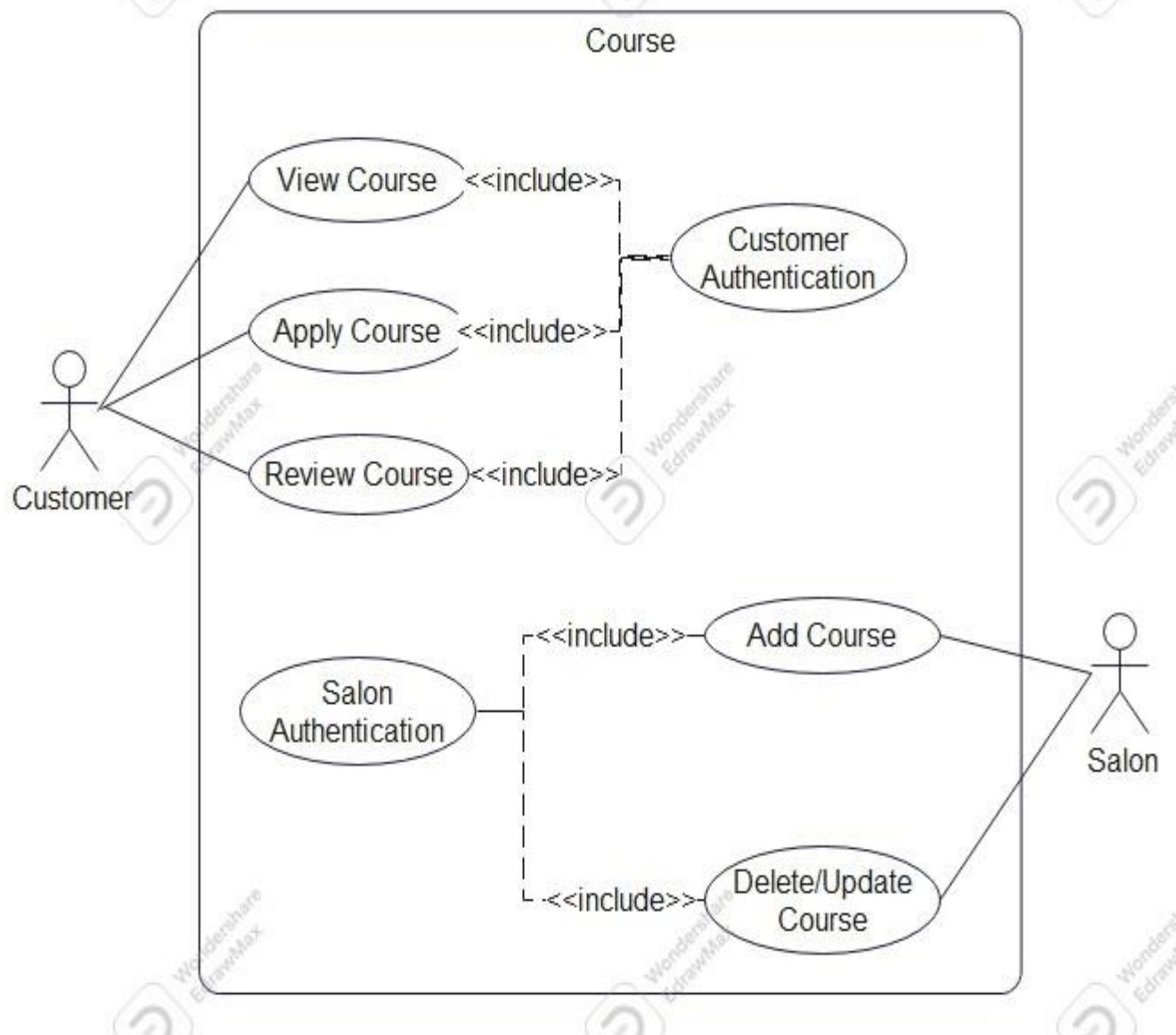
Column Name	Data Type	Constraint
key	String	Primary Key
Comment	String	NOT NULL
Rating	number	NOT NULL
Item key	String	Foreign Key

3.4 Use Case Diagrams

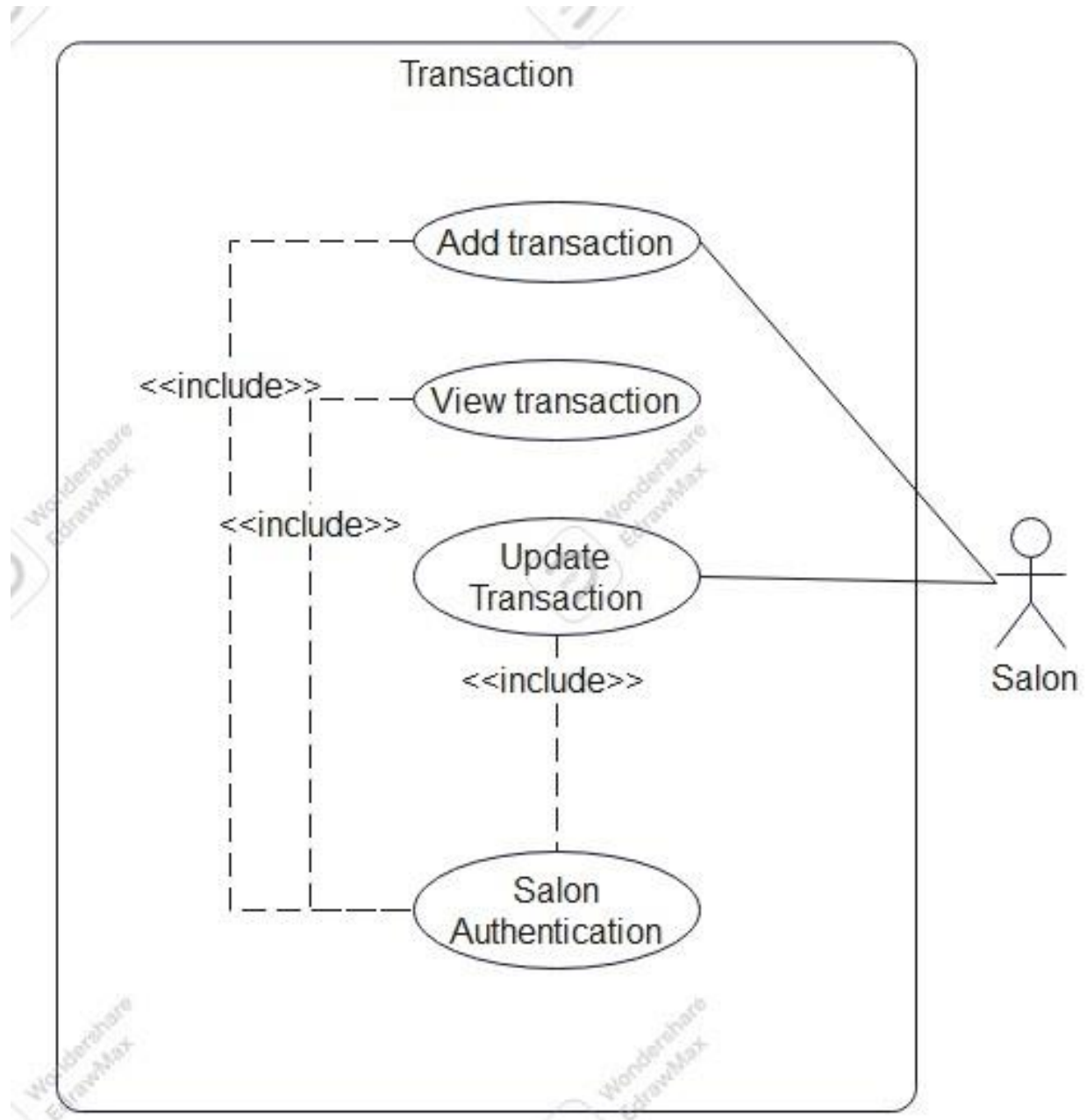
System: Use Case



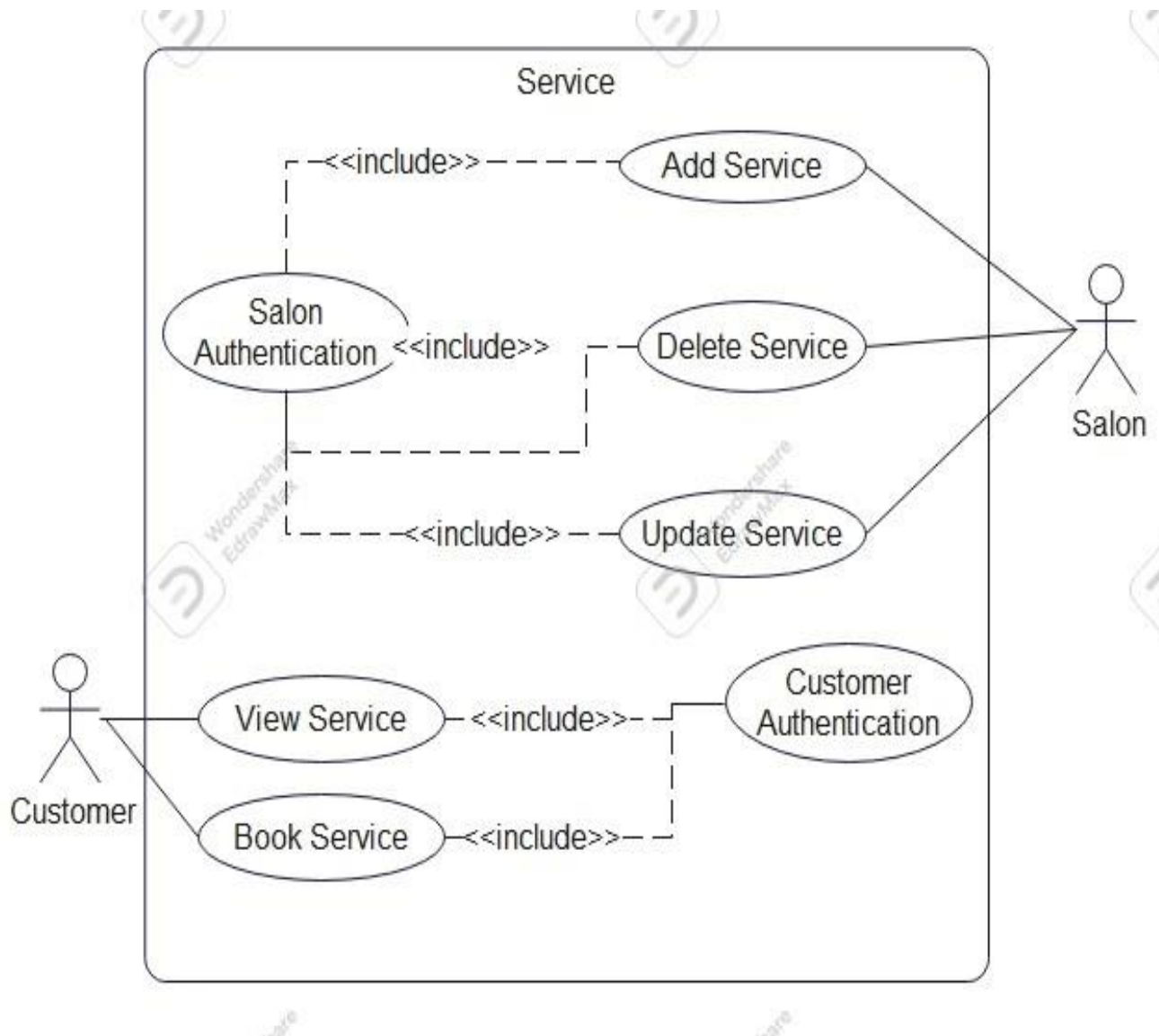
Course:Use case



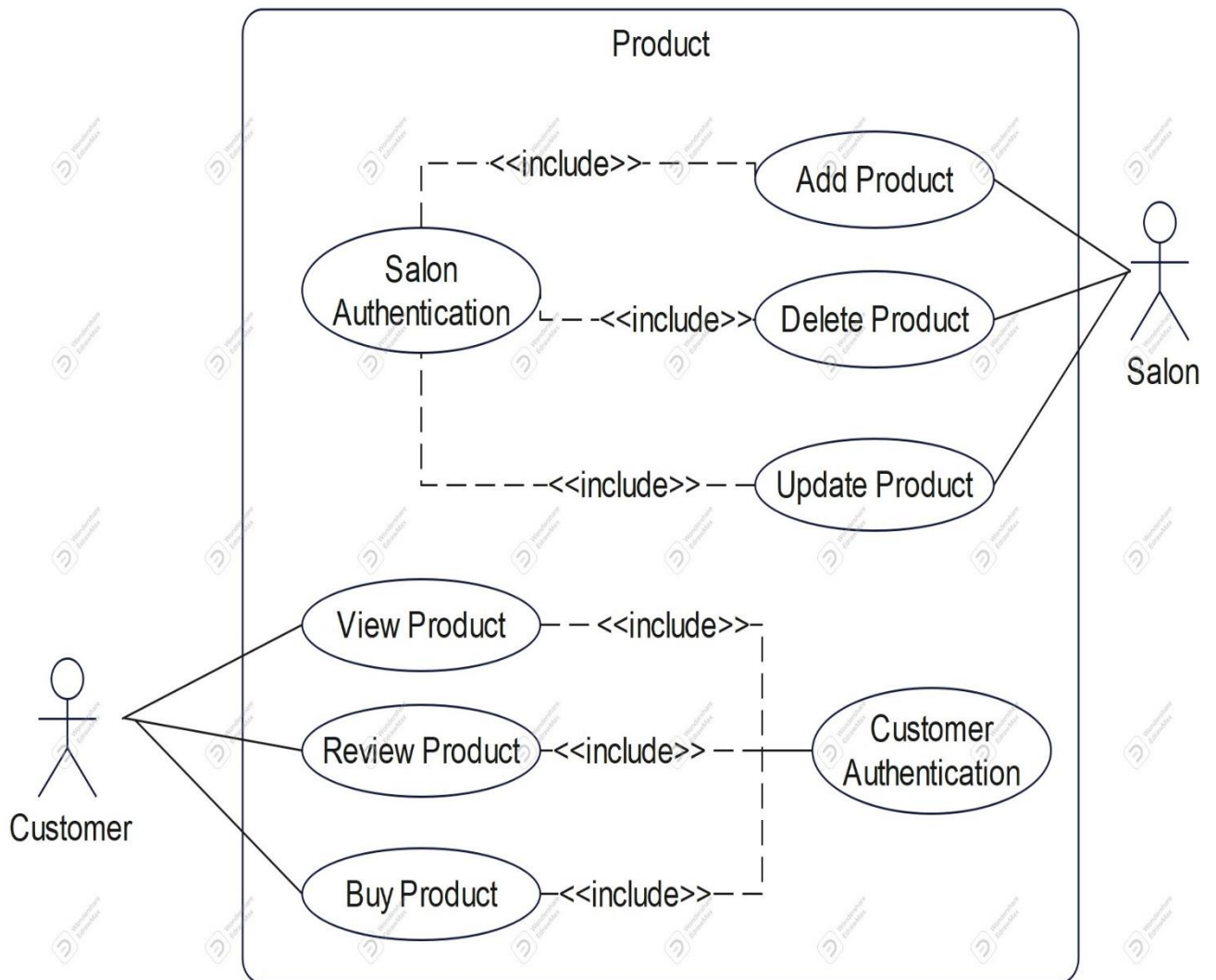
Transaction: Use case



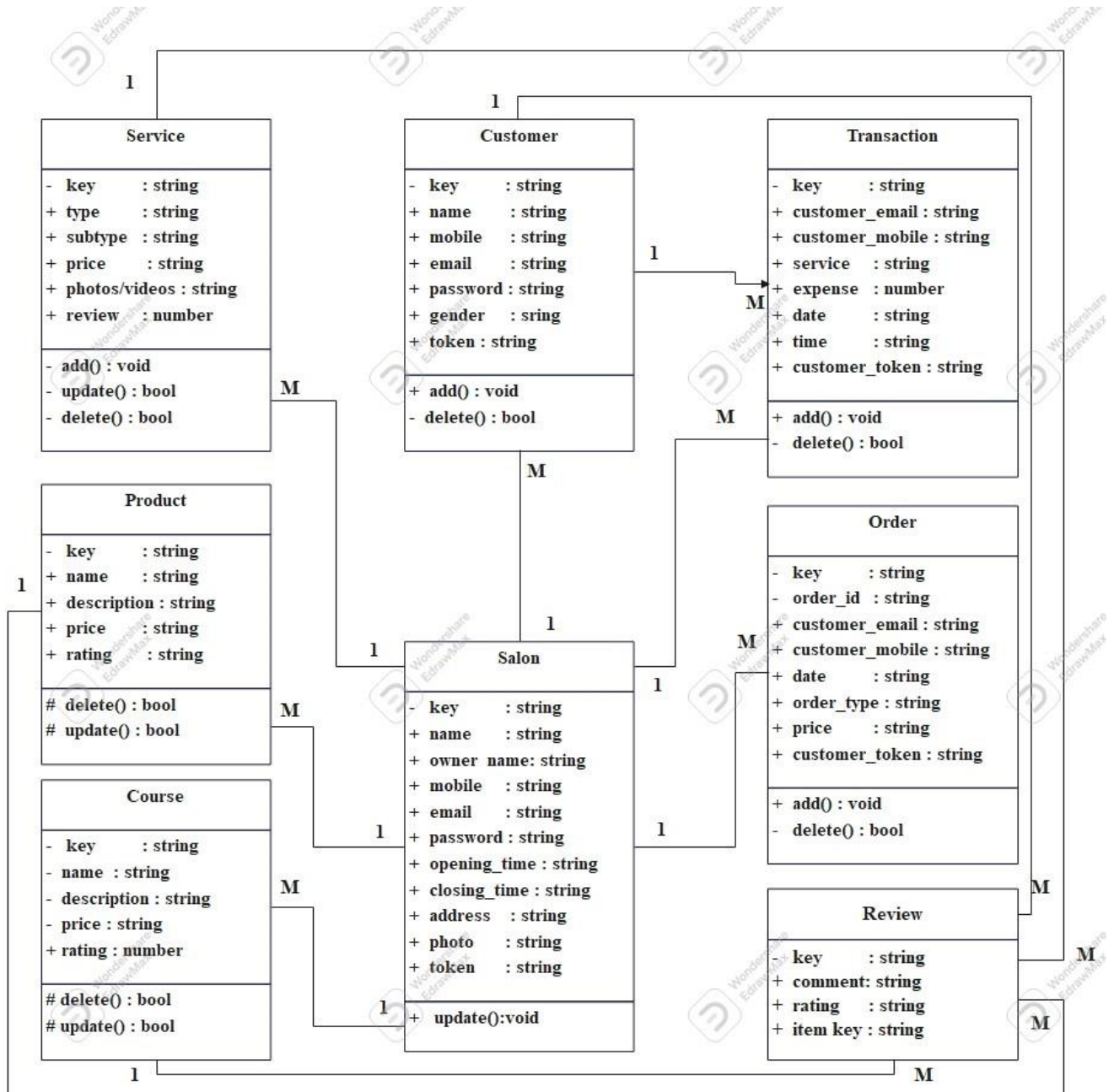
Service: Use case



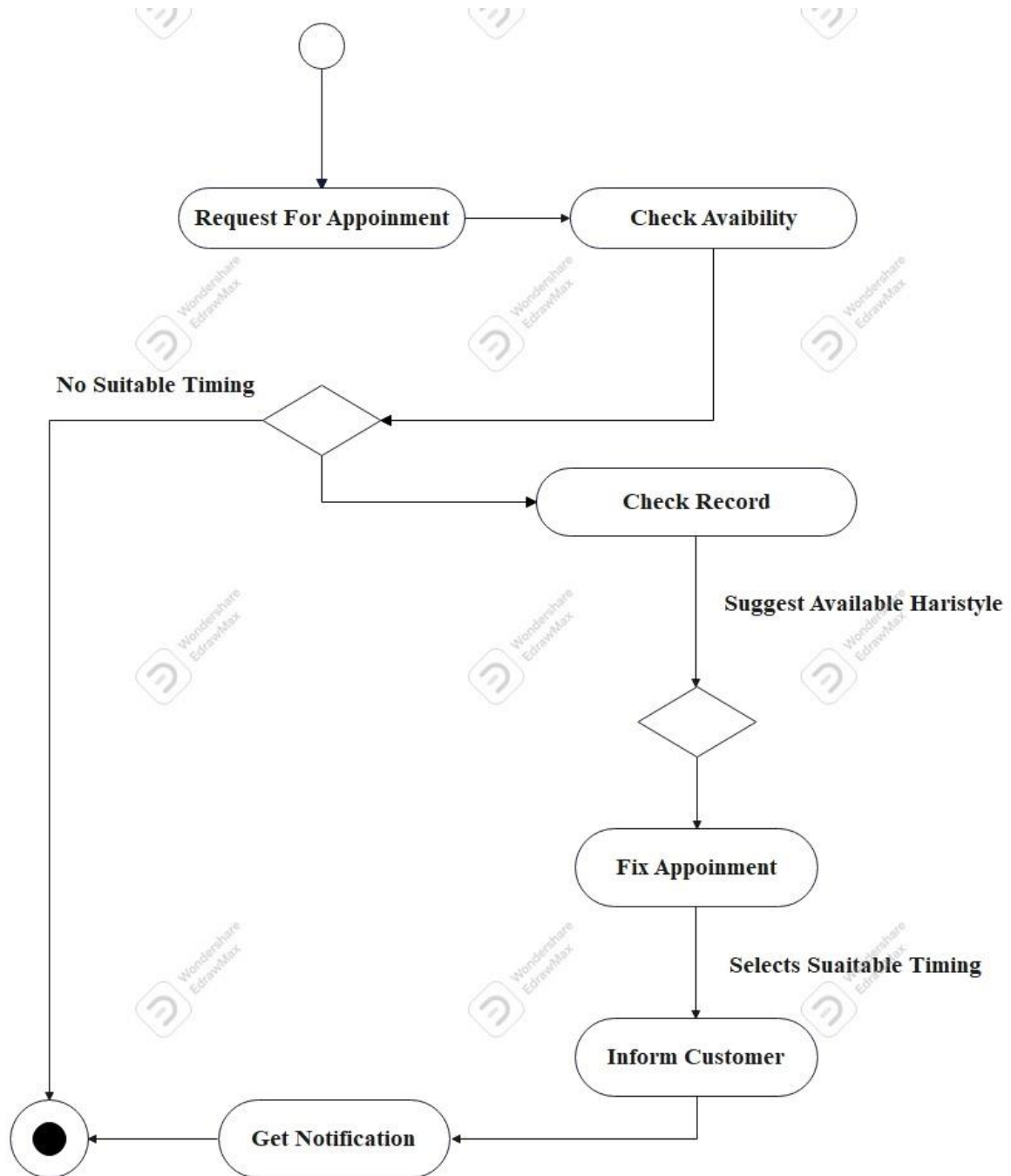
Product: Use case



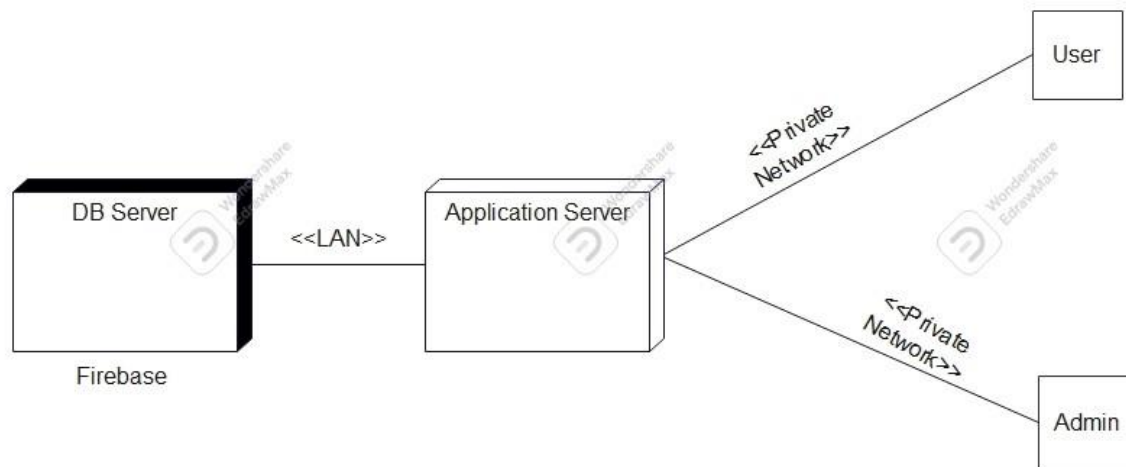
3.5 Class Diagram



3.6 Activity Diagram



3.7 Deployment Diagram



3.8 Module Hierarchy Diagram

Modules:

1. Services
2. Orders
3. Products
4. Tips and Suggestions
5. SMS Remainder
6. Courses
7. Reports
8. Review
9. Bill
10. QR Code

Functionality:

1.Services:

- Salon owner can add their many services like haircut, facials, makeup.
- Customers can view salon services, choose suitable service for them
- Salon owner manage salon services data like who cuts hair on which

dates and that's all.

- He can provide best offers to their customers according to their numbers of services.
- Salon owner can recommend a new hair style and other services.

2.Orders:

- Salon owner can manage their makeup orders, mehndi orders, etc.
- Customers can book makeup artist for their events or functions
- Salon owner can store total makeup orders details

3. Products:

- Salon owner can add their various products for example cosmetic products, some hair related products, trimmer and others electronic products related to beauty.
- All types of products which salon has to provides.
- Customer can able to view salon products.
- Customer can give review to the particular product.

4. Tips and Suggestions:

- Salon owner suggests the beauty tips and solutions for their customers
- Salon owner suggest customers for their services
- He can suggest beauty products according to customer's skin
- To provides best tips for customers
- Customers can get benefits of these tips and suggestions for beautiful look.

5. SMS Remainder:

- Salon owner notifies their customers for best services and special offers.
- Customer can get remainder for their services

- To notify customers for their services on time to time
- Salon owner can suggest some products for particular customers
- Salon owner can manage best services to regular customers using SMS

6. Report:

- Salon owner can generate daily report of salon like no of customers visited, no of products sell, etc.
- Generate yearly income report
- Salon owner can generate monthly customers report
- Salon owner can manage new customers according to report
- Customers can able to calculate their expenses for their beauty

7. Review:

- Customer can give the review of the salons.
- Customers can give review of the products also and check product quality according to others review

8. Bill:

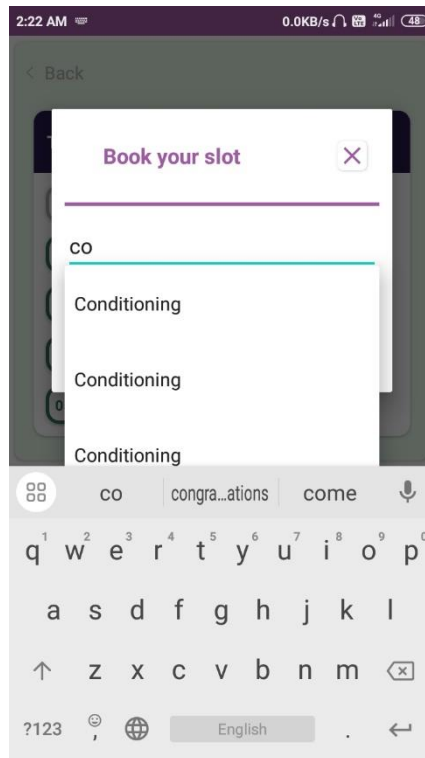
- Owner can generate the bill
- Customers can view their bills
- Bill have a payment QR code

9. QR Code:

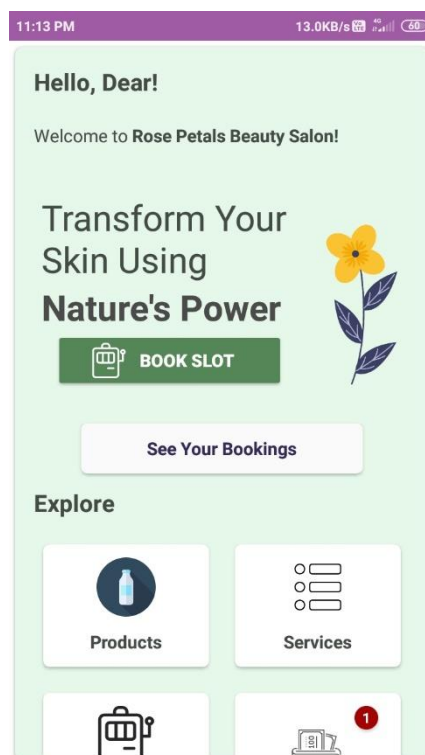
- Owner can generate QR code of services done by customers or purchased products
- Also, owner can generate payment QR code

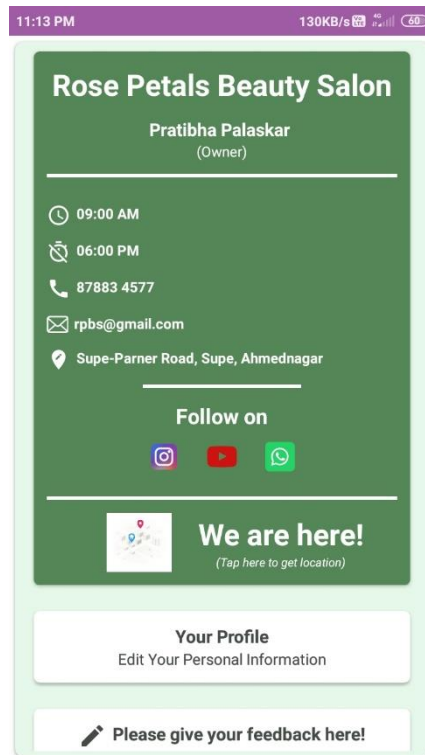
3.9 Sample Input and Output Screens

1. Book slot

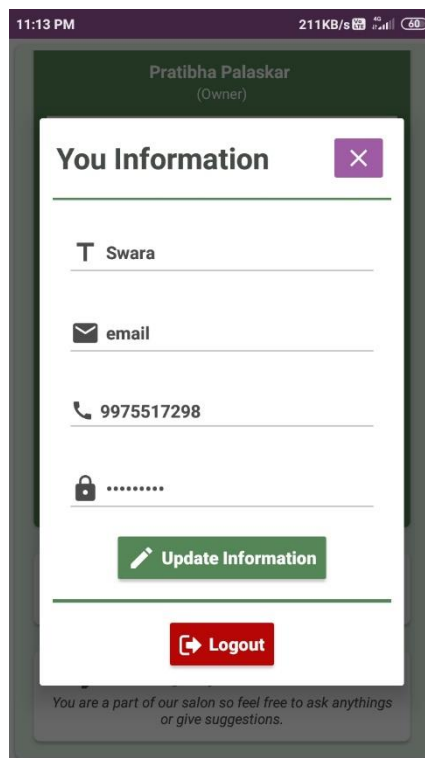


2. Customer dashboard





3. Customer information



4. Send feedback

11:13 PM 170KB/s

Pratibha Palaskar
(Owner)

09:00 AM
06:00 PM

Send Feedback ✕

Your name

Type comment here....

SUBMIT

Edit Your Personal Information

Please give your feedback here!
You are a part of our salon so feel free to ask anything or give suggestions.

5. Your slot

11:13 PM 152KB/s

✕

MONDAY 10 July

11:00 AM - 12:00 PM

Perm & Straightening

TOMORROW 09 July

09:00 AM - 10:00 AM

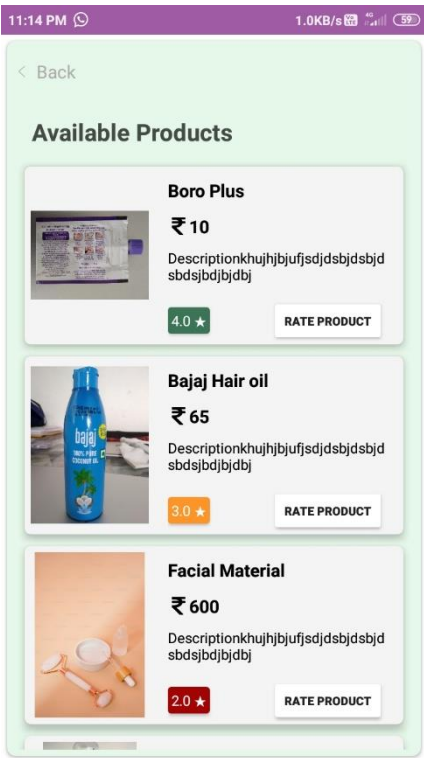
Swedish Massage

TUESDAY 11 July

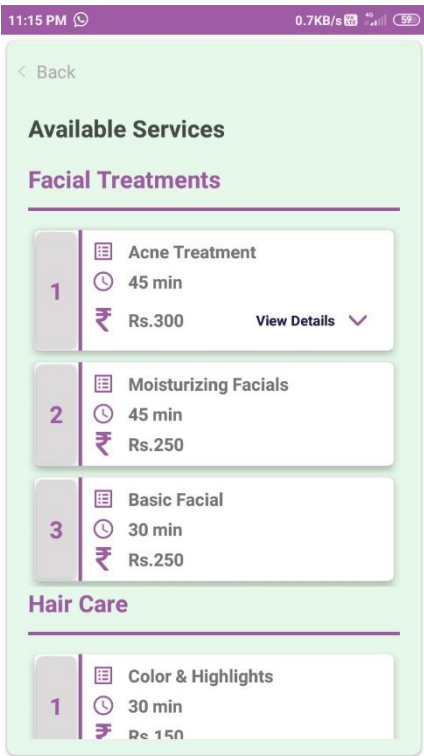
11:00 AM - 12:00 PM

Acne Treatment

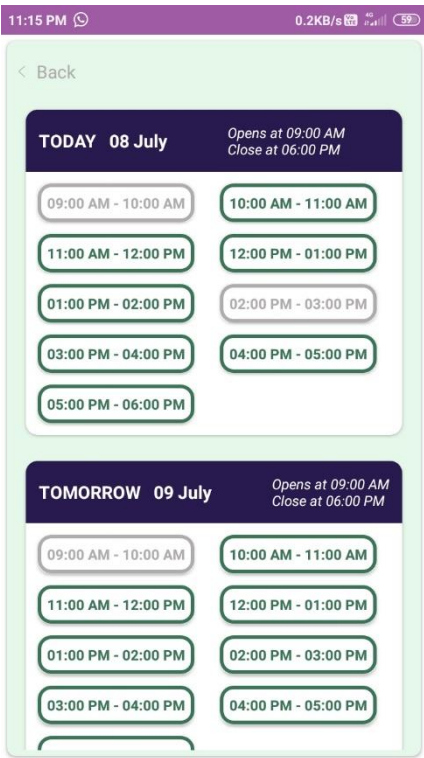
6. Available products



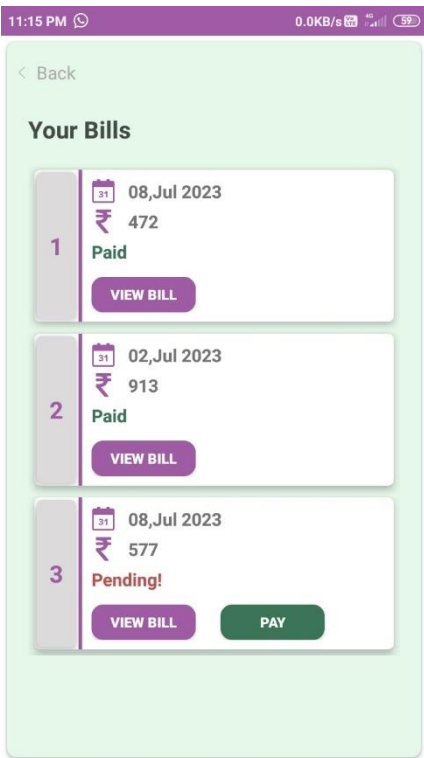
7. Available Services



8. Slots



9. Your Bills



10. Bill

11:16 PM

40.4KB/s

59

Bill No. : 524451

Invoice To

Swara

Date : 08,Jul 2023

Time : 05:25 PM

Rose Petals Beauty Salon

Supe-Pamer Road

Supe, Ahmednagar

Sr.No	Services/Products	PRICE	QTY	Total
1	Swedish Massage	450	-	450

UPI ID:8007878524@ybl

Scan to Pay

Subtotal

Rs.450.0

Tax

5%

Total

Rs.472

CONTACT

Pratibha Palaskar

rpbs@gmail.com

+91 8007878524

Terms & Conditions

Payment must be made in full at the time of service. Any cancellations or returns of products must be made within 24 hours.

11. Salon Report

11:26 PM

0.0KB/s

55

Dashboard

Weekly Report

SAT

SUN

MON

TUE

WED

THU

FRI

2.0

1.6

1.2

0.8

0.4

0.0

Weekly/Customer chart

Total income on a week: ₹6288

Total Customers

Total Services

12. Make Bill

11:27 PM

2.3KB/s

55

Make Bill

Choose purchase products

☒

Bajaj Hair oil

₹ 65

QTY :

-

1

+

Total : ₹ 65

Choose done services

☒

Conditioning

₹ 50

Generate Bill

13. Add Customer

11:27 PM

0.0KB/s

54

Make Bill

Choose purchase products

☒

Bajaj Hair oil

₹ 65

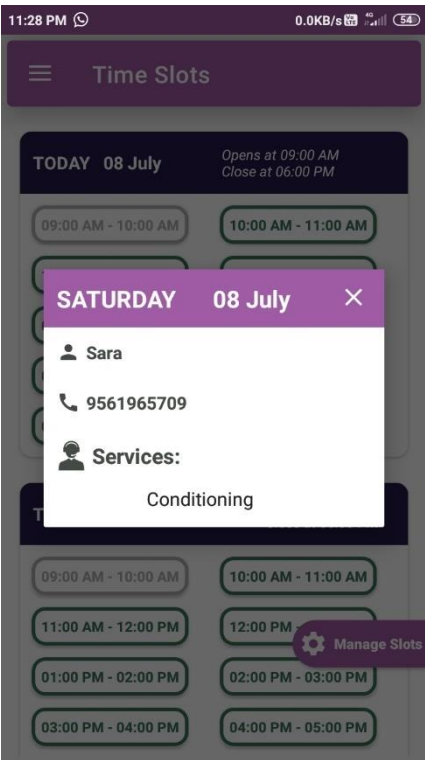
Customers Details

Priya

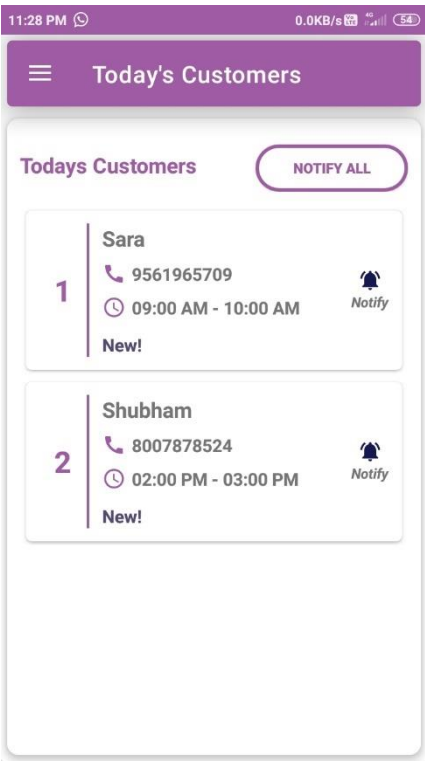
8390022530

SUBMIT

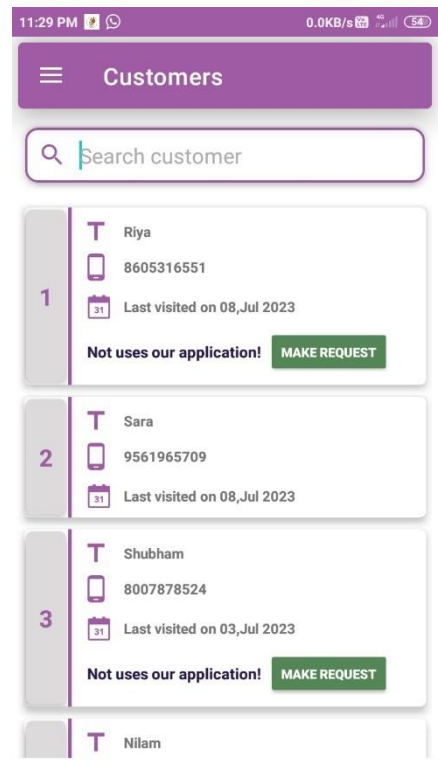
14. Check Slot Details



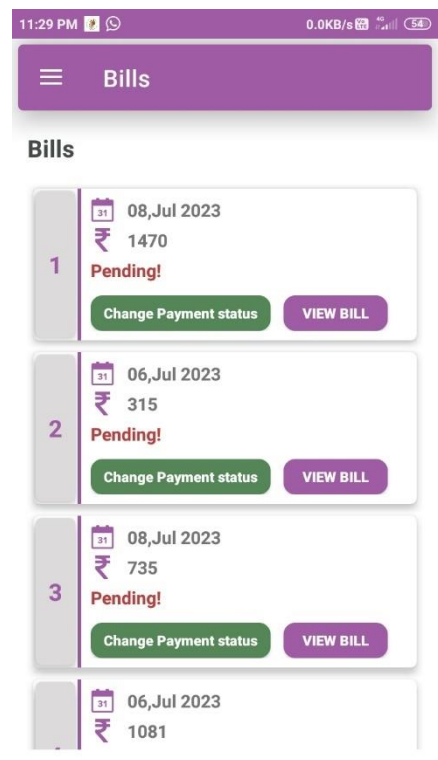
15. Today's customers



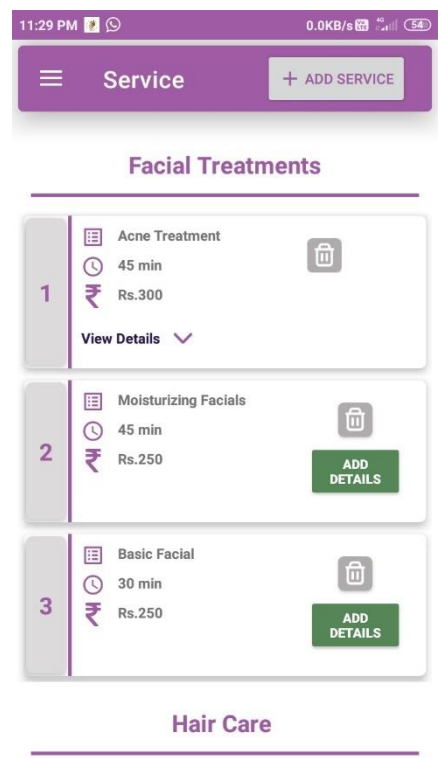
16. Customers



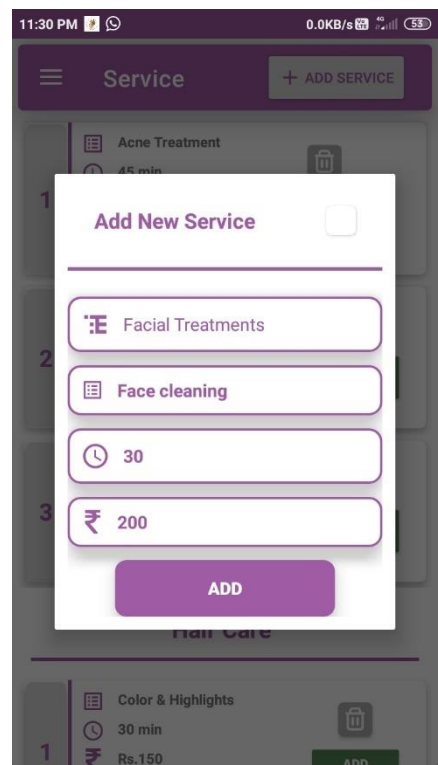
17. Bills



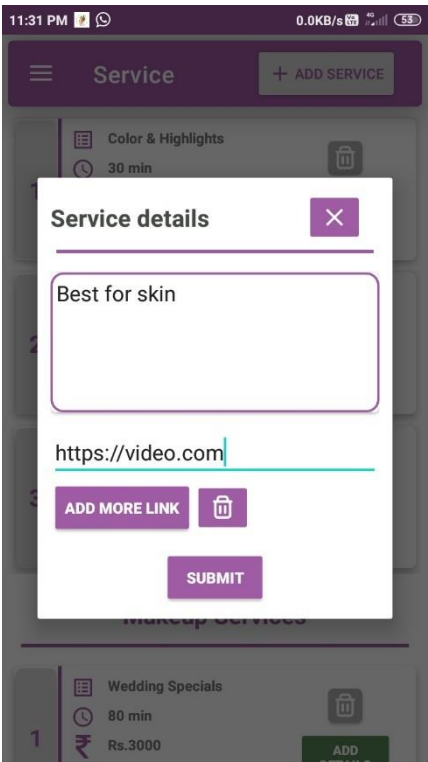
18. Services



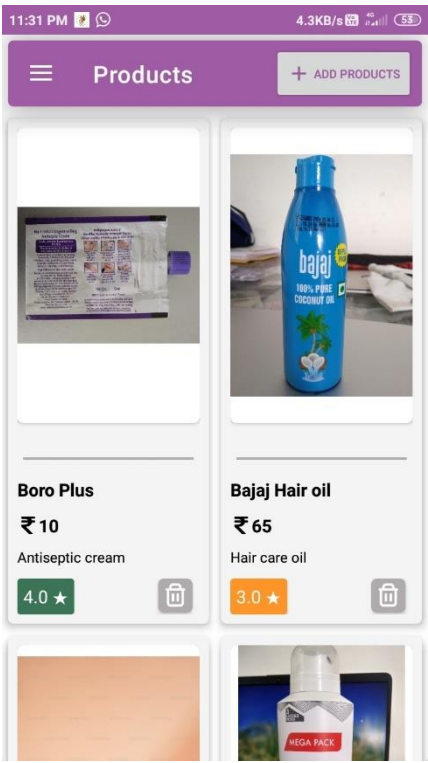
19. Add Service



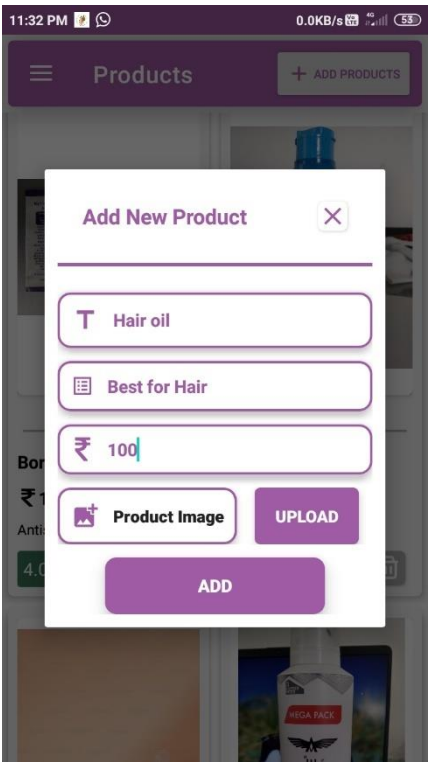
20. Add Service Details



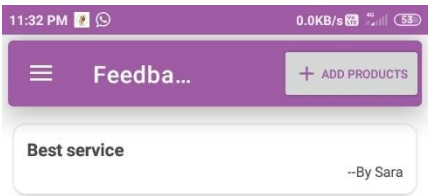
21. Products



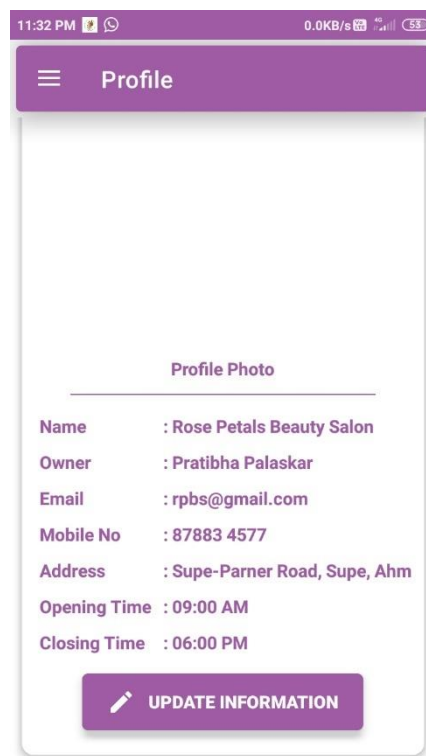
22. Add Product



23. Feedback



24. Salon Profile



A screenshot of a mobile application's 'Profile' screen. The screen has a purple header with a hamburger menu icon and the title 'Profile'. Below the header is a white card with a 'Profile Photo' placeholder. The card contains a list of details for 'Rose Petals Beauty Salon', including the owner's name, email, mobile number, address, opening and closing times. At the bottom of the card is a purple button with a pencil icon and the text 'UPDATE INFORMATION'.

11:32 PM 0.0KB/s

Profile

Profile Photo

Name : Rose Petals Beauty Salon

Owner : Pratibha Palaskar

Email : rpbs@gmail.com

Mobile No : 87883 4577

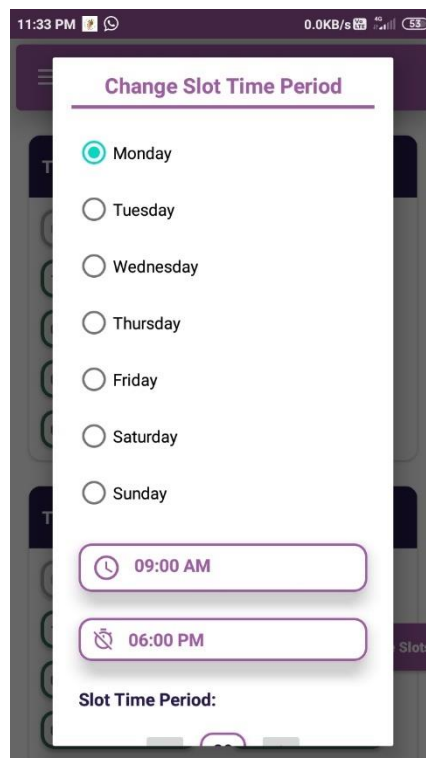
Address : Supe-Parner Road, Supe, Ahm

Opening Time : 09:00 AM

Closing Time : 06:00 PM

UPDATE INFORMATION

25. Change Slot time



A screenshot of a mobile application showing a 'Change Slot Time Period' dialog box. The dialog box is white with a purple header and contains a list of days of the week with radio buttons. Below the list are two input fields for the start and end times, each with a clock icon. At the bottom, it says 'Slot Time Period:'. The background shows a blurred view of the app's main interface.

11:33 PM 0.0KB/s

Change Slot Time Period

☒ Monday

☐ Tuesday

☐ Wednesday

☐ Thursday

☐ Friday

☐ Saturday

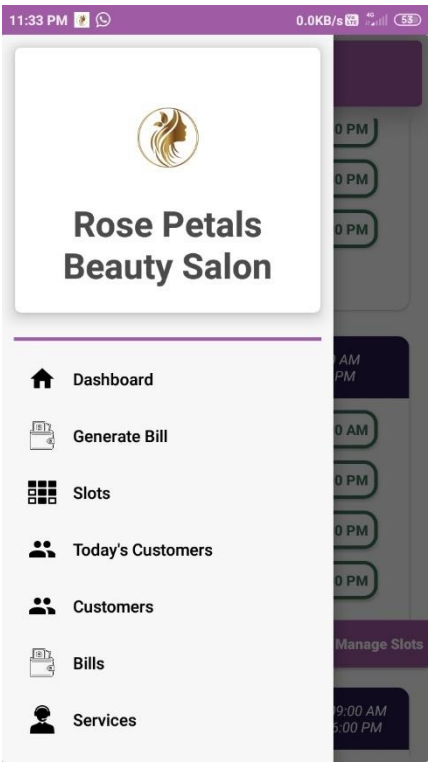
☐ Sunday

09:00 AM

06:00 PM

Slot Time Period:

26. Salon Menu



4. Coding

4.1 Code snippets

AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.subhdroid.rpbs">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/rpbs"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/rpbs_round"
        android:supportRtl="true"
        android:theme="@style/Theme.HairStylers"
        tools:targetApi="31">
```

```

<activity
    android:name="com.subhdroid.rpbs.Customer.CustomerDashboard"
    android:exported="false" />
<activity
    android:name="com.subhdroid.rpbs.Salon.SalonDashboard"
    android:exported="false" />
<activity
    android:name="com.subhdroid.rpbs.Login"
    android:exported="false" />
<activity
    android:name="com.subhdroid.rpbs.Registration"
    android:exported="false" />
<activity
    android:name="com.subhdroid.rpbs.SplashActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service
    android:name="com.subhdroid.rpbs.CloudMessage.FirebaseMessagingService"
    android:exported="true">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>

```

</service>

</application>

</manifest>

Login.java

```
package com.subhdroid.rpbs;
```

```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.AppCompatButton;
```

```
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.util.Log;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
```

```
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.messaging.FirebaseMessaging;
```

```

import com.subhdroid.rpbs.Customer.CustomerDashboard;
import com.subhdroid.rpbs.Customer.CustomerModel;
import com.subhdroid.rpbs.Salon.SalonDashboard;

import java.sql.Array;
import java.util.ArrayList;
import java.util.HashMap;

public class Login extends AppCompatActivity {
    TextView signUpTxt;
    AppCompatButton loginBtn;
    EditText username, password;
    public String customerToken = "";
    private static ArrayList<CustomerModel> customerList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        getToken();
        getAllCustomers();
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        signUpTxt = findViewById(R.id.signUpTxt);
        username = findViewById(R.id.username);
        password = findViewById(R.id.password);
        loginBtn = findViewById(R.id.loginBtn);

        signUpTxt.setOnClickListener(view -> {

```



```

Intent intent = new Intent(Login.this, Registration.class);

startActivity(intent);

});

AlertDialog.Builder alertDialog = new AlertDialog.Builder(Login.this);
alertDialog.setTitle("Error");
alertDialog.setMessage("Invalid credentials?\nPlease try again.");

alertDialog.setPositiveButton("Ok", (dialogInterface, i) -> dialogInterface.dismiss());

loginBtn.setOnClickListener(view -> {

    if (CheckAllFields()) {

        String key = checkValidCustomer(username.getText().toString(),
            password.getText().toString());

        if (username.getText().toString().contains("salon@gmail.com") &
password.getText().toString().contains("salon@2000")) {

            SharedPreferences pref = getSharedPreferences("Salon", MODE_PRIVATE);
            SharedPreferences.Editor editor = pref.edit();
            editor.putBoolean("salonLoggedIn", true);
            editor.apply();

            Intent intent = new Intent(Login.this, SalonDashboard.class);
            startActivity(intent);

        } else if (key != null) {

            SharedPreferences pref = getSharedPreferences("Customer",
MODE_PRIVATE);

            SharedPreferences.Editor editor = pref.edit();
            editor.putBoolean("CustomerLoggedIn", true);

```

```

        editor.putString("key", key);
        editor.apply();

        updateToken(username.getText().toString());
        Intent intent = new Intent(Login.this, CustomerDashboard.class);
        startActivity(intent);
    } else {
        alertDialog.show();
    }
}

});

}

@Override
public void onBackPressed() {
    AlertDialog.Builder alertDialog = new AlertDialog.Builder(Login.this);
    alertDialog.setTitle("Exit");
    alertDialog.setMessage("Do you want to exit app?");

    alertDialog.setPositiveButton("Yes", (dialogInterface, i) -> finishAffinity());

    alertDialog.setNegativeButton("No", (dialogInterface, i) -> dialogInterface.dismiss());

    alertDialog.show();
}

private boolean CheckAllFields() {

```

```

    if (username.length() == 0) {
        username.setError("Username is required");
        username.requestFocus();
        return false;
    }
    if (password.length() == 0) {
        password.setError("Password is required");
        password.requestFocus();
        return false;
    }

    return true;
}

private String checkValidCustomer(String username, String password) {
    if (customerList.size() != 0) {
        for (CustomerModel customer : customerList) {
            if
((username.equals(customer.getCustEmail())||username.equals(customer.getCustPhone()))
&& password.equals(customer.getCustPassword())) {
                return customer.getId();
            }
        }
    }

    return null;
}

```

```

public void getAllCustomers() {

    DatabaseReference customerRef =
    FirebaseDatabase.getInstance().getReference("customer");

    customerRef.addValueEventListener(new ValueEventListener() {

        @Override

        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            HashMap<String, Array> dataMap = (HashMap<String, Array>)
dataSnapshot.getValue();

            customerList = new ArrayList<>();

            if (dataMap != null) {

                for (String key : dataMap.keySet()) {

                    customerRef.child(key).addValueEventListener(new ValueEventListener() {

                        @Override

                        public void onDataChange(@NonNull DataSnapshot snapshot) {

                            CustomerModel customer = snapshot.getValue(CustomerModel.class);

                            customer.setId(key);

                            customerList.add(customer);

                        }

                    });

                }

            }

        }

    });

}
}

```

```

    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(Login.this, "Fail to get data.", Toast.LENGTH_SHORT).show();
    }
});

}

private String getToken() {
    FirebaseMessaging.getInstance().getToken()
        .addOnCompleteListener(task -> {
            if (!task.isSuccessful()) {
                Log.d("Log", "Fetching FCM registration token failed",
                    task.getException());
                return;
            }
            // Get new FCM registration token
            String token = task.getResult();
            customerToken = token;
        });
    return customerToken;
}

private void updateToken(String email) {
    DatabaseReference customerRef =
        FirebaseDatabase.getInstance().getReference("customer");

```

```

customerRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        HashMap<String, Array> dataMap = (HashMap<String, Array>)
dataSnapshot.getValue();
        for (String key : dataMap.keySet()) {

            customerRef.child(key).addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {

                    CustomerModel customer = snapshot.getValue(CustomerModel.class);

                    if (email.equals(customer.getCustEmail())) {
                        customerRef.child(key).child("fcmToken").setValue(customerToken);
                    }
                }
            });

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Log.d("DB Error : ", error.toString());
            }
        });
    }
});

@Override
public void onCancelled(@NonNull DatabaseError error) {

```

```

        Toast.makeText(Login.this, "Fail to get data.", Toast.LENGTH_SHORT).show();
    }
});

}

}

```

Registration.java

```

package com.subhdroid.rpbs;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.AppCompatButton;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.util.Log;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;

```

```

import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.messaging.FirebaseMessaging;
import com.subhdroid.rpbs.Customer.CustomerModel;

import java.sql.Array;
import java.util.ArrayList;
import java.util.HashMap;

public class Registration extends AppCompatActivity {
    boolean isAllFieldsChecked = false;
    public String customerToken = "";

    private static ArrayList<String> customerList;

    DatabaseReference customerRef =
        FirebaseDatabase.getInstance().getReference("customer");

    TextView signInTxt;
    EditText custName, custPhone, custEmail, custPassword, custCPassword, custGender;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_registration);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }
    }
}

```



```
}
```

```
// Set Salon type
```

```
String[] SalonThreeTypes = {"Men", "Women"};
```

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
    android.R.layout.simple_dropdown_item_1line, SalonThreeTypes);
```

```
AutoCompleteTextView textView = findViewById(R.id.custGender);
```

```
textView.setThreshold(3);
```

```
textView.setAdapter(adapter);
```

```
getAllCustomers();
```

```
getToken();
```

```
signInTxt = findViewById(R.id.signInTxt);
```

```
custName = findViewById(R.id.custName);
```

```
custPhone = findViewById(R.id.custPhone);
```

```
custEmail = findViewById(R.id.custEmail);
```

```
custPassword = findViewById(R.id.custPassword);
```

```
custCPassword = findViewById(R.id.custCPassword);
```

```
custGender = findViewById(R.id.custGender);
```

```
AppCompatActivity signUpBtn = findViewById(R.id.customerSignInBtn);
```

```
Dialog successDialog = new Dialog(Registration.this);
```

```
successDialog.setContentview(R.layout.registration_success_dialog);
```

```
successDialog.setCancelable(false);
```

```
AlertDialog.Builder alreadyExistsDialog = new AlertDialog.Builder(Registration.this);
```

```

alreadyExistsDialog.setTitle("Alert");
alreadyExistsDialog.setMessage("Account already exists?");
alreadyExistsDialog.setCancelable(false);
alreadyExistsDialog.setPositiveButton("Login", new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        finish();
        Intent intent = new Intent(Registration.this, Login.class);
        startActivity(intent);
    }
});

```

```

alreadyExistsDialog.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        dialogInterface.dismiss();
        custEmail.requestFocus();
    }
});

```

```

signUpBtn.setOnClickListener(view -> {

    isAllFieldsChecked = CheckAllFields();

    if (isAllFieldsChecked) {
        if (checkExistingUser(custEmail.getText().toString())) {
            alreadyExistsDialog.show();

```

```

        } else {
            insertRecord();
            successDialog.show();
        }
    }
});

```

```

AppCompatActivity okBtn = successDialog.findViewById(R.id.okBtn);
okBtn.setOnClickListener(view -> {
    successDialog.dismiss();
    Intent intent = new Intent(Registration.this, Login.class);
    startActivity(intent);
});

```

```

signInTxt.setOnClickListener(view -> {
    Intent intent = new Intent(Registration.this, Login.class);
    startActivity(intent);
});

```

```

}

```

```

@Override

```

```

public void onBackPressed() {
    super.onBackPressed();
    finish();
    Intent intent = new Intent(Registration.this, Login.class);
    startActivity(intent);
}

```

```

private void insertRecord() {

    CustomerModel customerModel = new CustomerModel(custName.getText().toString(),
    custPhone.getText().toString(),
        custEmail.getText().toString(), custCPassword.getText().toString(),
        custGender.getText().toString(), getToken());

    String customerID = customerRef.push().getKey();

    customerRef.child(customerID).setValue(customerModel);
}

```

```

private String getToken() {
    FirebaseMessaging.getInstance().getToken()
        .addOnCompleteListener(task -> {
            if (!task.isSuccessful()) {
                Log.d("Log", "Fetching FCM registration token failed",
                    task.getException());
                return;
            }

            // Get new FCM registration token
            String token = task.getResult();
            customerToken = token;

            Log.d("Log ", "token in fun : " + token);
            Toast.makeText(Registration.this, "Success", Toast.LENGTH_SHORT).show();

```

```
    });  
    return customerToken;  
}
```

```
private boolean CheckAllFields() {  
    if (custName.length() == 0) {  
        custName.setError("Name is required");  
        custName.requestFocus();  
        return false;  
    }
```

```
    if (custPhone.length() == 0) {  
        custPhone.setError("Phone is required");  
        custPhone.requestFocus();  
        return false;  
    }
```

```
    if (custEmail.length() == 0) {  
        custEmail.setError("Email is required");  
        custEmail.requestFocus();  
        return false;  
    }
```

```
    if (custPassword.length() == 0) {  
        custPassword.setError("Password is required");  
        custPassword.requestFocus();  
        return false;  
    } else if (custPassword.length() < 8) {  
        custPassword.setError("Password must be minimum 8 characters");  
    }
```

```

        custPassword.requestFocus();
        return false;
    }

    if
    (!((custCPassword.getText().toString()).equals((custCPassword.getText().toString())))) {
        custCPassword.setError("Password must be same");
        custCPassword.requestFocus();
        return false;
    }

    if (custGender.length() == 0 && (custGender.length() != 3 || custGender.length() != 5))
    {
        custGender.setError("Gender is required");
        custGender.requestFocus();
        return false;
    }
    return true;
}

private boolean checkExistingUser(String username) {
    for (String email : customerList) {
        if (username.equals(email)) {
            return true;
        }
    }
    return false;
}

```

```

public void getAllCustomers() {
    customerRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            customerList = new ArrayList<>();

            HashMap<String, Array> dataMap = (HashMap<String, Array>)
dataSnapshot.getValue();

            if (dataMap != null) {
                for (String key : dataMap.keySet()) {
                    customerRef.child(key).addValueEventListener(new ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull DataSnapshot snapshot) {

                            CustomerModel customer = snapshot.getValue(CustomerModel.class);
                            customerList.add(customer.getCustEmail());
                        }

                        @Override
                        public void onCancelled(@NonNull DatabaseError error) {
                            Log.d("DB Error : ", error.toString());
                        }
                    });
                }
            }
        }
    });
}

```

```

        @Override

        public void onCancelled(@NonNull DatabaseError error) {

            Toast.makeText(Registration.this, "Fail to get data.",
Toast.LENGTH_SHORT).show();

        }

    });

}

}

```

SalonDashboard.java

```

package com.subhdroid.rpbs.Salon;

import androidx.appcompat.app.ActionBarDrawerToggle;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.AppCompatButton;
import androidx.appcompat.widget.Toolbar;
import androidx.cardview.widget.CardView;
import androidx.core.view.GravityCompat;
import androidx.drawerlayout.widget.DrawerLayout;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.app.Dialog;
import android.app.ProgressDialog;
import android.app.TimePickerDialog;
import android.content.Intent;

```



```
import android.content.SharedPreferences;
import android.content.pm.ActivityInfo;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.airbnb.lottie.LottieAnimationView;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.navigation.NavigationView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.messaging.FirebaseMessaging;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.subhdroid.rpbs.Login;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.Feedbacks;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonBills;
```

```
import com.subhdroid.rpbs.Salon.SalonMenuFragments.TodayCustomers;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonCustomers;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonProducts;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonProfile;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.MakeBill;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonReport;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonServices;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonSlots;
import com.subhdroid.rpbs.R;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonSlotsModel;
import com.subhdroid.rpbs.Salon.SalonMenuFragments.SalonWeekModel;
```

```
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.TextStyle;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;
```

```
public class SalonDashboard extends AppCompatActivity {

    public static Toolbar toolbar;
    NavigationView navigationView;
    DrawerLayout drawerLayout;
```

```

LinearLayout warning;

DatabaseReference salonRef =
FirebaseDatabase.getInstance().getReference("salon").child(
    "information");

DatabaseReference slotsRef = FirebaseDatabase.getInstance().getReference("slots");

Dialog salonInformationDialog;

EditText salonName, ownerName, phone, email, address;

TextView openTime, closeTime, photoWarn;

private final int PICK_IMAGE_REQUEST = 22;

private ImageView imageView;

private String imgUrl = "";

boolean isAllFieldsChecked = false;

int uploadFlag = 0;

public String salonToken = "";

private Uri filePath;

AppCompatButton btnChoose, btnUpload, fillInfoBtn;

CardView imageCardView;

Boolean infoFlag = false;

LottieAnimationView loadingAnimation;

SimpleDateFormat timeFormat = new SimpleDateFormat("hh:mm a");

Calendar time = Calendar.getInstance();

ArrayList<SalonSlotsModel> slotList = new ArrayList<>();

@Override

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

```

```

setContentView(R.layout.activity_salon_dashboard);

getSalonInformation();

getToken();

setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

loadingAnimation = findViewById(R.id.loadingAnimation);

warning = findViewById(R.id.warning);

fillInfoBtn = findViewById(R.id.fillInfoBtn);


salonInformationDialog = new Dialog(SalonDashboard.this);
salonInformationDialog.setContentView(R.layout.salon_infromation);


fillInfoBtn.setOnClickListener(view -> salonInformationDialog.show());


salonName = salonInformationDialog.findViewById(R.id.salonName);
ownerName = salonInformationDialog.findViewById(R.id.ownerName);
phone = salonInformationDialog.findViewById(R.id.phone);
email = salonInformationDialog.findViewById(R.id.email);
address = salonInformationDialog.findViewById(R.id.address);
// Opening time and closing time
openTime = salonInformationDialog.findViewById(R.id.openTime);
closeTime = salonInformationDialog.findViewById(R.id.closeTime);
photoWarn = salonInformationDialog.findViewById(R.id.photoWarn);


openTime.setOnClickListener(v -> {

    final Calendar c = Calendar.getInstance();
    int hour = c.get(Calendar.HOUR_OF_DAY);
    int minute = c.get(Calendar.MINUTE);

```

```

TimePickerDialog timePickerDialog = new TimePickerDialog(SalonDashboard.this,
    (view, hourOfDay, minute1) -> {
        time.set(Calendar.HOUR_OF_DAY, hourOfDay);
        time.set(Calendar.MINUTE, minute1);
        time.set(Calendar.SECOND, 0);
        String formattedTime = timeFormat.format(time.getTime());
        openTime.setText(formattedTime);

        }, hour, minute, false);
timePickerDialog.show();
});

```

```

closeTime.setOnClickListener(v -> {

```

```

    final Calendar c = Calendar.getInstance();
    int hour = c.get(Calendar.HOUR_OF_DAY);
    int minute = c.get(Calendar.MINUTE);

```

```

TimePickerDialog timePickerDialog = new TimePickerDialog(SalonDashboard.this,
    (view, hourOfDay, minute1) -> {
        time.set(Calendar.HOUR_OF_DAY, hourOfDay);
        time.set(Calendar.MINUTE, minute1);
        time.set(Calendar.SECOND, 0);
        String formattedTime = timeFormat.format(time.getTime());
        closeTime.setText(formattedTime);

        }, hour, minute, false);
timePickerDialog.show();
});

```

```

btnChoose = salonInformationDialog.findViewById(R.id.btnChoose);
btnUpload = salonInformationDialog.findViewById(R.id.btnUpload);
imageView = salonInformationDialog.findViewById(R.id.imageView);
imageCardView = salonInformationDialog.findViewById(R.id.imageCardView);
btnChoose.setOnClickListener(view -> SelectImage());
btnUpload.setOnClickListener(view -> uploadImage());

```

```

Dialog successDialog = new Dialog(SalonDashboard.this);
successDialog.setContentview(R.layout.new_customer_added_succes_dialog);
successDialog.setCancelable(false);
salonInformationDialog.findViewById(R.id.submitBtn).setOnClickListener(view -> {

```

```

    isAllFieldsChecked = CheckAllFields();

```

```

    if (isAllFieldsChecked) {
        insertRecord();
        warning.setVisibility(View.GONE);
        successDialog.show();
    }

```

```

});

```

```

successDialog.findViewById(R.id.customerAddedOkBtn).setOnClickListener(new
View.OnClickListener() {

```

```

    @Override

```

```

    public void onClick(View view) {

```

```

        successDialog.dismiss();

```

```

        Intent intent = new Intent(SalonDashboard.this, SalonDashboard.class);

```

```

        startActivity(intent);
    }
}

```

```

    }
});

toolbar = findViewById(R.id.toolbar);
navigationView = findViewById(R.id.navigationView);
drawerLayout = findViewById(R.id.drawerLayout);
toolbar.setTitle("Dashboard");

ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this, drawerLayout,
toolbar,

    R.string.openDrawer, R.string.closeDrawer);

toggle.getDrawerArrowDrawable().setColor(getResources().getColor(R.color.white));
drawerLayout.addDrawerListener(toggle);
toggle.syncState();
drawerLayout.setVisibility(View.GONE);

if (infoFlag) {
    loadFragment(new SalonReport(), 0);
}

if (!infoFlag) {
    warning.setVisibility(View.VISIBLE);
}

navigationView.setNavigationItemSelectedListener(item -> {
    int id = item.getItemId();

    if (id == R.id.salonProfile) {
        loadFragment(new SalonProfile(), 1);
        toolbar.setTitle("Profile");
    } else if (id == R.id.salonCustomerItem) {
        loadFragment(new SalonCustomers(), 1);
    }
});

```

```

        toolbar.setTitle("Customers");
    } else if (id == R.id.salonDashboardItem) {
        loadFragment(new SalonReport(), 1);
        toolbar.setTitle("Dashboard");
    } else if (id == R.id.salonSlotsItem) {
        loadFragment(new SalonSlots(), 1);
        toolbar.setTitle("Time Slots");
    } else if (id == R.id.salonBill) {
        loadFragment(new MakeBill(), 1);
        toolbar.setTitle("Make Bill");
    } else if (id == R.id.salonServiceItem) {
        loadFragment(new SalonServices(), 1);
        toolbar.setTitle("Services");
    } else if (id == R.id.salonTodayCustmer) {
        loadFragment(new TodayCustomers(), 1);
        toolbar.setTitle("Today's Customers");
    } else if (id == R.id.salonProduct) {
        loadFragment(new SalonProducts(), 1);
        toolbar.setTitle("Products");
    } else if (id == R.id.salonTotalBills) {
        loadFragment(new SalonBills(), 1);
        toolbar.setTitle("Bills");
    } else if (id == R.id.salonFeedback) {
        loadFragment(new Feedbacks(), 1);
        toolbar.setTitle("Feedbacks");
    } else if (id == R.id.salonLogout) {
        logOut();
    }
    drawerLayout.closeDrawer(GravityCompat.START);

```



```

        return true;
    });
}

```

@Override

```

public void onBackPressed() {
    if (drawerLayout.isDrawerOpen(GravityCompat.START))
        drawerLayout.closeDrawer(GravityCompat.START);
    else {
        AlertDialog.Builder alertDialog = new AlertDialog.Builder(SalonDashboard.this);
        alertDialog.setTitle("Exit");
        alertDialog.setMessage("Do you want to exit app?");

        alertDialog.setPositiveButton("Yes", (dialogInterface, i) -> finishAffinity());

        alertDialog.setNegativeButton("No", (dialogInterface, i) -> dialogInterface.dismiss());
        alertDialog.show();
    }
}

```

```

public void loadFragment(Fragment fragment, int flag) {
    FragmentManager fm = getSupportFragmentManager();
    FragmentTransaction ft = fm.beginTransaction();
    if (flag == 0) {
        ft.add(R.id.SalonFragmentContainer, fragment);
    } else {
        ft.replace(R.id.SalonFragmentContainer, fragment);
    }
}

```

```

        ft.commit();
    }

    private void logOut() {
        AlertDialog.Builder alertDialog = new AlertDialog.Builder(SalonDashboard.this);
        alertDialog.setTitle("Logout");
        alertDialog.setMessage("Do you want to logout?");

        alertDialog.setPositiveButton("Yes", (dialogInterface, i) -> {
            SharedPreferences pref = getSharedPreferences("Salon", MODE_PRIVATE);
            SharedPreferences.Editor editor = pref.edit();
            editor.putBoolean("salonLoggedIn", false);
            editor.apply();

            Intent intent = new Intent(SalonDashboard.this, Login.class);
            startActivity(intent);
        });

        alertDialog.setNegativeButton("No", (dialogInterface, i) -> dialogInterface.dismiss());

        alertDialog.show();
    }

    public void getSalonInformation() {

        salonRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                SalonModel salonInformation = dataSnapshot.getValue(SalonModel.class);
            }
        });
    }

```

```

        loadingAnimation.setVisibility(View.GONE);
        if (salonInformation == null) {
            salonInformationDialog.show();
        } else {
            infoFlag = true;
            warning.setVisibility(View.GONE);
            drawerLayout.setVisibility(View.VISIBLE);
            loadFragment(new SalonReport(), 0);
        }
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }

    });
}

private void SelectImage() {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent, "Select Image from here..."),
PICK_IMAGE_REQUEST);
}

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {

        super.onActivityResult(requestCode, resultCode, data);

```

```

        if (requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK &&
            data != null && data.getData() != null) {

            filePath = data.getData();

            try {

                Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(),
filePath);

                imageView.setImageBitmap(bitmap);

                btnChoose.setText("Choose another image");

                imageCardView.setVisibility(View.VISIBLE);

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

    }
}

```

```

private void uploadImage() {

    if (filePath != null) {

        ProgressDialog progressDialog = new ProgressDialog(this);

        progressDialog.setTitle("Uploading...");

        progressDialog.show();

        // Defining the child of storageReference

        String timeStamp = new SimpleDateFormat("MMHHddmmssyyyy").format(new
java.util.Date());

        StorageReference ref =
FirebaseStorage.getInstance().getReference("SalonImages").child(timeStamp);

        ref.putFile(filePath).addOnSuccessListener(taskSnapshot -> {

```

```

        progressDialog.dismiss();

        StorageReference storageReference =
        FirebaseStorage.getInstance().getReference("SalonImages").child(

            timeStamp);

        Task<Uri> urlTask =
        storageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>()
        {

            @Override

            public void onSuccess(Uri uri) {

                imgUrl = uri.toString();

                Toast.makeText(SalonDashboard.this, "Image Uploaded",

                    Toast.LENGTH_SHORT).show();

            }

        });

        }).addOnFailureListener(e -> {

            progressDialog.dismiss();

            Toast.makeText(SalonDashboard.this, "Failed " + e.getMessage(),

                Toast.LENGTH_SHORT).show();

            }).addOnProgressListener(taskSnapshot -> {

                double progress = (100.0 * taskSnapshot.getBytesTransferred() /

                taskSnapshot.getTotalByteCount());

                progressDialog.setMessage("Uploaded " + (int) progress + "%");

            });

        uploadFlag = 1;

        photoWarn.setVisibility(View.INVISIBLE);

    }

}

```

```

private void insertRecord() {
    SalonModel SalonModel = new SalonModel(salonName.getText().toString(),
        ownerName.getText().toString(), phone.getText().toString(),
email.getText().toString(), openTime.getText().toString(), closeTime.getText().toString(),
        address.getText().toString(), imgUrl, salonToken);

    salonRef.setValue(SalonModel);

    infoFlag = true;

    for (int i = 0; i < 7; i++) {
        LocalDate date = LocalDate.now().plusDays(i);
        DayOfWeek futureDayOfWeek = date.getDayOfWeek();
        String dayName = futureDayOfWeek.getDisplayName(TextStyle.FULL,
Locale.getDefault());
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd MMMM");
        String formattedDate = date.format(formatter);

        addNewSlot(createSlot(), dayName, formattedDate);
        if (i == 6) {
            drawerLayout.setVisibility(View.VISIBLE);
            loadFragment(new SalonReport(), 0);
        }
    }

}

private boolean CheckAllFields() {
    if (salonName.length() == 0) {

```

```

        salonName.setError("Salon name is required");
        salonName.requestFocus();

        return false;
    }

    if (ownerName.length() == 0) {
        ownerName.setError("Pincode is required");
        ownerName.requestFocus();

        return false;
    }

    if (phone.length() == 0) {
        phone.setError("Phone is required");
        phone.requestFocus();

        return false;
    }

    if (email.length() == 0) {
        email.setError("Email is required");
        email.requestFocus();

        return false;
    }

    if (openTime.length() == 0) {
        openTime.setError("Open time is required");
        openTime.requestFocus();

        return false;
    }

    if (closeTime.length() == 0) {
        closeTime.setError("Close time is required");
    }

```

```

        closeTime.requestFocus();

        return false;
    }

    if (address.length() == 0) {
        address.setError("Address is required");
        address.requestFocus();
        return false;
    }

    if (uploadFlag == 0) {
        btnChoose.setError("Required");
        photoWarn.setVisibility(View.VISIBLE);
        btnChoose.requestFocus();
        return false;
    }

    return true;
}

private String getToken() {
    FirebaseMessaging.getInstance().getToken()
        .addOnCompleteListener(task -> {
            if (!task.isSuccessful()) {
                Log.d("Log", "Fetching FCM registration token failed",
                    task.getException());
                return;
            }

            String token = task.getResult();
            salonToken = token;
        });
}

```



```

        return salonToken;
    }

```

```

public static void removeToolbarChild() {
    int lastIndex = toolbar.getChildCount() - 1;
    if (lastIndex > 1) {
        View lastView = toolbar.getChildAt(lastIndex);
        toolbar.removeView(lastView);
    }
}

```

```

private void addNewSlot(ArrayList<SalonSlotsModel> slotList, String dayName, String
date) {
    SalonWeekModel weekModel = new SalonWeekModel(openTime.getText().toString(),
        closeTime.getText().toString(), "60", "Open",
        date,
        slotList);
    slotsRef.child(dayName.toUpperCase()).setValue(weekModel);
}

```

```

private ArrayList<SalonSlotsModel> createSlot() {
    ArrayList<SalonSlotsModel> slotList = new ArrayList<>();
    String openTimeString = openTime.getText().toString();
    String closeTimeString = closeTime.getText().toString();
    final String TIME_FORMAT = "hh:mm a";
    SimpleDateFormat timeFormat = new SimpleDateFormat(TIME_FORMAT);
}

```

```

try {
    int openHours;
    int openMinutes;

    int closeHours;
    int closeMinutes;

    Date date1 = timeFormat.parse(openTimeString);
    Date date2 = timeFormat.parse(closeTimeString);

    Calendar calendar1 = Calendar.getInstance();
    calendar1.setTime(date1);
    Calendar calendar2 = Calendar.getInstance();
    calendar2.setTime(date2);

    openHours = calendar1.get(Calendar.HOUR_OF_DAY);
    openMinutes = calendar1.get(Calendar.MINUTE);
    closeHours = calendar2.get(Calendar.HOUR_OF_DAY);
    closeMinutes = calendar2.get(Calendar.MINUTE);

    final int TIME_INTERVAL_MINUTES = 60;

    Calendar openingTime = Calendar.getInstance();
    openingTime.set(Calendar.HOUR_OF_DAY, openHours);
    openingTime.set(Calendar.MINUTE, openMinutes);
    openingTime.set(Calendar.SECOND, 0);

    Calendar closingTime = Calendar.getInstance();
    closingTime.set(Calendar.HOUR_OF_DAY, closeHours);
    closingTime.set(Calendar.MINUTE, closeMinutes);

```

```

closingTime.set(Calendar.SECOND, 0);

Calendar currentTime = openingTime;
SimpleDateFormat timeFormatter = new SimpleDateFormat(TIME_FORMAT);

String endTime = timeFormatter.format(closingTime.getTime());
while (currentTime.before(closingTime)) {
    String timeSlot = timeFormatter.format(currentTime.getTime());
    currentTime.add(Calendar.MINUTE, TIME_INTERVAL_MINUTES);
    String timeSlot2 = timeFormatter.format(currentTime.getTime());

    Date d1 = timeFormatter.parse(timeSlot2);
    Date d2 = timeFormatter.parse(endTime);
    String slotTime = timeSlot + " - " + timeSlot2;
    if (d1.compareTo(d2) <= 0) {
        ArrayList<String> emptyService = new ArrayList<>();
        emptyService.add("Other");

        SalonSlotsModel slotModel = new
SalonSlotsModel(slotTime,emptyService,"Not booked yet");

        slotList.add(slotModel);
    }
}
} catch (ParseException e) {
    e.printStackTrace();
}
return slotList;
}
}

```

5. Testing

5.1 Strategies used for Testing

1. Unit Testing

Unit testing concentrates verification on the smallest element of the program - the module. Using the detailed design description important control paths are tested to establish errors within the bounds of the module.

2. Integration testing

Once all the individual units have been tested there is a need to test how they were put together to ensure no data is lost across interface, one module does not have an adverse impact on another and a function is not performed correctly.

5.2 System testing for the current system:

In this level of testing, we are testing the system as a whole after integrating all the main modules of the project. We are testing whether system is giving correct output or not. All the modules were integrated and the flow of information among different modules was checked. It was also checked that whether the flow of data is as per the requirements or not. It was also checked that whether any particular module is non-functioning or not i.e., once the integration is over each and every module is functioning in its entirety or not. In this level of testing, we tested the following: - Whether all the forms are properly working or not. Whether all the forms are properly linked or not. Whether all the images are properly displayed or not. Whether data retrieval is proper or not.

5.3 Test Cases

Testing is the exposure of system to trial input to see whether it produces correct output. Testing assumes requirements that are already validated. Testing cannot guarantee correctness, no method can guarantee correctness. Testing is the process of detecting presence of faults. Debugging is the process of locating and correcting faults. Once source code has been generated, software must be tested to uncover as many errors as possible before delivery to your customer. Our goal is to design a series of test cases that have a high

likelihood of finding errors. That's where Software Testing techniques enter the picture. A set of test cases designed to exercise both internal logic and external requirements is designed and documented, expected results are defined and actual results are recorded.

5.4 Test Log

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Login with valid Data	<ol style="list-style-type: none"> 1. Go to app 2. Enter UserId 3. Enter Password 4. Click Submit 	Userid = salon Password = pass99	User should Login into an application	As expected,	Pass
TU02	Check Login with invalid Data	<ol style="list-style-type: none"> 1. Go to app 2. Enter UserId 3. Enter Password 4. Click Submit 	Userid = salon Password = salons99	User should not Login into an application	As expected,	Pass
TU03	Add new Customer	<ol style="list-style-type: none"> 1. Go to app 2. Enter customer details 3. Click on add button 	Name= Shubham Mobile=' '	Customer shouldn't be added	As expected,	Pass
TU04	SMS	<ol style="list-style-type: none"> 1. Go to app 2. Enter mobile number 3. Click on send button 	Message= Hello users Mobile Number= ' '	SMS will not go	As expected,	Pass
TU05	QR Generator	<ol style="list-style-type: none"> 1.Go to app 2.Select services or products. 3.Click on generate button 	Services=facial, hair cut Products=Shampoo, Hair oil	QR will generate	As expected,	Pass

Step 1) A simple test case to explain the scenario would be

Test Case #	Test Case Description
1	Check response when valid email and password is entered

Step 2) Test the Data.

In order to execute the test case, you would need Test Data. Adding it below

Test Case #	Test Case Description	Test Data
1	Check response when valid email and password is entered	Email: salon@email.com Password: lNf9^Oti7^2h

Identifying test data can be time-consuming and may sometimes require creating test data afresh. The reason it needs to be documented.

Step 3) Perform actions.

In order to execute a test case, a tester needs to perform a specific set of actions on the AUT. This is documented as below:

Test Case #	Test Case Description	Test Steps	Test Data
1	Check response when valid email and password is entered	1) Enter Email Address 2) Enter Password 3) Click Sign in	Email: salon@email.com Password: lNf9^Oti7^2h

Many times, the Test Steps are not simple as above, hence they need documentation. Also, the author of the test case may leave the organization or go on a vacation or is sick and off duty or is very busy with other critical tasks. A recently hire may be asked to execute the test case. Documented steps will help him and also facilitate reviews by other stakeholders.

Step 4) Check behaviour of the AUT.

The goal of test cases in software testing is to check behaviour of the AUT for an expected result. This needs to be documented as below

Test Case #	Test Case Description	Test Data	Expected Result
1	Check response when valid email and password is entered	Email: salon@email.com Password: lNf9^Oti7^2h	Login should be successful

During test execution time, the tester will check expected results against actual results and assign a pass or fail status

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Check response when valid email and password is entered	Email: salon@email.com Password: INf9^Oti7^2h	Login should be successful	Login was successful	Pass

Step 5) That apart your test case -may have a field like,
Pre – Condition which specifies things that must be in place before the test can run. For our test case, a pre-condition would be to have a browser installed to have access to the site under test. A test case may also include Post – Conditions which specifies anything that applies after the test case completes. For our test case, a postcondition would be time & date of login is stored in the database.

6. Limitations of Proposed System

- Cannot work in IOS devices.
- Cannot run without internet.
- UI only in light mode.

7. Proposed Enhancements

- I have provided best service for app users.
- Customers can book their time slots for haircut.
- Salon owner can store their customers data and remind for their hair cut in particular time.
- If any customer forgot to cut hair in his time period, then system will automatically remind it day by day
-

11. Conclusion

To conclude the description about project. The project developed using Android, Java, and Firebase is based on requirement specification of the user and the analysis of the existing user, with flexibility for future enhancement. The following conclusions can be deduced from the developed of the project.

- Automation of the entire system improves the efficiency.
- It provides a friendly graphical user interface which proves to be better when compared to the existing system.
- It gives appropriate access to the authorized users depending on their permissions.
- Updating of information is becomes so easier.
- The system has adequate scope for modification in future if it is necessary.
- This system has very softly to handle and execute.
- Work done manually is much slower compared with this system, which is must faster.
- Increase performance
- The system has less time consuming
- More efficient
- Highly flexible

9. Bibliography

- **Reference Books:**

ANDROID APP DEVELOPMENT

-By J. Paul Cardle

STARTING WITH ANDROID

-By M.M. Sharma

ANDROID PROGRAMMING GUIDE

-By Bharti N. Raut

- **Websites:**

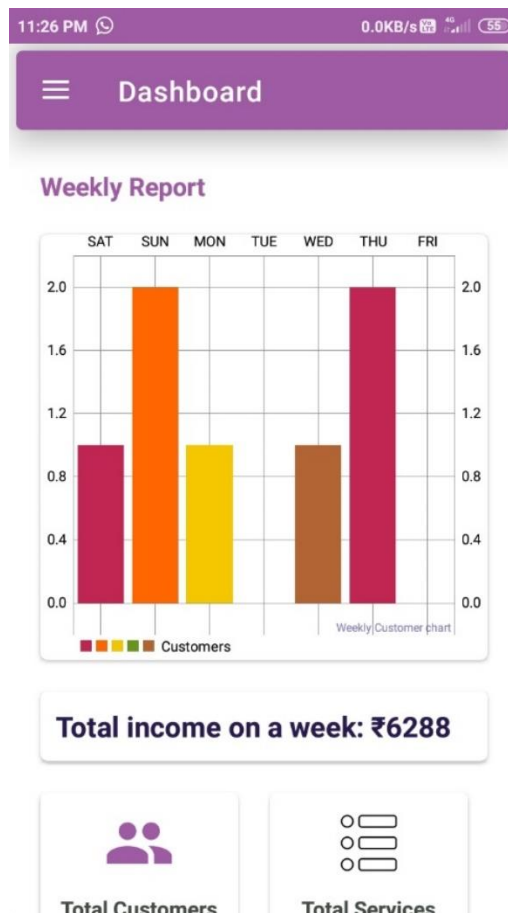
<https://console.firebase.google.com/>

<https://developer.android.com/studio>

<https://developer.android.com/guide/components/activities/intro-activities>

<https://developer.android.com/guide/fragments>

12.User Manual



This is Salon dashboard screen. This screen shows all reports of Salons. Like Salon weekly customers, income, etc.

This is also showing Salon customers, services and products.

11:27 PM 2.3KB/s 4G 55

Make Bill

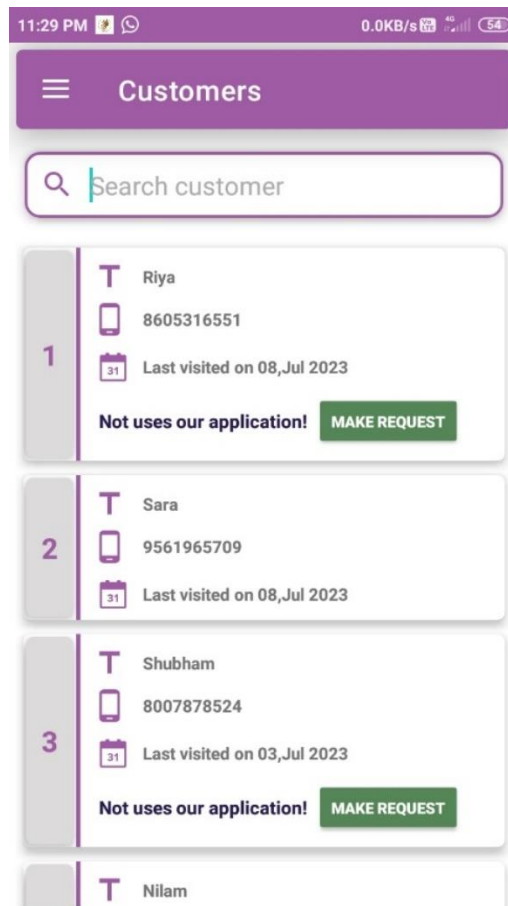
Choose purchase products

<input checked="" type="checkbox"/>	Bajaj Hair oil	₹ 65
QTY : <input type="button" value="-"/> 1 <input type="button" value="+"/> Total : ₹ 65		

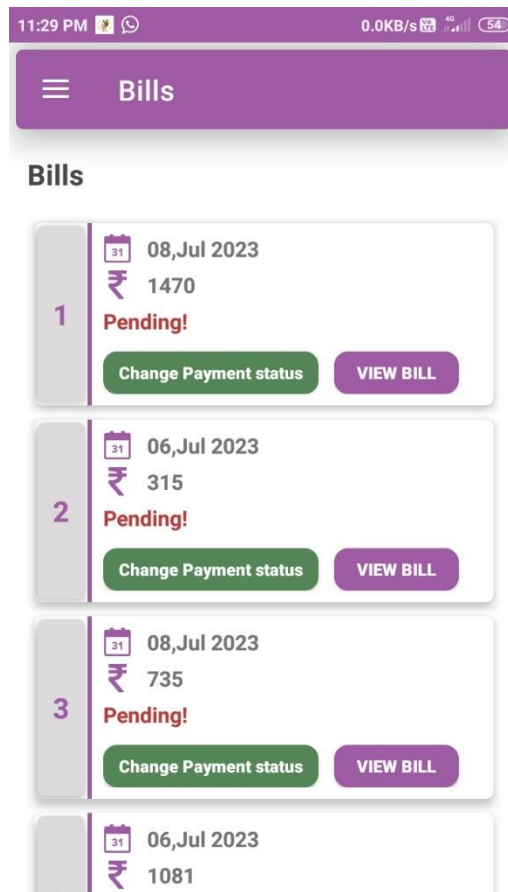
Choose done services

<input checked="" type="checkbox"/>	Conditioning	₹ 50
-------------------------------------	--------------	------

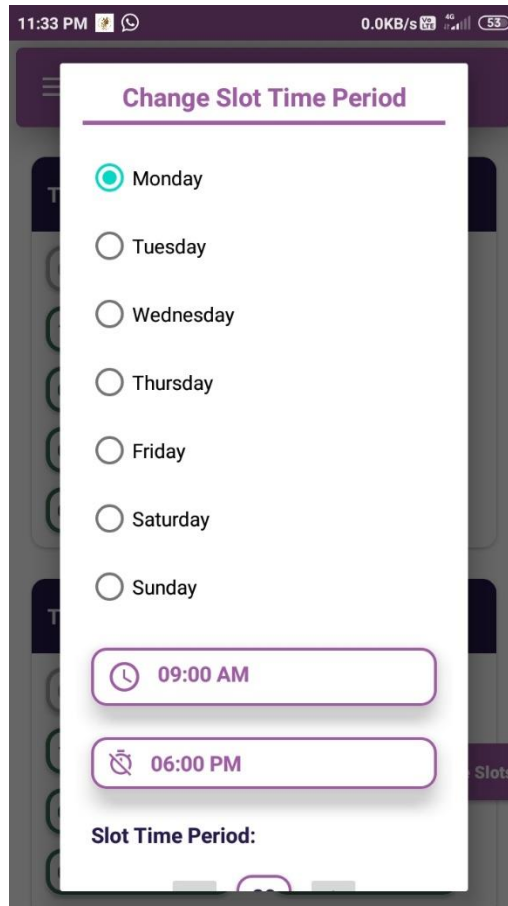
This page generates the bill. There is payment QR present on bill and system will automatically generate the this QR code.



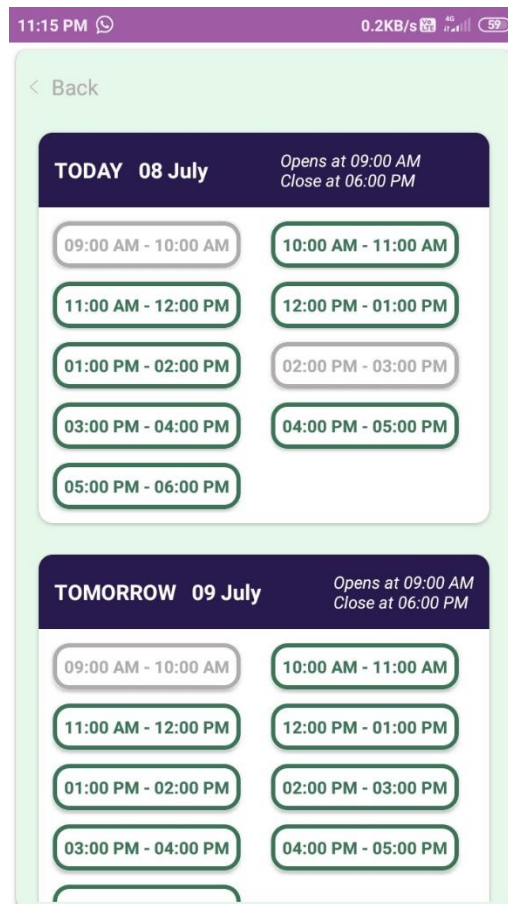
This page shows the available customers of salon. Also, it can show which users are used salon app and not, Owner can make request to customers which are not uses salon Application.



This page shows all the bills of customers. There is a pay option here. Also, if customers have paid bill the bill status is automatically change.



This page used to change slot timing or salon open time and close time.



This page shows all the available slot and booked slots. Available slots are mark by green border or green colour. And booked slots are marked by Gray colour.