

Salon Appointment App

Higher National Diploma in information system management

Final Project Report

21.1P

W.M. Pawani Maheshika Bandara

KAHNDISM21.1P-001

U.D.Jayarathna

KAHNDISM21.1P-007



School of Computing

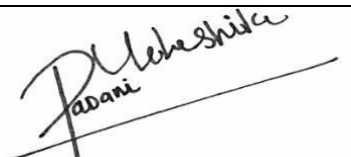
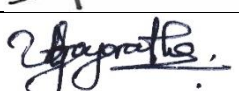
National Institute of Business Management

Kandy

Declaration

“I’m certify that this project does not incorporate without acknowledgement, any materials previously submitted for a Higher National Diploma in any institution and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my project report, if accepted, to be made available for Photocopying and interlibrary loans, and for the title and summary to be made available to outside organizations.”

Project: Salon Appointment App

Name	Index Number	Signature
W.M. Pawani Maheshika Bandara	KAHNDISM21.1P-001	
U.D.Jayarathna	KAHNDISM21.1P-007	

Supervisor : Mrs.Inoka Abhayasinghe

Date :

Signature :

Acknowledgement

First of all I am grateful to Mrs.Inoka Perera the client of this project and without whom this project wouldn't have been a success. The support she had given during the project and the time spent in requirement gathering were the main factors for the success of this project.

I also express my gratitude to members of the staff who gave me immense support and guidance towards the success of this project. I must also mention my deep sense of appreciation to Mrs.Inoka Abhayasinghe, the project supervisor who had provided guidance from the very beginning of this project.

Finally, I would like to thank all the colleagues who have been with me in all difficult times with suggestions, encouragement, and unconditional support they have extended to me in numerous ways.

Abstract

A salon appointment app with the feature of tailor shop management is a software application that provides a centralized platform for businesses to manage both their salon and tailor shop operations. The app enables customers to schedule appointments for both types of services and provides real-time updates on availability, wait times, and order statuses.

For salon and tailor shop owners and managers, the app offers features such as appointment scheduling, inventory management, order tracking, and production management. The app can also provide valuable data insights into business performance, allowing owners and managers to identify areas for improvement and inform business strategy.

By combining salon appointment scheduling with tailor shop management features, this type of app can help businesses improve efficiency, reduce administrative work, and enhance the overall customer experience.

Table of Contents

1.0 Introduction.....	5
1.1 Client Information.....	5
1.2 Scope of project	6
1.3 Aims and Objectives.....	6
2.0 Analysis.....	7
2.1 Requirements Gathering Methods	7
2.2 Failures in Existing system	7
2.2Requirements	8
2.2.1 Functional Requirements	8
2.2.2 Non-functional Requirements.....	9
2.3 Feasibility study.....	9
2.3.1 Technical feasibility	9
2.3.2 Social feasibility	9
2.3.3 Economic feasibility	9
3.0 System Design.....	10
3.1 System Methodology	10
3.2 System Deliverables.....	10
3.2.1 Entity Relationship Diagram	10
3.2.2 Class Diagram	12
3.2.3 Use Case	14
3.2.4 Activity Diagram	15
3.2.5 Sequence Diagram.....	17
3.3 User Interfaces.....	18
3.3.1 Registration Interface	18
3.3.2 Login Interface	20
3.3.3 Scheduling Interface	23
3.3.4 Service selection Interface	28
3.3.5 Measurement Interface.....	32
3.3.6 App icon	35
4.0 Implementation.....	35
4.1 Software Environment	35
4.2 Hardware Environment.....	35
4.3 Technologies to be used	36
4.4 Development Tools.....	36

4.5 Code Features	36
5.0 Evaluation and Testing	43
5.1 Techniques of Software Testing	43
5.1.1 Black Box Testing.....	43
5.1.2 White Box Testing.....	44
5.2 Types of Testing	44
5.2.1 Unit Testing	44
5.2.2 Integration Testing.....	44
5.2.3 System Testing.....	44
5.2.4 Acceptance Testing	44
5.3 Test Plan and Test Case.....	44
5.4 User Evaluation	50
5.5 Use Evaluation Summary	51
6.0 Conclusion.....	51
7.0 Future Enhancements.....	52
8.0 References.....	52
9.0 Appendix.....	52

1.0 Introduction

A salon appointment app with the feature of tailor shop management is a software application designed to help businesses manage both their salon appointment scheduling and tailor shop operations.

This type of app can be particularly useful for businesses that offer both salon and tailor services, as it provides a centralized platform to manage both types of operations. The app can offer features such as appointment scheduling, inventory management, order tracking, and production management.

Customers can use the app to schedule appointments for salon services such as haircuts, styling, and coloring, as well as tailor services such as measurements, fittings, and alterations. The app can provide real-time updates on appointment availability, wait times, and order statuses, allowing customers to stay informed and up to date.

For salon and tailor shop owners and managers, the app can provide valuable insights into business performance. By analyzing data such as appointment frequency, inventory levels, and production throughput, the app can help identify areas for improvement and inform business strategy.

Overall, a salon appointment app with the feature of tailor shop management can help businesses streamline their operations, improve the customer experience, and enhance business performance.

1.1 Client Information

Client Name – Isansu Fashion Designers

Salon Isansu is a small-scale salon and dress making business which was started in 1989 and was officially registered as a business in 1993. The guarantee of quality services for the patron's money worth is the vision of the business. She started her journey as a small bridal salon and expanded it with several staff members. It leads to create them to a huge customer base. They are using a manual system for salon management and tailor shop management. They maintain manual documentation for their each and every business activity like making appointments, cancelling appointments, managing customer payments, tracking customers, managing complaints, managing personal documentation of staff etc.

Services offered by the Client.

Bridal – Hair styles, Makeups, Dressing, Saree Designing and all the services of bride's maids and flower girls including flower bouquets.

Hair – Haircuts, Hair coloring, Head Massage, Oil Treatments

Beauty – Facials, Eyebrow, Threading, Pedicure, Manicure

1.2 Scope of project

This salon management system will be a mobile application specification made for tailors and beauticians. This application helps to manage all the work of tailor shops and salons. There are four users who use the application. They are admin, tailor, beautician and customer, admin (receptionist) has access to all functionalities of the application admin can add the customer, search customer, can set booking and billing. Tailor has access to login, can manage measurement book and handing over the finished garments. Beauticians have access to make appointments and can view the appointments. Customers have limited access to the functionalities of the application. They can login to their accounts and can update and delete their information. Other than that, the customer can make appointments.

1.3 Aims and Objectives

Mobilization -All the details of customer whether it is small or big, will be mobilization.

No redundant data: as this management system will be centralized, the chances of data in the system are close to nil.

Automation: The automation feature of this management system will mitigate the take of writing the paper.

E.g.: -There is no need to waste papers to write measurements of customers.

Time saving: In today's rush hour of the life, it is difficult for a customer to go to tailor shop or salon to make an appointment or give measurements for cloths. By this management system, it will be easier for a customer to make appointments and also tailor or beautician can easily manage their day today work. As a matter of fact, it will be easier for each individual person who is associated with the system to be in touch as per needed.

2.0 Analysis

2.1 Requirements Gathering Methods

The most important step in the analysis phase is gathering the client's needs utilizing fact-finding methodologies. To avoid conflicts with client expectations, these strategies should be handled properly. The requirements were gathered using a variety of methods in order to clearly define the project scope.

- **Interviews**

The most popular method for acquiring requirements is through interviews. Using face-to-face interactions, information is gathered from people using this fact-finding technique. These interviews may be conducted in a systematic or unstructured manner. Clarifying the facts and removing any ambiguity was a great help. It increases the project's advantage.

- **Observations**

Most of the target audience expressed their needs in their own words, and there were also conflicting needs. In those circumstances, this technique can be quite dependable. The actions that customers neglected or misdescribed were noticed using the manual system that is currently in place.

- **Sample Documentation**

Through the examination of pertinent records and reports, this strategy would assist in gaining a thorough understanding of the system. Sample paperwork, like a diary or account book, is utilized to clarify the needs that have been obtained.

2.2 Failures in Existing system

Before the project was implemented, "Salon Isansu" stakeholders employed a manual process, which caused several issues with their day-to-day work (at the salon and the tailor shop). The business owner didn't use any software to organize her workload. As a result, every time the following situation arose, all the parties encountered significant challenges.

The salon owner and cashier had the following issues:

- It was challenging to recognize regular clients.
- Difficult to manage her own employees' vacations.
- Had to consistently record appointments, as well as their dates, times, services, and payments.
- She was required to keep daily reminders on her phone regarding client appointment information.
- Had to set service prices and periodically update them in papers.
- Had trouble locating costs that were appropriate for each service.
- Issue of manual bills. The entire service information, including rates, was handwritten in it.
- It was challenging to manually add up the bills with or without calculators.

The following issues were faced by the staff:

- They constantly had to turn the diary pages to look up appointment information.
- In order to update consumers on the status of their appointments, I had to text or call them every time.
- Confused by the shifts, leaves, and holidays because there was no organized manner to handle them.

Customers encountered the following challenges as well:

- New clients had to physically visit the salon to schedule appointments.
- It was necessary to retain regular smartphone reminders for appointments.
- To find out the status of my appointment, I had to call the salon on my phone.
- Had to wait a long time for the manual payment receipt to be issued.

2.2Requirements

2.2.1 Functional Requirements

- **Appointments Calendar**

The appointment Calendar enables customers to book dates for their relevant services as they require. Further it also gives the authority to the system admin to block the holidays of the salon.

- **Appointment Management**

By this function of the application when a customer books a relevant date, the services of the need can be selected from the application. Confirmation of the appointment is sent to the customer through an email. Further the total payable of the customer is also displayed and notified. The application also gives the customer the customer to cancel their appointments anytime.

- **Tailor Shop Management**

With this function of the application the tailor will be able to view, edit and add new customer measurements and order details.

2.2.2 Non-functional Requirements

Basically, non-functional requirements relate to qualities of the system that cut across user facing features, such as security, reliability, and performance.

- **Accessibility** – The system should be able to be accessed anywhere at any time by the authorized users.
- **Accuracy** – The correctness of data inputs to the system should be ensured.
- **Availability** – System should be available within working hours. But can be used at special occasions also. E.g.: - At a bridal dressing
- **Efficiency** – Users should be given the facility to perform the salon and tailor shop management processes correctly through the salon and tailor shop sales record management system.
- **Effectiveness** – Users should be given the facility to perform correct salon and tailor shop management processes via the suggested system.
- **Maintainability** – This is a considerable factor, especially for a non-technical user. The maintainability of the system should not be more complex.
- **Privacy** – The confidentiality of the data input to the system should be assured.
- **Reliability** - Ability of the suggested system to function under stated conditions for a specified period should be assured.
- **Robustness** – When handling payments this function should be considered.
- **Security** – The data feeds to the system should be protected by controlling the user access privileges.

2.3 Feasibility study

2.3.1 Technical feasibility

The system is being developed in android studio and it provides comprehensive functions to make it user friendly.

2.3.2 Social feasibility

As this system is user-friendly and flexible some problems will also be solved which employees may be facing when using existing manual system. So, we can say that system is socially feasible.

2.3.3 Economic feasibility

The cost of converting from a manual system to a new automatic computerized system is probably not more. For construction of the new system, the not rooms and its facilities are available, so it does not require any extra resources, only the software requirement is there.

3.0 System Design

The process of defining the elements of a system such as the architecture, modules, and components, the various interfaces of those components, and the data that flows through that system is known as system design. The IEEE defines design as "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" as well as "the result of [that] process." It is intended to meet unique goals and requirements of a business or organization by creating a cohesive and well-functioning system.

3.1 System Methodology

3.2 System Deliverables

Process modeling and data modeling are two modeling methodologies that will give information about the proposed system's deliverables. The data model is a conceptual model that depicts the physical and logical representations of system elements and their interactions. This modeling method was used to create the system's Entity Relationship Diagram and Class Diagram. Under the process modeling method, UML, Use Case Diagrams, Activity Diagrams, and Sequence Diagrams will be possible to design for the suggested system.

3.2.1 Entity Relationship Diagram

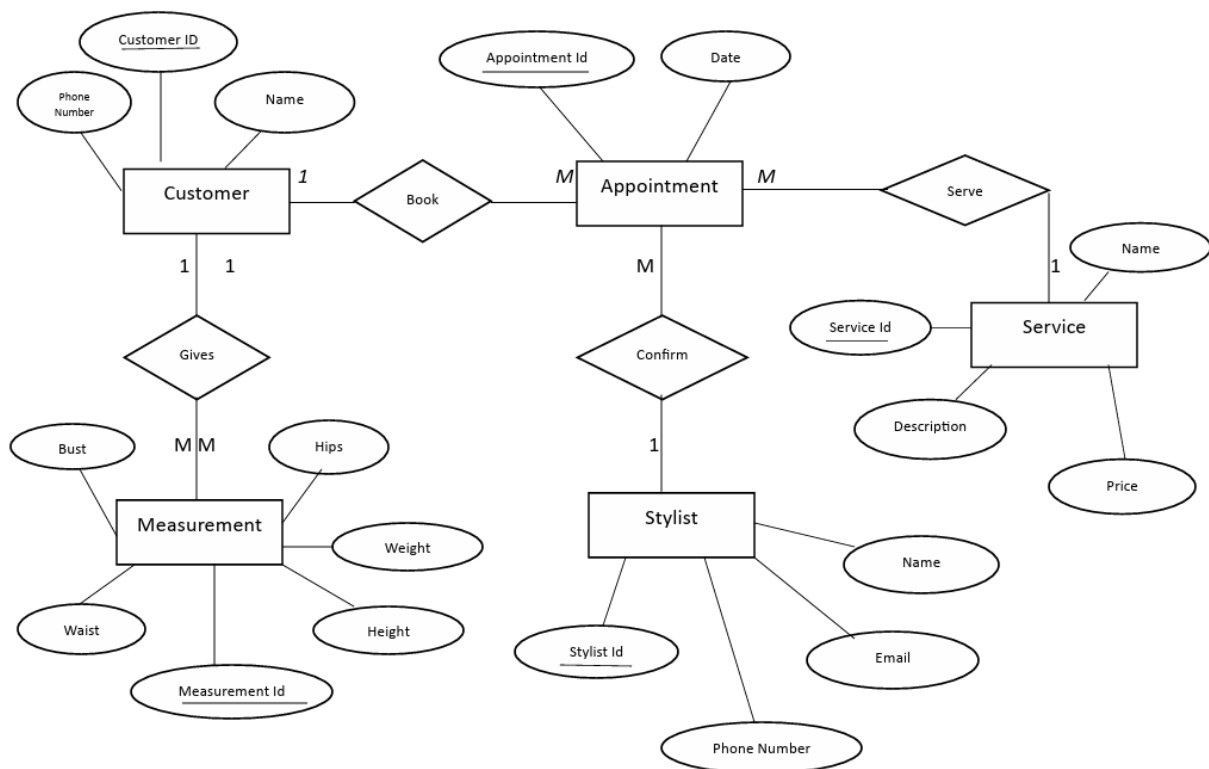


Figure 02: Entity Relationship diagram

An entity-relationship model describes the relationships between objects of interest in a certain domain of knowledge. A simple ER model is made up of entity types and defines the relationships that can exist between them.

ER diagram represents a salon appointment and measurement management system. It captures the relationships and entities involved in the system.

The main entities identified in the diagram are:

1. Customer: Represents salon customers and includes attributes like customer ID, name, and phone number.
2. Appointment: Represents a specific appointment made by a customer. It includes attributes such as appointment ID and date/time. It is associated with a customer, stylist, and service.
3. Stylist: Represents salon stylists and includes attributes like stylist ID, name, phone number, and email.
4. Service: Represents salon services offered. It includes attributes like service ID, name, description, and price.
5. Measurement: Represents customer measurements. It includes attributes like measurement ID, height, weight, bust, waist, and hips. It is associated with a customer.

The relationships captured in the diagram are:

1. Customer has a one-to-many relationship with Appointment, indicating that a customer can have multiple appointments.
2. Customer has a one-to-many relationship with Measurement, indicating that a customer can have multiple measurements.
3. Appointment has a many-to-one relationship with Stylist, indicating that multiple appointments can be assigned to a stylist.
4. Appointment has a many-to-one relationship with Service, indicating that multiple appointments can include the same service.
5. Measurement has a many-to-one relationship with Customer, indicating that multiple measurements belong to a customer.

The diagram highlights the key entities and their attributes, as well as the relationships between them, providing an overview.

3.2.2 Class Diagram

A class diagram is a graphical depiction of the system's static view that represents many parts of the application. The entire system is represented by a series of class diagrams.

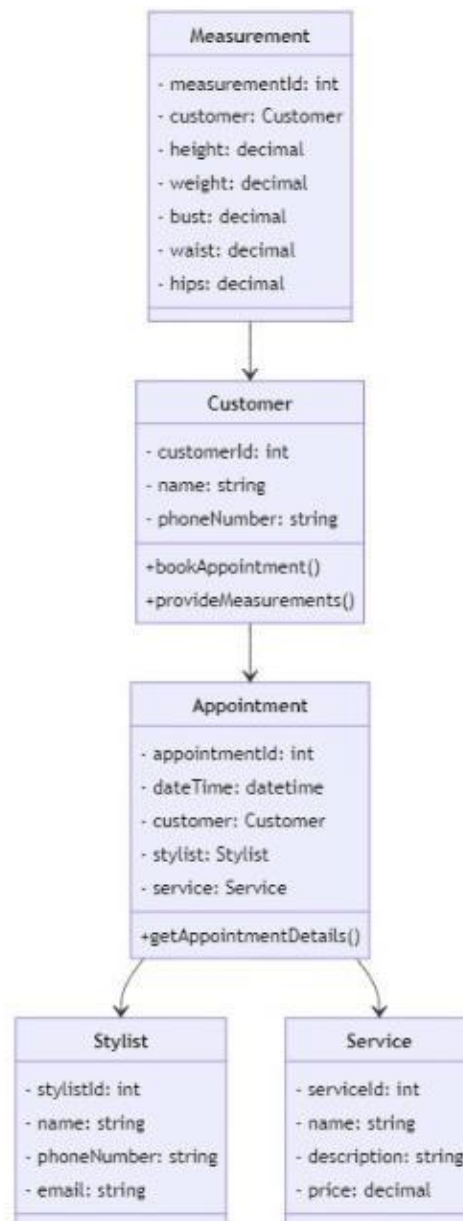


Figure 03: class diagram

Class diagram represents the classes and their relationships in a salon appointment and measurement management system.

The main classes identified in the diagram are:

1. Customer: Represents salon customers and includes attributes like customer ID, name, and phone number. It also has methods such as `bookAppointment()` and `provideMeasurements()` to allow customers to book appointments and provide their measurements.
2. Appointment: Represents a specific appointment made by a customer. It includes attributes such as appointment ID and date/time. It has a composition relationship with the Customer class, indicating that an appointment is associated with a specific customer. It also has associations with the Stylist and Service classes, indicating that an appointment can be assigned to a stylist and include a specific service. The Appointment class provides a method `getAppointmentDetails()` to retrieve the details of the appointment.
3. Stylist: Represents salon stylists and includes attributes like stylist ID, name, phone number, and email.
4. Service: Represents salon services offered. It includes attributes like service ID, name, description, and price.
5. Measurement: Represents customer measurements and includes attributes like measurement ID, height, weight, bust, waist, and hips. It has a composition relationship with the Customer class, indicating that measurements belong to a specific customer.

The relationships between the classes are depicted through arrows. For example, the Customer class is associated with the Appointment and Measurement classes, indicating that a customer can have multiple appointments and measurements. The Appointment class has associations with the Stylist and Service classes, indicate.

3.2.3 Use Case

Use case is a description of the set of sequins of the actions that a system performs that yields an observable result of values to a particular thing in a model.

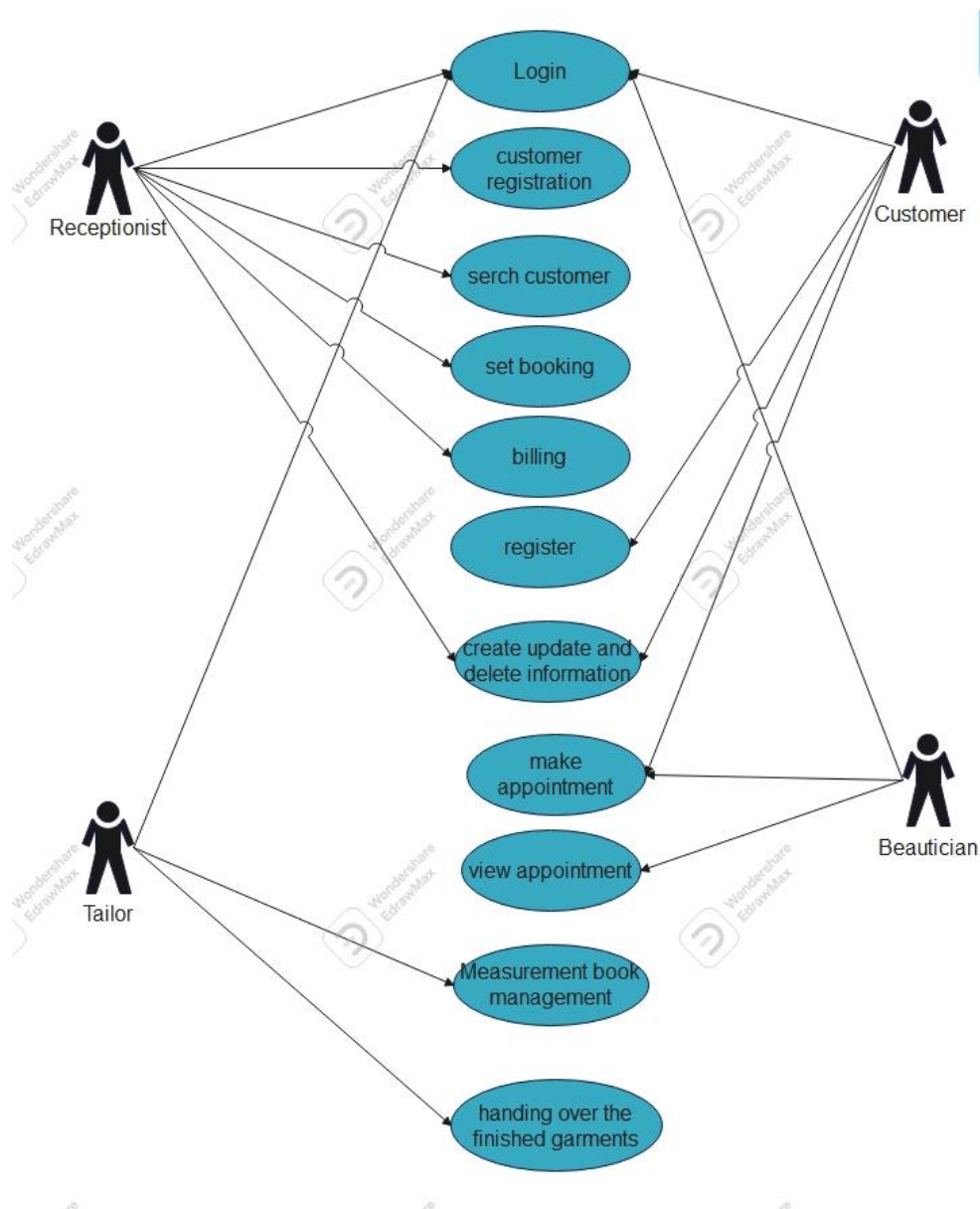


Figure 04: Use case diagram

3.2.4 Activity Diagram

Similar to a flowchart or data flow diagram, an activity diagram visually displays a series of actions or the flow of control in a system. A common tool in business process modeling is the activity diagram. Additionally, they can outline the procedures in a use case diagram.

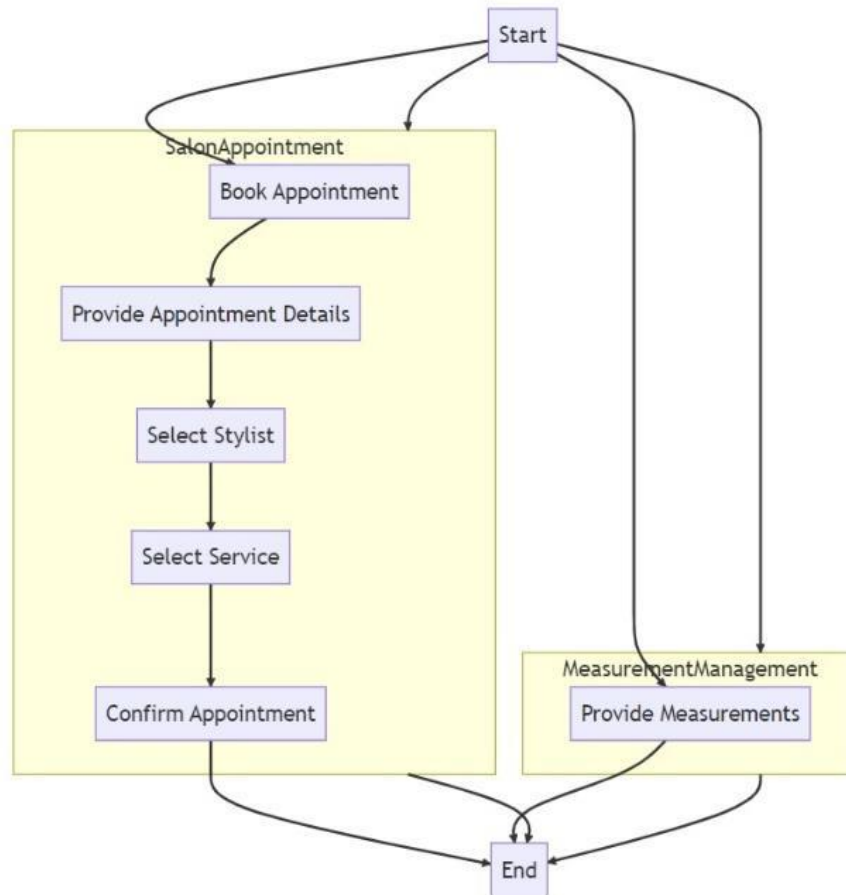


Figure 05: Activity diagram

Activity diagram represents the activities and their flow in the salon appointment and measurement management system.

The main activities depicted in the diagram are:

1. Salon Appointment: Represents the activities related to booking and confirming appointments.
2. Measurement Management: Represents the activity of providing measurements.

Under the "Salon Appointment" activity:

1. The activity starts with the "Start" node.
2. The customer initiates the process by performing the "Book Appointment" action.
3. The customer then proceeds to provide appointment details through the "Provide Details" activity.
4. After providing the details, the customer selects a stylist through the "Select Stylist" activity.
5. Following that, the customer selects a service from the available options through the "Select Service" activity.
6. Once the selections are made, the customer confirms the appointment using the "Confirm Appointment" activity.
7. The activity ends with the "End" node.

Under the "Measurement Management" activity:

1. The activity starts with the "Start" node.
2. The customer provides measurements as part of the "Provide Measurements" activity.
3. The activity ends with the "End" node.

The diagram showcases the flow of activities in the system, starting from the "Start" node and ending at the respective "End" nodes. It visualizes the steps involved in booking an appointment, selecting a stylist and service, confirming the appointment, and providing measurements. The activity diagram provides a high-level overview.

3.2.5 Sequence Diagram

A sequence diagram is a form of interaction diagram in that it describes how and in what order a set of items interact with one another. Software engineers and business experts use these diagrams to understand the requirements for a new system or to describe an existing process

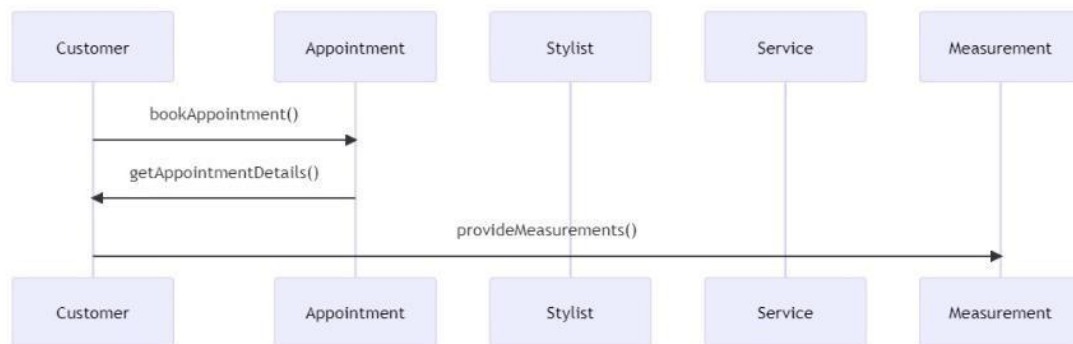


Figure 06: Sequence diagram

Sequence diagram illustrates the interaction between different participants in the salon appointment and measurement management system.

The participants in the diagram are:

1. Customer: Represents a customer who wants to interact with the system.
2. Appointment: Represents the appointment component of the system responsible for handling appointment-related actions.
3. Stylist: Represents a stylist who is assigned to the appointment.
4. Service: Represents a service that can be selected for the appointment.
5. Measurement: Represents the measurement component of the system responsible for handling measurements provided by the customer.

The sequence of actions in the diagram is as follows:

1. The Customer initiates the interaction by calling the ``bookAppointment()`` method on the Appointment participant. This indicates the customer's intent to book an appointment.
2. The Appointment participant receives the request and communicates with the Customer participant to gather the necessary appointment details using the ``getAppointmentDetails()`` method.
3. The Appointment participant then interacts with the Stylist participant, possibly through some internal process or method call represented by the ``SelectStylist`` step. This step involves selecting an appropriate stylist for the appointment.
4. Next, the Appointment participant communicates with the Service participant, represented by the ``SelectService`` step. This step involves selecting a service to be included in the appointment.
5. Once the appointment details and services are finalized, the Appointment participant confirms the appointment using the ``ConfirmAppointment`` step.
6. The sequence diagram ends with the successful completion of the appointment process.

This sequence diagram provides a visual representation of the order and flow of interactions between the participants in the salon appointment and measurement management system. It illustrates the steps involved in booking an appointment, selecting a stylist and service, and confirming the appointment.

3.3 User Interfaces

3.3.1 Registration Interface

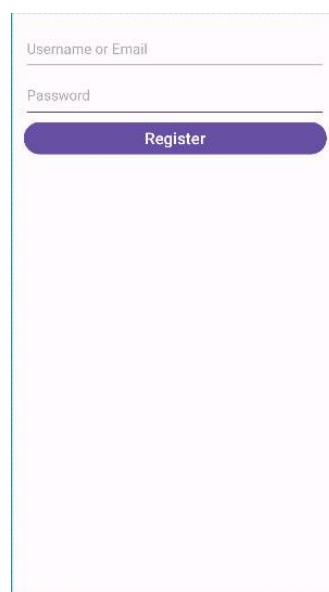
A registration form with a light purple background. It features two input fields: 'Username or Email' and 'Password', both with light blue borders. Below the 'Password' field is a purple rounded rectangular button with the text 'Register' in white. The form is enclosed in a thin blue border.

Figure 07: Registration interface

XML code

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".RegistrationActivity">

    <EditText
        android:id="@+id/txtusername"
        android:layout_width="378dp"
        android:layout_height="57dp"
        android:hint="Username or Email" />

    <EditText
        android:id="@+id/txtpassword"
        android:layout_width="378dp"
        android:layout_height="59dp"
        android:hint="Password"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btnregister"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Register"
        android:textSize="20sp" />

</LinearLayout>
```

Kotlin code

```
class Registration : AppCompatActivity() {
    @SuppressWarnings("MissingInflatedId")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_registration)
        val usernameEditText = findViewById<EditText>(R.id.txtusername)
        val passwordEditText = findViewById<EditText>(R.id.txtpassword)
        val registerButton = findViewById<Button>(R.id.btnregister)
        val db=DBHelper(this,null)

        registerButton.setOnClickListener {
            val username = usernameEditText.text.toString().toLowerCase().trim()
            val password = passwordEditText.text.toString().toLowerCase().trim()
        }
    }
}
```

```

if (username.isEmpty() && password.isEmpty()) {

    db.insert1(username, password)
    Toast.makeText(this, "Registration Successful", Toast.LENGTH_SHORT).show()
    finish()
} else {
    Toast.makeText(this, "Please enter username and password",
Toast.LENGTH_SHORT)
        .show()
    }
}
}
}
}

```

3.3.2 Login Interface

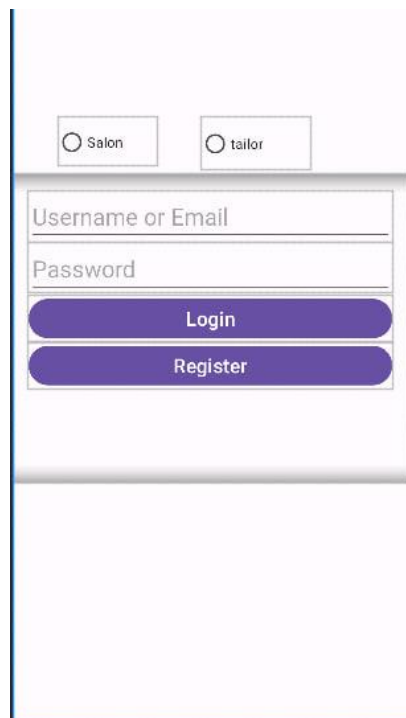


Figure 08: Login interface

XML code

```

</LinearLayout>

<RadioButton
    android:id="@+id/rdbsalon"
    android:layout_width="102dp"
    android:layout_height="51dp"
    android:layout_marginStart="40dp"
    android:layout_marginTop="112dp"

```

```

        android:text="Salon"
        app:layout_constraintEnd_toStartOf="@+id/rdbtailor"
        app:layout_constraintStart_toStartOf="parent"
<LinearLayout
    android:layout_width="406dp"
    android:layout_height="316dp"
    android:layout_marginTop="171dp"
    android:layout_marginEnd="5dp"
    android:layout_marginBottom="244dp"
    android:orientation="vertical"
    android:padding="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:context=".LoginActivity">

```

<EditText

```

    android:id="@+id/txtusername"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Username or Email"
    android:textSize="24sp"
    tools:ignore="DuplicateIds"
    android:autofillHints="" />

```

<EditText

```

    android:id="@+id/txtpassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Password"
    android:inputType="textPassword"
    android:textSize="24sp"
    tools:ignore="DuplicateIds" />

```

<Button

```

    android:id="@+id/btnlogin"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Login"
    android:textSize="20sp" />

```

<Button

```

    android:id="@+id/btnregister"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Register"
    android:textSize="20sp" />

```

```

app:layout_constraintTop_toTopOf="parent" />

<RadioButton
    android:id="@+id/rdbtailor"
    android:layout_width="112dp"
    android:layout_height="54dp"
    android:layout_marginStart="40dp"
    android:layout_marginTop="112dp"
    android:layout_marginEnd="102dp"
    android:text="tailor"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/rdbsalon"
    app:layout_constraintTop_toTopOf="parent" />

```

Kotlin code

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val db=DBHelper(this,null)

        val usernameEditText = findViewById<EditText>(R.id.txtusername)
        val passwordEditText = findViewById<EditText>(R.id.txtpassword)
        val loginButton = findViewById<Button>(R.id.btnlogin)
        val registerButton = findViewById<Button>(R.id.btnregister)
        val rdbsalon = findViewById<RadioButton>(R.id.rdbsalon)
        val rdbtailor = findViewById<RadioButton>(R.id.rdbtailor)

        loginButton.setOnClickListener {
            val username = usernameEditText.text.toString()
            val password = passwordEditText.text.toString()

            if (username.isNotEmpty() && password.isNotEmpty()){
                if (rdbsalon.isChecked){
                    val intent = Intent(this, ScheduleAppointment::class.java)
                    intent.putExtra("KEY_username", username)
                    startActivity(intent)
                } if (rdbtailor.isChecked){
                    val intent = Intent(this, MesurementManagement::class.java)
                    intent.putExtra("KEY_username", username)
                    startActivity(intent)
                }
                Toast.makeText(this, "Login Successful", Toast.LENGTH_SHORT).show()
            } else{
                Toast.makeText(this, "Select shop", Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

```

    }

    } else {
        Toast.makeText(this, "Please enter username and password",
Toast.LENGTH_SHORT).show()
    }
}

registerButton.setOnClickListener {
    navigateToRegistration()
}

}

private fun navigateToRegistration() {
    val intent = Intent(this, Registration::class.java)
    startActivity(intent)
}

}

```

3.3.3 Scheduling Interface

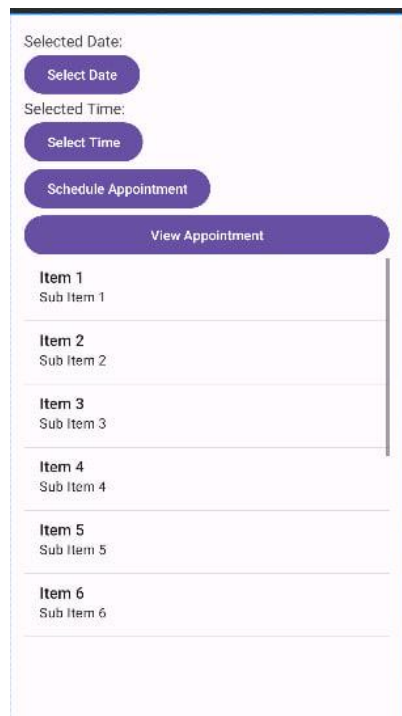


Figure 09: Scheduling Interface

XML code

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".ScheduleAppointmentActivity">

    <TextView
        android:id="@+id/tvSelectedDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Selected Date: "
        android:textSize="16sp" />

    <Button
        android:id="@+id/btnSelectDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Date" />

    <TextView
        android:id="@+id/tvSelectedTime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Selected Time: "
        android:textSize="16sp" />

    <Button
        android:id="@+id/btnSelectTime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Time" />

    <Button
        android:id="@+id/btnSchedule"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Schedule Appointment" />

    <Button
        android:id="@+id/btnview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="View Appointment" />

    <ListView
        android:id="@+id/listview4"
```

```
android:layout_width="wrap_content"
android:layout_height="267dp" />
```

```
</LinearLayout>
```

Kotlin code

```
class ScheduleAppointment : AppCompatActivity() {
    private lateinit var selectedDate: String
    private lateinit var selectedTime: String
    private lateinit var calendar: Calendar
    private var appointmentsCount: Int = 0
    private lateinit var selectedTimeButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_schedule_appointment)

        val btnSelectDate = findViewById<Button>(R.id.btnSelectDate)
        val btnSelectTime = findViewById<Button>(R.id.btnSelectTime)
        val btnSchedule = findViewById<Button>(R.id.btnSchedule)
        val btnview = findViewById<Button>(R.id.btnview)
        btnSelectDate.setOnClickListener {
            showDatePicker()
        }

        btnSelectTime.setOnClickListener {
            // Time selection logic here
        }

        btnview.setOnClickListener{
            val intentReceiver: Intent = intent
            val receivedFROM: String? = intentReceiver.getStringExtra( "username")
            val username =receivedFROM.toString()
            val subject_list: ArrayList<Subject> = ArrayList()
            val subjectList= findViewById<ListView>(R.id.listview4)
            val adapter = Adpater1(this, subject_list)
            subjectList.adapter = adapter
            val subject = Subject("1",username,selectedDate, selectedTime, "")
            subject_list.add(subject)
        }

        btnSchedule.setOnClickListener {
            if (selectedDate.isNotEmpty() && selectedTime.isNotEmpty()) {
                // Check if the selected time slot is available and not blocked by the admin
                if (isTimeSlotAvailable(selectedDate, selectedTime)) {
                    // Check if the appointment limit for the selected date is not exceeded
                }
            }
        }
    }
}
```

```

        if (isAppointmentLimitNotExceeded(selectedDate)) {
            // Proceed with scheduling the appointment
            scheduleAppointment(selectedDate, selectedTime)
            Toast.makeText(
                this,
                "Appointment scheduled successfully!",
                Toast.LENGTH_SHORT
            ).show()
            finish()
        } else {
            Toast.makeText(
                this,
                "Appointment limit for the selected date has been reached.",
                Toast.LENGTH_SHORT
            ).show()
        }
    } else {
        Toast.makeText(
            this,
            "Selected time slot is not available.",
            Toast.LENGTH_SHORT
        ).show()
    }
} else {
    Toast.makeText(this, "Please select a date and time.", Toast.LENGTH_SHORT)
        .show()
}
}
}

@SuppressLint("SetTextI18n")
private fun showDatePicker() {
    calendar = Calendar.getInstance()
    val year = calendar.get(Calendar.YEAR)
    val month = calendar.get(Calendar.MONTH)
    val day = calendar.get(Calendar.DAY_OF_MONTH)
    val tvSelectedTime = findViewById<TextView>(R.id.tvSelectedTime)
    val tvSelectedDate = findViewById<TextView>(R.id.tvSelectedDate)

    val datePicker = DatePickerDialog(this, { _, selectedYear, selectedMonth, selectedDay -
>
        calendar.set(selectedYear, selectedMonth, selectedDay)
        selectedDate =
            SimpleDateFormat("dd/MM/yyyy", Locale.getDefault()).format(calendar.time)
        tvSelectedDate.text = "Selected Date: $selectedDate"
    }, year, month, day)

    datePicker.datePicker.minDate = System.currentTimeMillis()
    datePicker.show()
}

```

```

private fun isTimeSlotAvailable(date: String, time: String): Boolean {
    // Check if the selected time slot is available and not blocked by the admin
    return true
}

private fun isAppointmentLimitNotExceeded(date: String): Boolean {

    return appointmentsCount < 20
}

private fun scheduleAppointment(date: String, time: String) {
    //Appointment scheduling logic here

}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_schedule_appointment)

    selectedTimeButton = findViewById(R.id.btnSelectTime)
    selectedTimeButton.setOnClickListener { showTimePickerDialog() }
}

private fun showTimePickerDialog() {
    val currentTime = Calendar.getInstance()
    val hour = currentTime.get(Calendar.HOUR_OF_DAY)
    val minute = currentTime.get(Calendar.MINUTE)

    val timePickerDialog = TimePickerDialog(
        this,
        { _, selectedHour, selectedMinute ->
            val selectedTime = formatTime(selectedHour, selectedMinute)
            selectedTimeButton.text = selectedTime
        },
        hour,
        minute,
        false
    )
    timePickerDialog.show()
}

private fun formatTime(hour: Int, minute: Int): String {
    val calendar = Calendar.getInstance()
    calendar.set(Calendar.HOUR_OF_DAY, hour)
    calendar.set(Calendar.MINUTE, minute)

    val timeFormat = SimpleDateFormat("hh:mm a", Locale.getDefault())

```

```

        return timeFormat.format(calendar.time)
    }
}

```

3.3.4 Service selection Interface

Figure 10: Service selection interface

XML code

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:ignore="MissingConstraints">

    <TextView
        android:id="@+id/tvSelectService"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:textStyle="bold"
        android:padding="16dp"

```

```
android:text="Select Service" />
```

<RadioGroup

```
android:id="@+id/radioGroupServices"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:orientation="vertical">
```

<RadioButton

```
android:id="@+id/radioButtonBridal"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="Bridal"  
android:textSize="16sp" />
```

<RadioButton

```
android:id="@+id/radioButtonHair"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="Hair"  
android:textSize="16sp" />
```

<RadioButton

```
android:id="@+id/radioButtonBeauty"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="Beauty"  
android:textSize="16sp" />
```

</RadioGroup>

<TextView

```
android:id="@+id/tvCustomization"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:textSize="18sp"  
android:textStyle="bold"  
android:padding="16dp"  
android:text="Select Customization" />
```

<CheckBox

```
android:id="@+id/checkboxCustomization1"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="Hair styles"  
android:textSize="16sp" />
```

<CheckBox

```
android:id="@+id/checkboxCustomization2"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Makeups"
        android:textSize="16sp" />

<CheckBox
    android:id="@+id/checkBoxCustomization3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Dressing"
    android:textSize="16sp" />

<CheckBox
    android:id="@+id/checkBoxCustomization4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Saree Designing"
    android:textSize="16sp" />

<CheckBox
    android:id="@+id/checkBoxCustomization5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="All services of bride's maids and flower girls including flower bouquets"
    android:textSize="16sp" />

<Button
    android:id="@+id/btnConfirm"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Confirm"
    android:textSize="20sp" />

</LinearLayout>

```

Kotlin code

```

class ServiceSelection : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_service_selection)

        val radioGroupServices = findViewById<RadioGroup>(R.id.radioGroupServices)
        val checkBoxCustomization1 =
            findViewById<CheckBox>(R.id.checkBoxCustomization1)
        val checkBoxCustomization2 =
            findViewById<CheckBox>(R.id.checkBoxCustomization2)
    }
}

```

```

        val checkBoxCustomization3 =
            findViewById<CheckBox>(R.id.checkBoxCustomization3)
        // Initialize more CheckBox variables as needed

        val btnConfirm = findViewById<Button>(R.id.btnConfirm)

        btnConfirm.setOnClickListener {
            val selectedService = getSelectedService()
            val selectedCustomizations = getSelectedCustomizations()

            if (selectedService.isNotEmpty()) {
                // Process the selected service
                Toast.makeText(this, "Selected Service: $selectedService",
                    Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "Please select a service", Toast.LENGTH_SHORT).show()
            }

            if (selectedCustomizations.isNotEmpty()) {
                // Process the selected customizations
                val customizationsText = selectedCustomizations.joinToString(", ")
                Toast.makeText(this, "Selected Customizations: $customizationsText",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun getSelectedService(): String {
        val radioGroupServices = findViewById<RadioGroup>(R.id.radioGroupServices)
        val radioButtonId = radioGroupServices.checkedRadioButtonId
        return findViewById<RadioButton>(radioButtonId)?.text?.toString() ?: ""
    }

    private fun getSelectedCustomizations(): List<String> {
        val checkBoxCustomization1 =
            findViewById<CheckBox>(R.id.checkBoxCustomization1)
        val checkBoxCustomization2 =
            findViewById<CheckBox>(R.id.checkBoxCustomization2)
        val checkBoxCustomization3 =
            findViewById<CheckBox>(R.id.checkBoxCustomization3)
        val selectedCustomizations = mutableListOf<String>()

        if (checkBoxCustomization1.isChecked) {
            selectedCustomizations.add(checkBoxCustomization1.text.toString())
        }
        if (checkBoxCustomization2.isChecked) {
            selectedCustomizations.add(checkBoxCustomization2.text.toString())
        }
        if (checkBoxCustomization3.isChecked) {
            selectedCustomizations.add(checkBoxCustomization3.text.toString())
        }
    }

```



```

    }
    // Add more CheckBox conditions as needed

    return selectedCustomizations
}
}

```

3.3.5 Measurement Interface

Figure 11: Measurement Interface

XML code

```

<TextView
    android:id="@+id/tventer"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"
    android:text="Enter your measurements"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```
<EditText
    android:id="@+id/etBust"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="2dp"
    android:hint="Bust"
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tventer" />
```

```
<EditText
    android:id="@+id/etWaist"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Waist"
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etBust" />
```

```
<EditText
    android:id="@+id/etHip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:hint="Hip"
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etWaist" />
```

```
<Button
    android:id="@+id/btnSaveMeasurements"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Save Measurements"
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etHip" />
```

```
<ListView
    android:id="@+id/listview32"
    android:layout_width="401dp"
    android:layout_height="377dp"
    android:layout_marginStart="1dp"
```

```

android:layout_marginTop="1dp"
android:layout_marginEnd="9dp"
android:layout_marginBottom="85dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/btnSaveMeasurements"
app:layout_constraintVertical_bias="0.0" />

```

Kotlin code

```

class MeasurementManagement : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_measurement_management)
        val intentReceiver: Intent = intent
        val receivedFROM: String? = intentReceiver.getStringExtra( "username")
        val etBust = findViewById<EditText>(R.id.etBust)
        val etWaist = findViewById<EditText>(R.id.etWaist)
        val etHip = findViewById<EditText>(R.id.etHip)
        val btnSaveMeasurements = findViewById<Button>(R.id.btnSaveMeasurements)

        btnSaveMeasurements.setOnClickListener {
            val subject_list: ArrayList<Subject> = ArrayList()
            val subjectList= findViewById<ListView>(R.id.listview32)
            val adapter = Adpater1(this, subject_list)
            subjectList.adapter = adapter

            val bust = etBust.text.toString()
            val waist = etWaist.text.toString()
            val hip = etHip.text.toString()

            if (bust.isNotEmpty() && waist.isNotEmpty() && hip.isNotEmpty()) {
                // Process the measurements
                Toast.makeText(this, "Measurements saved successfully!",
                    Toast.LENGTH_SHORT).show()
                val subject = Subject("1",bust, waist, hip)
                subject_list.add(subject)
            } else {
                Toast.makeText(this, "Please enter all measurements",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

```
}  
}
```

3.3.6 App icon



Figure 12: App icon

4.0 Implementation

This is the phase that software becomes executable. The goal of this phase is to implement the design using selected programming language in the best possible manner. Implementation affects both testing and maintenance phases and also it would be the most time-consuming stage. This phase was started with installing system environments and tools which were identified.

4.1 Software Environment

- Operating System – Microsoft Windows 7/8/10 (32 or 64 bit)
- Front End – Android Studio
- Back End – MySQL

4.2 Hardware Environment

- Processor - Intel Core i3 or above Processor
- Hard Disk – 500 MB disk space
- RAM – 2 GB
- 1 GB for Android SDK
- 1280x800 screen resolution

4.3 Technologies to be used

- Front End – Android Studio
- Back End – MySQL, Visual Studio Code

4.4 Development Tools

- **Android Studio**

Android Studio is an integrated development environment (IDE) used for developing Android applications. It is the official IDE for Android app development and provides a range of features including a code editor, layout editor, Gradle-based build system, debugger, emulator, Android Virtual Device Manager, integration with Google services, and version control integration. Android Studio makes Android app development easier and more efficient.

- **MySQL**

MySQL is an open-source relational database management system (RDBMS) that is widely used for managing and storing data. It uses the Structured Query Language (SQL) for managing data and is known for its reliability, performance, and ease of use. MySQL is widely used for web applications, e-commerce sites, and content management systems, among other applications. It is a popular choice for both small and large businesses due to its ease of use, scalability, and reliability.

- **Visual Studio Code**

Visual Studio Code (VS Code) is a free and open-source code editor developed by Microsoft. It is a lightweight yet powerful tool that supports multiple programming languages and is available for Windows, macOS, and Linux operating systems. Overall, VS Code is a popular code editor among developers due to its ease of use, powerful features, and rich extension ecosystem.

4.5 Code Features

- **Database code**

```
class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
    SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {

    override fun onCreate(DB: SQLiteDatabase) {
        DB.execSQL(SQL_CREATE_user)
        DB.execSQL(SQL_CREATE_Appointment)
        DB.execSQL(SQL_CREATE_Customer)
        DB.execSQL(SQL_CREATE_Service)
        DB.execSQL(SQL_CREATE_measurement)
        DB.execSQL(SQL_CREATE_metrics)
        DB.execSQL(SQL_CREATE_Stylists)
```

```

}

override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    db.execSQL(SQL_DELETE_Service2)
    db.execSQL(SQL_DELETE_user)
    db.execSQL(SQL_DELETE_Appointment2)
    db.execSQL(SQL_DELETE_Customer)
    db.execSQL(SQL_DELETE_measurement)
    db.execSQL(SQL_DELETE_metrics)
    db.execSQL(SQL_DELETE_Stylists)
    onCreate(db)
}

fun insert1(username : String, password : String ) {
    val values = ContentValues()
    values.put(user_id1, getAutoId())
    values.put(username1, username)
    values.put(password1, password)

    val db = this.writableDatabase
    db.insert(usertable, null, values)
}

fun ServiceInsert(ServiceName : String, description : String, price : String) {
    val values = ContentValues()
    values.put(Service_id, getAutoId())
    values.put(Service_name, ServiceName)
    values.put(Description, description)
    values.put(Price, price)

    val db = this.writableDatabase
    db.insert(Servicetable, null, values)
}

fun StylistsInsert(StylistsName : String, Pnumber : String, email : String) {
    val values = ContentValues()
    values.put(Stylists_id, getAutoId())
    values.put(Stylists_name, StylistsName)
    values.put(phone_number1, Pnumber)
    values.put(Email, email)

    val db = this.writableDatabase
    db.insert(Styliststable, null, values)
}

fun CustomerInsert(CustomerName : String, pnumber : String, Email : String) {
    val values = ContentValues()
    values.put(Customer_id, getAutoId())
    values.put(customer_name, CustomerName)
    values.put(phone_number2, pnumber)
    values.put(email, Email)
}

```

```

        val db = this.writableDatabase
        db.insert(Customertable, null, values)
    }
    fun Appointmentinsert(customerid : String, stylistid : String, serviceid : String, endtime:
String, date2 : String ) {
        val values = ContentValues()
        values.put(Appointment_id, getAutoId())
        values.put(customer_id1, customerid)
        values.put(stylist_id2, stylistid)
        values.put(service_id3, serviceid)
        values.put(end_time, endtime)
        values.put(date, date2)

        val db = this.writableDatabase
        db.insert(Appointmenttable, null, values)
    }
    fun measurementinsert(metrics_id : String, value : String, LineStamp : String) {
        val values = ContentValues()
        values.put(measurement_id, getAutoId())
        values.put(metrics_id2, metrics_id)
        values.put(value1, value)
        values.put(lineStamp, LineStamp)

        val db = this.writableDatabase
        db.insert(measurementtable, null, values)
    }
    fun metricsinsert(metricsname : String, Description : String, unit : String) {
        val values = ContentValues()
        values.put(metrics_id, getAutoId())
        values.put(metrics_name, metricsname)
        values.put(description, Description)
        values.put(unit2, unit)

        val db = this.writableDatabase
        db.insert(metricstable, null, values)
    }
}

companion object{
    private const val DATABASE_NAME = "Salon"
    private const val DATABASE_VERSION = 1
    const val usertable = "user"
    const val user_id1 = "user_id"
    const val username1 = "usernameTEXT"
    const val password1 = "passwordTEXT"

    const val Servicetable = "Service"
    const val Service_id = "Service_id"
    const val Service_name = "Service_nameTEXT"
    const val Description = "DescriptionTEXT"
    const val Price= "PriceTEXT"
}

```

```

const val Appointmenttable = "Appointment"
const val Appointment_id = "Appointment_id"
const val customer_id1 = "customer_idTEXT"
const val stylist_id2 = "stylist_idTEXT"
const val service_id3= "service_idTEXT"
const val end_time= "end_timeTEXT"
const val date= "dateTEXT"

const val Styliststable = "Stylists"
const val Stylists_id = "Stylists_id"
const val Stylists_name = "Stylists_nameTEXT"
const val phone_number1 = "Phone_numberTEXT"
const val Email= "EmailTEXT"

const val Customertable = "Customer"
const val Customer_id = "Customer_id"
const val customer_name = "customer_nameTEXT"
const val phone_number2 = "phone_numberTEXT"
const val email= "emailTEXT"

const val metricstable = "metrics"
const val metrics_id = "metrics_id"
const val metrics_name = "metrics_nameTEXT"
const val description = "descriptionTEXT"
const val unit2= "unitTEXT"

const val measurementtable = "metrics"
const val measurement_id = "measurement_id"
const val metrics_id2 = "metrics_idTEXT"
const val value1 = "valueTEXT"
const val lineStamp= "lineStampTEXT"

fun getAutoId():Int{
    val random = Random as Random
    return random.nextInt(100)}
}

public fun getSchedule(): Cursor? {
    //Make database readable
    val db = this.readableDatabase
    val query = "select* from $Appointmenttable"
    return db.rawQuery(query, null)
}

public fun deleteItem(id: String){
    val db=writableDatabase
    db.delete(Appointmenttable, Appointment_id + "=" + id, null)
    db.close()
}

```



```

    }
    fun getSelectedData(from:String,to:String): Cursor? {
        val db = this.readableDatabase
        return db.rawQuery("SELECT * FROM user WHERE username = ? AND password = ?",
            arrayOf(from,to))
    }
}

object ServiceEntry : BaseColumns {
    const val Service2 = "Service"
    const val Service_id = "Service_id"
    const val Service_name = "Service_name"
    const val Description = "Description"
    const val Price= "Price"
}

const val SQL_CREATE_Service =
    "CREATE TABLE $Service2 (" +
        "$Service_id PRIMARY KEY," +
        "${Service_name}TEXT," +
        "${Description}TEXT,"+
        "${Price}TEXT)"

object StylistsEntry : BaseColumns {
    const val Stylists = "Stylists"
    const val Stylists_id = "Stylists_id"
    const val Stylists_name = "Stylists_name"
    const val Phone_number = "Phone_number"
    const val Email= "Email"
}

const val SQL_CREATE_Stylists =
    "CREATE TABLE $Stylists (" +
        "$Stylists_id PRIMARY KEY," +
        "${Stylists_name}TEXT," +
        "${Phone_number}TEXT,"+
        "${Email}TEXT)"

object AppointmentE : BaseColumns {
    const val Appointment2 = "Appointment"
    const val Appointment_id = "Appointment_id"
    const val customer_id1 = "customer_id"
    const val stylist_id2 = "stylist_id"
    const val service_id3= "service_id"
    const val end_time= "end_time"
    const val date= "date"
}

```

```
const val SQL_CREATE_Appointment =
    "CREATE TABLE $Appointment2 (" +
        "${Appointment_id} PRIMARY KEY," +
        "${customer_id1}TEXT," +
        "${stylist_id2}TEXT," +
        "${service_id3}TEXT," +
        "${end_time}TEXT," +
        "${date}TEXT)"
```

```
object CustomerE: BaseColumns {
    const val Customer = "Customer"
    const val Customer_id = "Customer_id"
    const val customer_name = "customer_name"
    const val phone_number = "phone_number"
    const val email= "email"
}
```

```
const val SQL_CREATE_Customer =
    "CREATE TABLE $Customer (" +
        "${Customer_id} PRIMARY KEY," +
        "${customer_name}TEXT," +
        "${phone_number}TEXT," +
        "${email}TEXT)"
```

```
object metricsE: BaseColumns {
    const val metrics = "metrics"
    const val metrics_id = "metrics_id"
    const val metrics_name = "metrics_name"
    const val description = "description"
    const val unit= "unit"
}
```

```
const val SQL_CREATE_metrics =
    "CREATE TABLE $metrics (" +
        "${metrics_id} PRIMARY KEY," +
        "${metrics_name}TEXT," +
        "${description}TEXT," +
        "${unit}TEXT)"
```

```
object measurementE: BaseColumns {
    const val measurement = "metrics"
    const val measurement_id = "Customer_id"
    const val metrics_id2 = "customer_name"
    const val value1 = "value"
    const val lineStamp= "lineStamp"
}
```

```
const val SQL_CREATE_measurement =
    "CREATE TABLE $measurement (" +
```

```

"$measurement_id PRIMARY KEY," +
"${metrics_id2}TEXT," +
"${value1}TEXT," +
"${lineStamp}TEXT)"

object userE: BaseColumns {
    const val user = "user"
    const val user_id = "user_id"
    const val username = "username"
    const val password = "password"
}

const val SQL_CREATE_user =
    "CREATE TABLE $user (" +
        "$user_id PRIMARY KEY," +
        "${username}TEXT," +
        "${password}TEXT)"

const val SQL_DELETE_Service2 = "DROP TABLE IF EXISTS $Service2"
const val SQL_DELETE_user = "DROP TABLE IF EXISTS $user"
const val SQL_DELETE_measurement = "DROP TABLE IF EXISTS $measurement"
const val SQL_DELETE_metrics = "DROP TABLE IF EXISTS $metrics"
const val SQL_DELETE_Customer = "DROP TABLE IF EXISTS $Customer"
const val SQL_DELETE_Appointment2 = "DROP TABLE IF EXISTS $Appointment2"
const val SQL_DELETE_Stylists = "DROP TABLE IF EXISTS $Stylists"

```

- **Custom Adapter -1 (Measurement view)**

```

class Adpater1 (private val context: Activity,
    private val subjectlist:ArrayList<Subject>

) : ArrayAdapter<Subject>(context, R.layout.card1, subjectlist ){
    @SuppressWarnings("ViewHolder", "InflateParams", "MissingInflatedId",
    "SetTextI18n")
    override fun getView(position: Int, view: View?, parent: ViewGroup): View {
        val subjectItem = context.layoutInflater.inflate(R.layout.card1, null, true)
        val lblTo: TextView = subjectItem.findViewById(R.id.lblbust)
        val lblFromTime: TextView = subjectItem.findViewById(R.id.lblwaist)
        val lblToTime: TextView = subjectItem.findViewById(R.id.lblhip)
        val bust : String = subjectlist[position].getN0()
        val waist: String = subjectlist[position].getFrom()
        val hip : String = subjectlist[position].getTime()

        lblTo.setText("Bust: $bust").toString()
    }
}

```

```

lblFromTime.setText("Waist: $waist").toString()
lblToTime.setText("Hip: $hip").toString()

```

```

    return subjectItem
}
}

```

- **Custom Adapter -2 (Appointment view)**

```

class adapter2 (private val context: Activity,
private val subjectlist:ArrayList<Subject>

) : ArrayAdapter<Subject>(context, R.layout.card1, subjectlist ){
    @SuppressWarnings("ViewHolder", "InflateParams", "MissingInflatedId")
    override fun getView(position: Int, view: View?, parent: ViewGroup): View {
        val subjectItem = context.layoutInflater.inflate(R.layout.card1, null, true)
        val lblTo: TextView = subjectItem.findViewById(R.id.lblbust)
        val lblFromTime: TextView = subjectItem.findViewById(R.id.lblwaist)
        val lblToTime: TextView = subjectItem.findViewById(R.id.lblhip)

        lblTo.setText(subjectlist[position].getN0())
        lblFromTime.setText(subjectlist[position].getFrom())
        lblToTime.setText(subjectlist[position].getTime())

        return subjectItem
    }
}

```

5.0 Evaluation and Testing

Software testing is a crucial procedure that involves running a software implementation with test data and assessing the software's outputs to see if it satisfies the requirements or not. Testing is a dynamic validation and verification technique. Verification relates to how effectively the system implements the required functions, while validation refers to whether the system satisfies the requirements.

5.1 Techniques of Software Testing

Software testing is done using two different methods.

5.1.1 Black Box Testing

Black box testing is a testing method that ignores the internal workings of the system and concentrates on the results produced in response to any input and system execution. Additionally known as functional testing.

5.1.2 White Box Testing

White box testing is a testing method that considers a system's internal workings. Glass box testing and structural testing are other names for it.

White box testing is frequently used for verification whereas black box testing is frequently utilized for validation.

5.2 Types of Testing

5.2.1 Unit Testing

The coding task includes performing unit tests. This stage is based on the software design for a specific piece of code. The following about the code should be demonstrated through unit testing:

- Robustness: The code must function flawlessly at all times.
- Functionally accurate: The code should work as expected under the given conditions.
- Correct interface: The code's inputs and outputs must match those specified in the design.

5.2.2 Integration Testing

After the different software modules have been tested, integration testing is conducted. The functional specification of the software serves as the foundation for integration testing. In this level, a bottom-up method to integration testing was utilized, where low-level components were integrated and tested prior to the development of high-level components.

5.2.3 System Testing

After the integration testing is done, system testing is conducted. This was done to test the system's non-functional needs. System testing is done to demonstrate that the software satisfies agreed-upon user criteria and functions as intended in the target environment. Both functional and non-functional requirements are covered by system testing.

5.2.4 Acceptance Testing

Customers frequently do acceptance testing to make sure the supplied product satisfies their needs and performs as expected. It belongs to the category of black box testing.

5.3 Test Plan and Test Case

A test plan is a comprehensive document that outlines the objectives, timeline, and deliverables of the test. It outlines the testing process, who will conduct the testing, what will be tested, how long the testing will last, and the level of quality at which the testing will be done. Testing will proceed with the test types stated above for this project. The following part will go over the test plan for the system's key module functionalities.

Test ID	Module Name	Test Description	Priority	Expected Results	Actual Results
T001	User Login	Verify login screen displays correctly	High	Login screen should display with fields	Login screen displayed correctly

T002	User Login	Login with valid credentials	High	User should be logged in successfully	User logged in successfully
T003	User Login	Login with invalid username or password	High	Error message should be displayed	Error message displayed for invalid credentials
T004	User Login	Login with empty username or password	High	Error message should be displayed	Error message displayed for empty fields
T005	User Login	Click on "Register here" link	Medium	Registration screen should be displayed	Registration screen displayed
T006	User Registration	Verify registration screen displays correctly	High	Registration screen should display with fields	Registration screen displayed correctly
T007	User Registration	Register with valid credentials	High	User should be registered successfully	User registered successfully
T008	User Registration	Register with existing username or email	High	Error message should be displayed	Error message displayed for existing credentials
T009	User Registration	Register with empty required fields	High	Error message should be displayed	Error message displayed for empty fields
T010	User Registration	Click on "Login here" link	High	Login screen should be displayed	Login screen displayed
T011	Date Selection	Verify if the user can select a date for the appointment	High	User is able to select a date and the selected date is displayed correctly in the "Selected Date" TextView	Passed

T012	Time Selection	Verify if the user can select a time for the appointment	High	User is able to select a time and the selected time is displayed correctly in the "Selected Time" TextView	Passed
T013	Schedule Appointment Button	Verify if the user can schedule an appointment when date and time are selected	High	Appointment is scheduled successfully and a success message is displayed	Passed
T014	Appointment Limit	Verify if the user is prevented from scheduling an appointment when the appointment limit for the selected date is exceeded	Medium	User receives an error message indicating that the appointment limit has been reached	Passed
T015	Blocked Time Slot	Verify if the user is prevented from scheduling an appointment during a blocked time slot	Medium	User receives an error message indicating that the selected time slot is not available	Passed
T016	Missing Date or Time	Verify if the user is prevented from scheduling an appointment when either the date or time is not selected	Medium	User receives an error message indicating that a date and time must be selected	Passed
T017	Service Selection	Verify if the user can select a service from the available options	High	User is able to select a service option and it is reflected in the UI	Passed

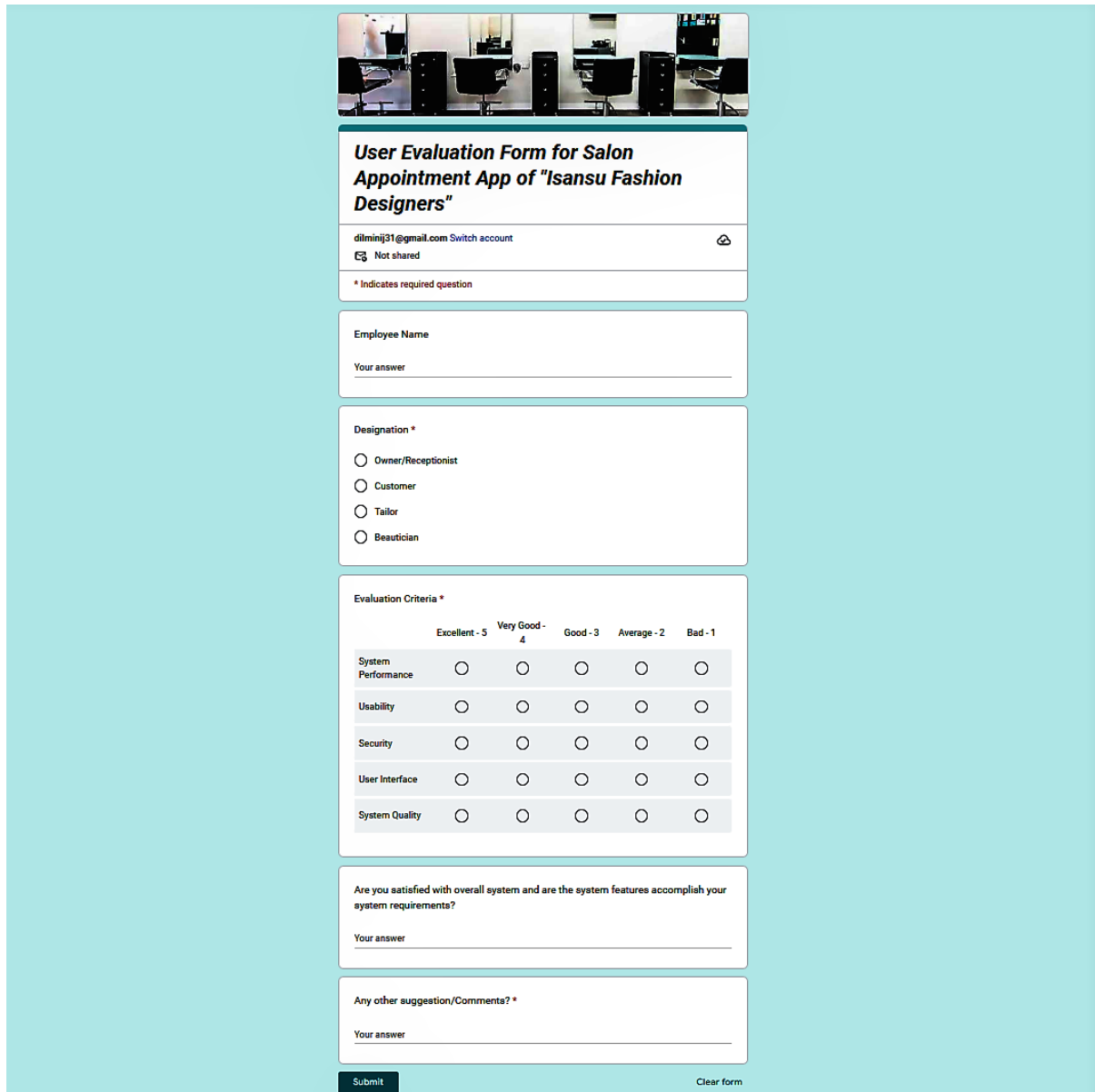
T018	Customization Selection	Verify if the user can select customization options for the selected service	High	User is able to select one or more customization options and they are displayed/updated in the UI	Passed
T019	Conform Button	Verify if the user can confirm the selected service and customization choices	High	Upon clicking the "Confirm" button, the selected service and customization choices are saved/processed accordingly	Passed
T020	Service Options Availability	Verify if the available service options are displayed correctly	Medium	The service options (e.g., Bridal, Hair, Beauty) are correctly displayed in the UI and the user can choose from the available options	Passed
T021	Customization Options Display	Verify if the customization options are displayed correctly for the selected service	Medium	The customization options associated with the selected service (e.g., Bridal - Hair styles, Makeups, Dressing) are correctly displayed in the UI	Passed
T022	Missing Service Selection	Verify if the user is prevented from proceeding without	Medium	If the user clicks the "Confirm" button without selecting a	Passed

		selecting a service		service, an error message is displayed prompting the user to select a service	
T023	Measurement	Save Measurements - Valid Input	High	Enter valid measurements for bust, waist, and hip. Click on "Save Measurements" button.	Passed
T024	Measurement	Save Measurements - Missing Measurement	Medium	Leave one of the measurement fields empty. Click on "Save Measurements" button.	Passed
T025	Measurement	Save Measurements - Empty Measurements	Medium	Leave all measurement fields empty. Click on "Save Measurements" button.	Passed
T026	Measurement	Save Measurements - Special Characters	Low	Enter measurements with special characters in the fields. Click on "Save Measurements" button.	Passed
T027	Measurement	Save Measurements - Large Measurements	Low	Enter very large values for bust, waist, and hip measurements. Click on "Save Measurements" button.	Passed
T028	Measurement	Save Measurements -	Low	Enter decimal values for bust,	Passed

		Decimal Measurements		waist, and hip measurements. Click on "Save Measurements" button.	
T029	Measurement	Save Measurements - Leading/Trailing Whitespace	Low	Enter measurements with leading or trailing whitespace. Click on "Save Measurements" button.	Passed
T030	Measurement	Save Measurements - Multiple Attempts	Low	Enter valid measurements for bust, waist, and hip. Click on "Save Measurements" button multiple times.	Passed

5.4 User Evaluation

The user evaluation was done by using a google form as follows.



The screenshot shows a Google Form titled "User Evaluation Form for Salon Appointment App of 'Isansu Fashion Designers'". The form is set by "dilmirij31@gmail.com" and is not shared. It includes a header image of a salon interior. The form contains the following sections:

- Employee Name**: A text input field with the placeholder "Your answer".
- Designation ***: A radio button selection with four options: Owner/Receptionist, Customer, Tailor, and Beautician.
- Evaluation Criteria ***: A table with five rows of criteria and five columns of rating options (Excellent - 5, Very Good - 4, Good - 3, Average - 2, Bad - 1). Each cell contains a radio button.
- Satisfaction Question**: A text input field for the question "Are you satisfied with overall system and are the system features accomplish your system requirements?" with the placeholder "Your answer".
- Comments ***: A text input field for the question "Any other suggestion/Comments? *" with the placeholder "Your answer".
- Submit**: A dark blue button at the bottom left.
- Clear form**: A link at the bottom right.

	Excellent - 5	Very Good - 4	Good - 3	Average - 2	Bad - 1
System Performance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Usability	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Security	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
User Interface	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
System Quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure13 : User Evaluation Google Form

Evaluation Google form was shared among three employees at the salon premise along with the salon owner: after implementing the system at the client site.

Selected employees were under the three user types maintained by the system. Main purpose of this was to get overall feedback on each area module of the system.

The form will also be shared among five regular customers of the salon and tailor shop and feedback of those customers also will be collected.

5.5 Use Evaluation Summary

Evaluation Criteria

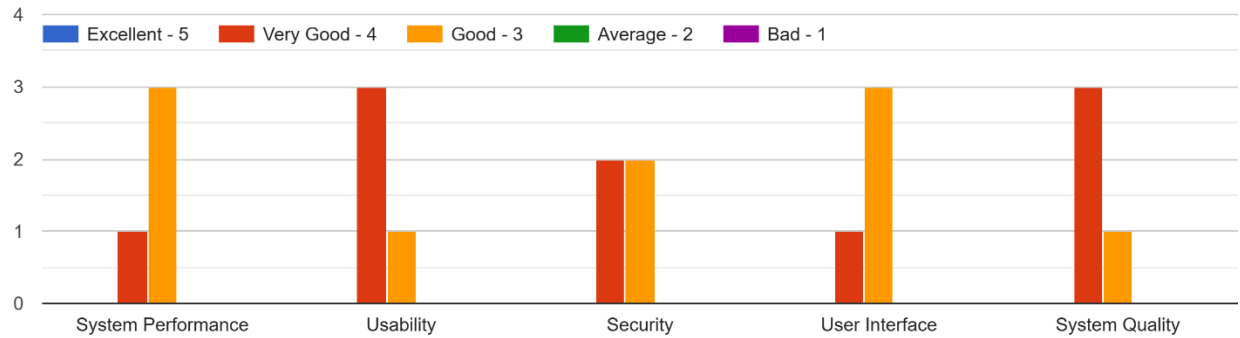


Figure 14: Use Evaluation Summary graph

The user feedbacks are overall in an average level as they suggest more improvements in the application. Main suggestions are to improve the speed of the system and make more attractive user interfaces.

6.0 Conclusion

In this 21st Century with the advancement of technology tailors and beauticians need to be updated to enhance their business. To do that an automated mobilization system is needed. So, we developed a system that helps to enhance their performance and manage the workload at the workplace. Our salon management system helps them to manage customer details, measurements, and customer appointments. The system not only helps to reduce the word load but also it helps to save time by reducing the paperwork. Most importantly the system is very easy to use, only the basic mobile knowledge is needed. This system can be more developed in the future with more advanced features so that these kinds of businessmen will be able to handle their business easily.

7.0 Future Enhancements

The Salon Management System can be enhanced in the future by adding the following features to the system.

- Adding a family-based portal: with this feature businessmen will be able to communicate easily with their client.
- Online orders can be done.
- Online design pages
- Payment Processing.

8.0 References

- [1] J. A. J. N. Jayaweera and P. D. Dilshan Navinda, "Online Salon Management System For Samudra Bridal Palace," 2017.
- [2] G. J. V. P. S. O. Jayawardena, "SALON MANAGEMENT SYSTEM FOR 'SALON NIROSHA,'" 2017.
- [3] B. J. Yang and R. Ibrahim, "Hair Technique Salon Management System Using Mobile Application," *Applied Information Technology And Computer Science*, vol. 2, no. 2, pp. 1013–1029, 2021, doi: 10.30880/aitcs.2021.02.02.063.

9.0 Appendix

1. Introduction:

The Salon Appointment and Measurement Management App is designed to streamline the process of managing salon appointments and customer measurements. This user manual provides instructions on how to use the various features of the app effectively.

2. Installation:

- Download the Salon Appointment and Measurement Management App from the App Store or Play Store.
- Install the app on your mobile device.
- Launch the app to get started.

3. Account Creation:

- Open the app and click on the "Sign Up" button.
- Fill in the required details, such as name, email, and password.
- Click on "Create Account" to create your account.
- Alternatively, you can use social media accounts to sign up (if supported).

4. Logging In:

- Launch the app and click on the "Log In" button.
- Enter your registered email and password.
- Click on "Log In" to access your account.

5. Dashboard:

- After logging in, you will be taken to the app dashboard.
- The dashboard provides an overview of your scheduled appointments and measurements.
- It may also display other relevant information, such as upcoming appointments, stylist availability, etc.

6. Salon Appointment:

- To schedule a new appointment, click on the "Schedule Appointment" button.
- Select the desired date and time from the available slots.
- Choose a stylist and the services you require.
- Confirm the appointment details and click on "Schedule" to book the appointment.
- You can view and manage your scheduled appointments from the dashboard.

7. Measurement Management:

- To add customer measurements, click on the "Add Measurement" button.
- Enter the required measurements, such as height, weight, bust, waist, etc.
- Save the measurements to associate them with the customer's profile.
- You can view and edit the measurements from the dashboard.

8. Stylist Management:

- If you are an administrator or salon owner, you can manage stylists.
- Click on the "Stylist Management" option in the menu.
- Add new stylists, remove existing ones, or edit their details.
- Ensure that stylist availability is up to date for accurate appointment scheduling.

9. Service Management:

- As an administrator or salon owner, you can manage services offered by the salon.
- Click on the "Service Management" option in the menu.
- Add new services, remove existing ones, or update service details.
- Specify service names and their corresponding prices.

10. Notifications:

- The app may send notifications to remind you of upcoming appointments.
- Ensure that you have enabled notifications for the app in your device settings.
- You can also check your notifications within the app for any updates or changes.

11. Settings:

- Access the app settings to customize your preferences.
- Adjust notification settings, account information, or other preferences.
- Ensure that your account details and contact information are up to date.

12. Help and Support:

- If you encounter any issues or have questions, refer to the app's Help section.
- The Help section provides answers to frequently asked questions and troubleshooting tips.
- If needed, contact customer support via the provided contact information.

13. Logout:

- To log out of your account, click on the "Logout" button in the menu.
- Confirm your action and ensure that you have saved any unsaved changes.