

Software Engineering FAST HMS

Project Report



May 2023

Ms. Noureen Fatima

Group Members

Ahad Shaikh (20K-0319)

Basil Ali Khan (20K-0477)

Ali Jodat (20K-0155)

ABSTRACT

Hospital Management System is a system which enables Admin to register and remove Doctors and check pharmacy medicines stocks. Doctors can only login can check their appointment timings and update them according to availability and update their data if they want. Individual Patients can register and login and check their hospital updated record. It enables patients to not only login and register themselves but also can booked appointments online with the availability of Doctor at particular time and date, they can also check if the hospital's pharmacy has medicine available and can buy them which results in generation of token number that will be showed to pharmacist. It is a user-friendly system which makes it easy to login, register, update records, check and book appointments. Admin can also add stocks when he sees something out of stock.

Table of Contents

Contents

Software Engineering..... 1

 Project Report..... 1

 Chapter 14

 INTRODUCTION4

 Chapter 2.....6

 REQUIREMENT SPECIFICATION6

 Chapter 3.....7

 DETAILED DESIGN7

 Chapter 4..... 12

 IMPLEMENTATION..... 12

 Chapter 5..... 17

 TESTING..... 17

 Chapter 6.....18

 SNAPSHOTS.....18

 Chapter 7.....26

 CONCLUSION26

 Chapter 8.....27

 FUTURE ENHACEMENTS27

 REFERENCES 28

Chapter 1

INTRODUCTION

1.1 OVERVIEW

Hospital and Management System is designed to help Patients and Doctors to register, login, update their records. It enables patients to book appointment and check medicine availability at pharmacy and book them with token number. It also enables patients to view specialist according to their symptoms and booked appointments. It also allows them to view their total profiles, appointment timing and stock availability.

1.2 PROBLEM STATEMENT

The main aim of Hospital Management System is to make an easy interface for Admin, Patients and Doctors to login, register, and update and help them buy medicines and booked appointment of their choice.

1.3 SUPABASE

SupaBase is open source. We choose open source tools which are scalable and make them simple to use.

SupaBase is not a 1-to-1 mapping of Firebase. While we are building many of the features that Firebase offers, we are not going about it the same way: our technological choices are quite different; everything we use is open source; and wherever possible, we use and support existing tools rather than developing from scratch.

Most notably, we use PostgreSQL rather than a NoSQL store. This choice was deliberate. We believe that no other database offers the functionality required to compete with Firebase, while maintaining the scalability required to go beyond it.

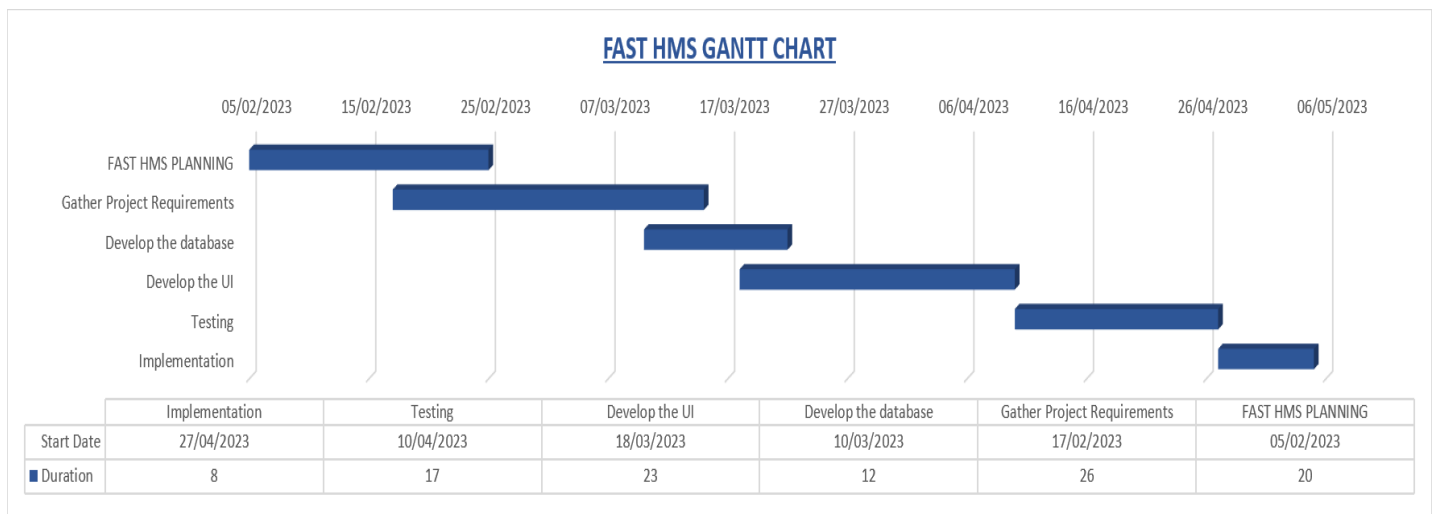
1.4 FLUTTER

Flutter – a simple and high performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system's canvas rather than through native framework.

Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

To be specific, Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming. Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive programming) compare the widget's state (old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

1.5 Gantt Chart



Chapter 2

REQUIREMENT SPECIFICATION

A computerized way of handling information about doctor, patient, ticket and appointment details is efficient, organized and time saving, compared to a manual way of doing so. This is done through database driven mobile application whose requirements are mentioned in this section.

2.1 OVERALL DESCRIPTION

A reliable and scalable database driven mobile application with security features that is easy to use and maintain is the requisite.

2.2 SPECIFIC REQUIREMENTS

The specific requirements of the Stock Market System are stated as follows:

2.2.1 SOFTWARE REQUIREMENTS

IDE – Visual Studio Code

Emulator – Android Studio

Android Version – 11.0

Dart – 2.18.4

2.2.2 HARDWARE REQUIREMENTS

Octa-core (4x2.0 GHz Kryo 260 Gold & 4x1.8 GHz Kryo 260

Silver) RAM – 4 GB or more

Screen Resolution – 720 x 1500

Hard disk – 3 GB or more

Touch Screen

2.3 FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

Functional Requirements

1. Admin should be able to register and remove doctors.
2. Admin should be able to check the pharmacy medicines stocks and add stocks if necessary.
3. Doctors should be able to login and view their appointment timings.
4. Doctors should be able to update their appointment timings according to their availability.
5. Doctors should be able to update their personal information.
6. Patients should be able to register and login to the system.
7. Patients should be able to view their updated hospital records.
8. Patients should be able to book appointments online.
9. Patients should be able to check the availability of doctors at a particular time and date.
10. Patients should be able to check if the pharmacy has medicine available.
11. Patients should be able to buy medicines from the pharmacy and generate a token number.
12. The system should be able to generate reports on doctor appointments, patient records and pharmacy stocks.
13. The system should be able to send notifications to patients about their appointments.
14. The system should be able to send notifications to doctors about changes in their appointment timings.
15. The system should be able to track the token numbers generated for pharmacy purchases.

Non-functional Requirements

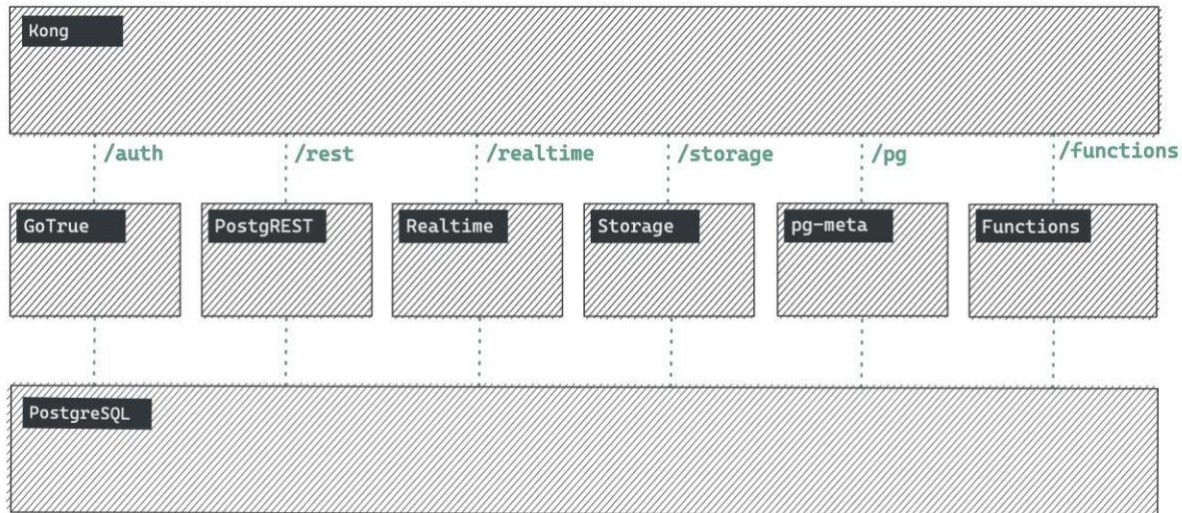
1. The system should be user-friendly and easy to use.
2. The system should be secure and protect patient data.
3. The system should be reliable and available 24/7.
4. The system should be scalable to accommodate future growth.
5. The system should be easy to install and maintain.
6. The system should be compatible with multiple devices and browsers.
7. The system should be fast and responsive to user inputs.
8. The system should be accessible to people with disabilities.
9. The system should have backup and recovery mechanisms in case of data loss.
10. The system should have a clear and concise user interface.
11. The system should be customizable to meet the specific needs of the hospital.
12. The system should have good performance under high load conditions.
13. The system should have adequate documentation for users and administrators.
14. The system should have good error handling and reporting mechanisms.
15. The system should comply with relevant laws and regulations, such as data protection laws.

Chapter 3

DETAILED DESIGN

3.1 SYSTEM DESIGN

Each Supabase project consists of several tools:



PostgreSQL (Database):

PostgreSQL is the core of Supabase. We do not abstract the PostgreSQL database — you can access it and use it with full privileges. We simply provide tools which makes PostgreSQL as easy to use as Firebase.

Studio (Dashboard):

An open source Dashboard for managing your database and services.

GoTrue (Auth):

A JWT-based API for managing users and issuing access tokens. This integrates with PostgreSQL's Row Level Security and the API servers.

PostgREST (API):

A standalone web server that turns your PostgreSQL database directly into a RESTful API. We use this with our `pg_graphql` extension to provide a GraphQL API.

Realtime (API & multiplayer):

A scalable websocket engine for managing user Presence, broadcasting messages, and streaming database changes.

Storage API (large file storage):

An S3-compatible object storage service that stores metadata in Postgres.

Deno (Edge Functions):

A modern runtime for JavaScript and TypeScript.

postgres-meta (Database management):

A RESTful API for managing your Postgres. Fetch tables, add roles, and run queries.

PgBouncer:

A lightweight connection pooler for PostgreSQL. This is useful for connecting to Postgres when using Server less functions.

Kong (API Gateway):

A cloud-native API gateway, built on top of Nginx.

3.2 ENTITY RELATIONSHIP DIAGRAM

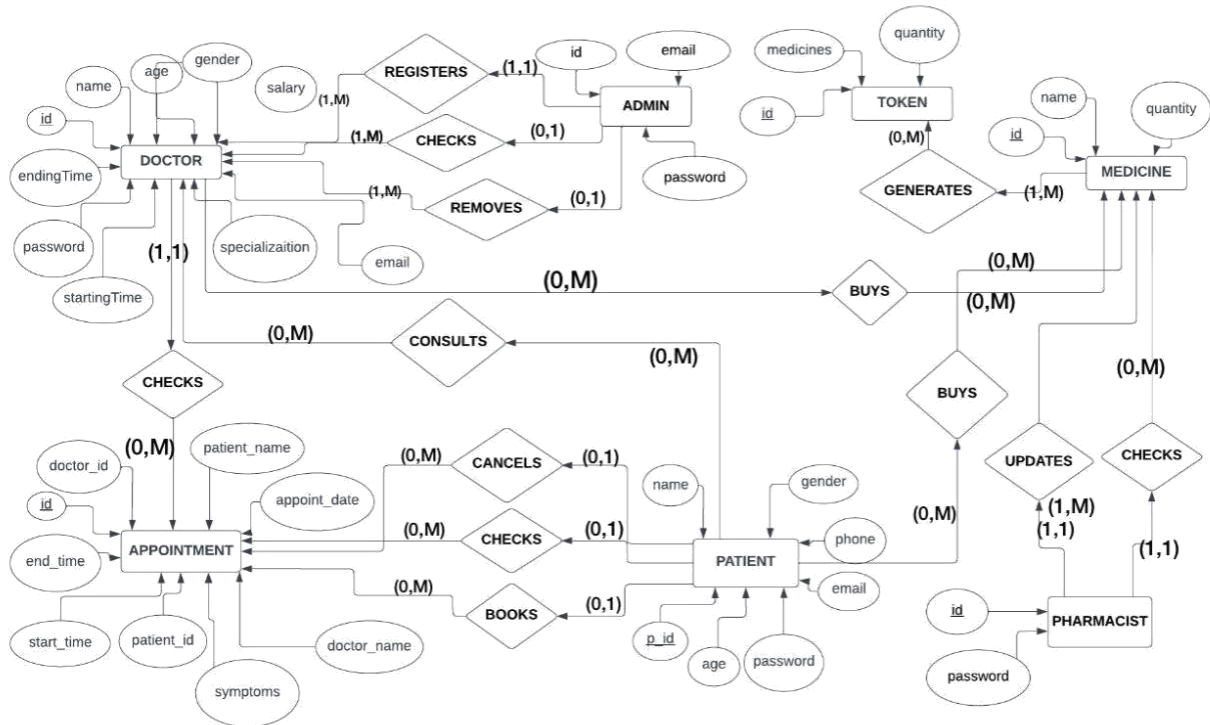
An entity–relationship model is usually the result of systematic analysis to define and describe what is important to processes in an area of a business.

An E-R model does not define the business processes; it only presents a business data schema in graphical form. It is usually drawn in a graphical form as boxes (entities) that are connected by lines (relationships) which express the associations and dependencies between entities.

Entities may be characterized not only by relationships, but also by additional properties(attributes), which include identifiers called "primary keys". Diagrams created to represent attributes as well as entities and relationships may be called entity-attribute-relationship diagrams, rather than entity-relationship models.

An ER model is typically implemented as a database. In a simple relational database implementation, each row of a table represents one instance of an entity type, and each field in a table represents an attribute type. In a relational database a relationship between entities is implemented by storing the primary key of one entity as a pointer or "foreign key" in the table of another entity.

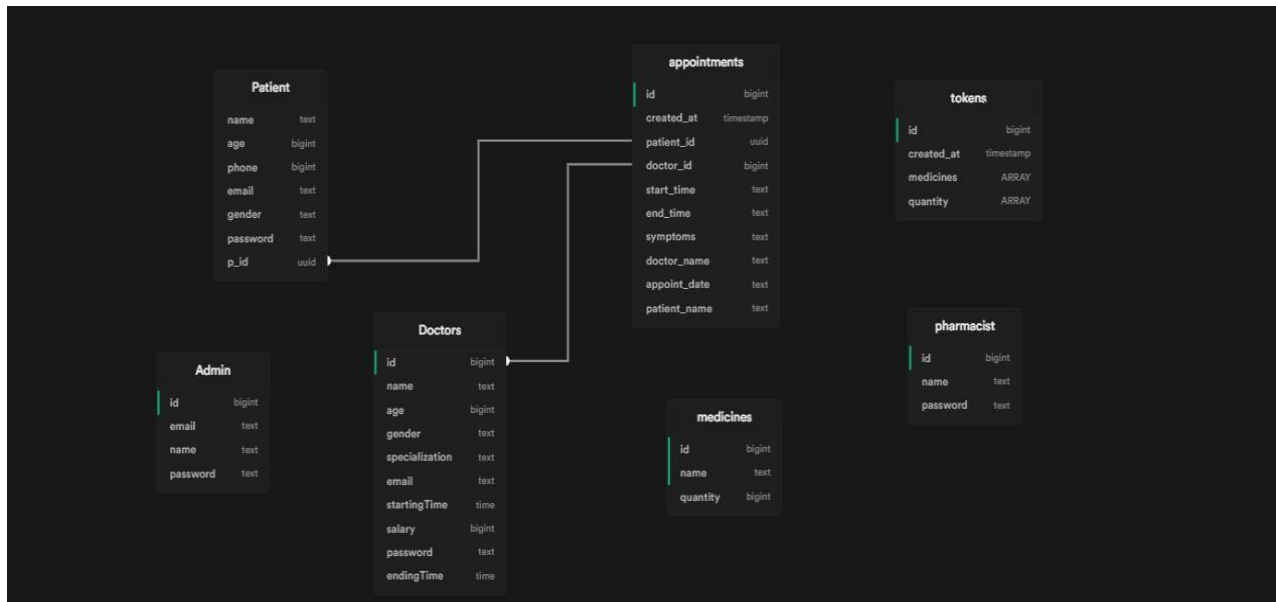
There is a tradition for ER/data models to be built at two or three levels of abstraction. Note that the conceptual-logical-physical hierarchy below is used in other kinds of specification, and is different from the three schema approach to software engineering. While useful for organizing data that can be represented by a relational structure, an entity -relationship diagram can't sufficiently represent semi-structured or unstructured data, and an ER Diagram's unlikely to be helpful on its own in integrating data into a pre-existing information system.



Cardinality notations define the attributes of the relationship between the entities. Cardinalities can denote that an entity is optional

3.3 RELATIONAL SCHEMA

The term "schema" refers to the organization of data as a blueprint of how the database is constructed. The formal definition of a database schema is a set of formulas called integrity constraints imposed on a database. A relational schema shows references among fields in the database. When a primary key is referenced in another table in the database, it is called a foreign key. This is denoted by an arrow with the head pointing at the referenced key attribute. A schema diagram helps organize values in the database. The following diagram shows the schema diagram for the database.



3.4 DESCRIPTION OF TABLES

The database consist of 7 tables.

1. Admin:
 - a. Id
 - b. Email
 - c. Name
 - d. Password
2. Pharmacist:
 - a. Id
 - b. Name
 - c. Password
3. Doctor:
 - a. Id
 - b. Name
 - c. Age
 - d. Gender
 - e. Specialization
 - f. Email
 - g. Starting time
 - h. Ending time
 - i. Salary
 - j. Password
4. Patient:
 - a. Name
 - b. Age
 - c. Phone
 - d. Email
 - e. Gender
 - f. Password
 - g. P_id

- 5. Appointments:
 - a. Id
 - b. Created at
 - c. Patient_id
 - d. Doctor_id
 - e. Start_time
 - f. End_time
 - g. Symptoms
 - h. Doctor_name
 - i. Appointment_date
 - j. Patient_name
- 6. Medicines:
 - a. Id
 - b. Name
 - c. Quantity
- 7. Tokens:
 - a. Id
 - b. Created_at
 - c. Medicines
 - d. Quantity

Chapter 4

IMPLEMENTATION

4.1 WORK BREAK DOWN STRUCTURE (WBS)

1. Project Initiation
 - 1.1 Define project objectives and scope
 - 1.2 Identify project stakeholders
 - 1.3 Create project proposal and obtain approval
2. Requirements Gathering and Analysis
 - 2.1 Identify user roles and their requirements (Admin, Doctor, Patient, Pharmacist)
 - 2.2 Document functional and non-functional requirements
 - 2.3 Perform system analysis and create use case diagram.
3. Design and Architecture
 - 3.1 Design the user interface for each user role (Admin, Doctor, Patient, Pharmacist)
 - 3.2 Define the database schema
 - 3.3 Determine the system architecture (client-server or cloud-based)
 - 3.4 Create wireframes and UI mockups
4. Front-End Development
 - 4.1 Set up the development environment (Flutter, Dart, Android Studio)
 - 4.2 Implement user registration and login functionality
 - 4.3 Develop screens for Admin, Doctor, Patient, and Pharmacist interfaces
 - 4.4 Implement appointment booking and cancellation features
 - 4.5 Create BMI calculator and diet suggestion functionality
 - 4.6 Implement the random user feature for purchasing medicines
5. Back-End Development
 - 5.1 Set up the backend server (SupaBase)
 - 5.2 Develop API endpoints for user authentication and authorization
 - 5.3 Implement API endpoints for data retrieval and manipulation (doctors, patients, appointments, medicines)
 - 5.4 Handle data validation and error handling
 - 5.5 Integrate the front-end with the back-end APIs
6. Testing and Quality Assurance
 - 6.1 Perform unit testing for individual components and functions
 - 6.2 Conduct integration testing to ensure proper communication between front-end and back-end
 - 6.3 Test different user scenarios (registration, login, appointment booking, etc.)
 - 6.4 Identify and fix bugs and issues
 - 6.5 Perform system testing to ensure the overall functionality and usability
7. Deployment and Maintenance
 - 7.1 Prepare the application for deployment (compile, package)
 - 7.2 Deploy the application on a suitable platform (web server or cloud platform)
 - 7.3 Set up database and server configurations
 - 7.4 Monitor and maintain the system for performance, security, and updates
 - 7.5 Provide user support and handle maintenance requests
8. Documentation and Training
 - 8.1 Create user manuals and system documentation
 - 8.2 Prepare technical documentation for developers and administrators
 - 8.3 Conduct training sessions for users (Admin, Doctor, Patient, Pharmacist)
 - 8.4 Provide ongoing support and documentation updates as needed

4.2 MODULES AND THEIR ROLES

4.2.1 DISPLAYING DOCTOR APPOINTMENTS

```
final messageStream = Supabase.instance.client
  .from('appointments')
  .stream(primaryKey: ['id'])
  .eq(['doctor_id', widget.curr_doc_data[0]['id']]);
```

4.2.2 REGISTER DOCTOR

```
void inputDoctorInfo() async {
  await Supabase.instance.client.from('Doctors').insert(
    {
      'name': s1,
      'age': i1,
      'specialization': s2,
      'email': s3,
      'salary': 10000,
      'gender': s4,
      'password': password1,
      'startingTime': '09:00:00',
      'endingTime' : '17:00:00',
    },
  );
}
```


4.2.3 REMOVING DOCTOR

```
var data = await Supabase.instance.client.from('Doctors').select('name,id').match({'name':name, 'id':id});
if(data.isEmpty){
  dialog_msg = 'No such record exist';
}else{
  //print(data.toString());
  await Supabase.instance.client.from('appointments').delete().match({'doctor_id':id});
  var res = await Supabase.instance.client.from('Doctors').delete().match(
    {'name':name, 'id':id});
  dialog_msg = 'Doctor record removed Successfully';
}
```

4.2.4 SHOWING DOCTOR DETAILS

```
var data = await Supabase.instance.client.from('Doctors').select('name,id').match({'name':name, 'id':docid});
if(data.isEmpty){
  print(data.toString());
  dialog_msg = 'No Record Found!';
}else{
  dialog_msg = 'Doctor Found!';
}
```

4.2.5 BOOKING PATIENT APPOINTMENT

```
if(sym == 'Heart Related'){
  doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization',const FetchOptions(
    count: CountOption.exact,
  )),match({'specialization':'Heart Specialist'}));
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Heart Specialis
}

else if(sym.toString() == 'Brain Related'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),match({'specialization':'Neurologist'}));
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Neurologist'});
}

else if(sym == 'Skin Related'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),match({'specialization':'Skin specialist'}));
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Skin specialist
}

else if(sym == 'Injured'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),match({'specialization':'Orthopedic'}));
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Orthopedic'});
}
```

```

else if(sym == 'Eye Related'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),).match({'specialization':'Eye specialist'});
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Eye specialist'});
}

else if(sym == 'Teeth Related'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),).match({'specialization':'Dentist'});
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'Dentist'});
}

else if(sym == 'Not Sure' || sym == 'Cough, Sneeze, Fever'){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),).match({'specialization':'General physician'});
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'General physician'});
}
print(suit_doc_data);

if(suit_doc_data.isEmpty){
  doc_data = await Supabase.instance.client.from('Doctors').select('specialization',const FetchOptions(
    count: CountOption.exact,
  )),).match({'specialization':'General physician'});
  suit_doc_data = await Supabase.instance.client.from('Doctors').select('name,id,specialization').match({'specialization':'General physician'});
}

```

4.2.6 CANCELLING APPOINTMENT

```

onPressed: () async {
  if (_formKey.currentState!.validate()) {
    late String dialog_msg;
    final List pat_app = await Supabase.instance.client.from('appointments').select('id').match(
      {'patient_id':widget.curr_pat_data[0]['p_id'], 'id': id});

    if(pat_app.isEmpty){
      dialog_msg = 'No appointment found with this appointment ID';
    }else{
      print(widget.curr_pat_data[0]['p_id']);
      dialog_msg = 'Appointment cancelled successfully!';
      await Supabase.instance.client.from('appointments').delete().match(
        {'patient_id':widget.curr_pat_data[0]['p_id'], 'id': id});
    }
  }
}

```

4.1.6 DISPLAYING PATIENT APPOINTMENTS

```

final messageStream = Supabase.instance.client
  .from('appointments')
  .stream(primaryKey: ['id'])
  .eq('patient_id', widget.curr_pat_data[0]['p_id']);

```

4.1.7 UPDATING DOCTOR DETAILS

```
if (_formKey.currentState!.validate()) {
  if(s1 == '' && s2== '' && s3== '' && password=='' && password1==''){
    dialog_msg = 'Atleast enter one field';
  }else{
    dialog_msg = 'Information updated successfully';
    if(s1!=''){
      await Supabase.instance.client.from('Doctors').update(
        {'name':s1,}).match({'id':widget.curr_doc_data[0]['id']});
    }
    if(s2!=''){
      await Supabase.instance.client.from('Doctors').update(
        {'specialization':s2, }).match({'id':widget.curr_doc_data[0]['id']});
    }
    if(s3!=''){
      await Supabase.instance.client.from('Doctors').update(
        {'email':s3,}).match({'id':widget.curr_doc_data[0]['id']});
    }
    if(password1!=''){
      await Supabase.instance.client.from('Doctors').update(
        {'password':password1,}).match({'id':widget.curr_doc_data[0]['id']});
    }
  }
}
```

4.1.8 UPDATE MEDICINES IN STOCK

```
if(norvasac!=0){
  final q1 = await Supabase.instance.client
    .from('medicines')
    .select('name,quantity')
    .match({'name': 'norvasac'});

  int quan1 = q1[0]['quantity'];
  quan1 = quan1 + (norvasac);

  if(quan1 < 0){
    outOfStock.add(q1[0]['name']);
  }else{
    final data = await Supabase.instance.client
      .from('medicines')
      .update({'quantity': quan1 })
      .match({'name':'norvasac'});
  }
}
```

4.1.9 SHOW MEDICINE IN STOCK

```
Widget build(BuildContext context) {  
  final medStream = Supabase.instance.client.from('medicines').stream(primaryKey: ['id']);
```

4.3 RESULT

- ✓
- ✓ Authenticate Admin, Patient and Pharmacist
- ✓ Admin login and Pharmacist login
 - Admin
 - Registers Doctor
 - Removes Doctor
 - Check Doctor details
 - Doctor
 - Update his/her details
 - Check his Appointments
 - Patient
 - Registers himself
 - Login
 - Book Appointment
 - Cancel Appointment
 - BMI calculator
 - Diet Suggestion
 - Random User (Without Login/Registration)
 - Buy medicines
 - Pharmacist
 - Login
 - Update stock

Chapter 5

TESTING

5.1 SOFTWARE TESTING

Testing is the process used to help identify correctness, completeness, security and quality of developed software. This includes executing a program with the intent of finding errors. It is important to distinguish between faults and failures. Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors. It can be conducted as soon as executable software (even if partially complete) exists. Most testing occurs after system requirements have been defined and then implemented in testable programs.

5.2 MODULE TESTING AND INTEGRATION

Module testing is a process of testing the individual subprograms, subroutines, classes, or procedures in a program. Instead of testing whole software program at once, module testing recommend testing the smaller building blocks of the program. It is largely white box oriented. The objective of doing Module testing is not to demonstrate proper functioning of the module but to demonstrate the presence of an error in the module. Module testing allows implementing of parallelism into the testing process by giving the opportunity to test multiple modules simultaneously. The final integrated system too has been tested for various test cases such as duplicate entries and type mismatch

5.3 TEST CASES

1. Test Case: Admin Registration Description: Verify that the admin can successfully register with valid credentials. Steps:
 1. Open the application.
 2. Click on the Admin Registration button.
 3. Enter valid admin details.
 4. Click on the Register button.Expected Result: Admin registration is successful.
2. Test Case: Doctor Registration Description: Ensure that a doctor can register successfully with valid information. Steps:
 1. Open the application.
 2. Click on the Doctor Registration button.
 3. Enter valid doctor details.
 4. Click on the Register button.Expected Result: Doctor Registration is successful.
3. Test Case: Patient Registration Description: Verify that a patient can register successfully with valid information. Steps:
 1. Open the application.
 2. Click on the Patient Registration button.
 3. Enter valid patient details.
 4. Click on the Register button.Expected Result: Patient registration is successful.

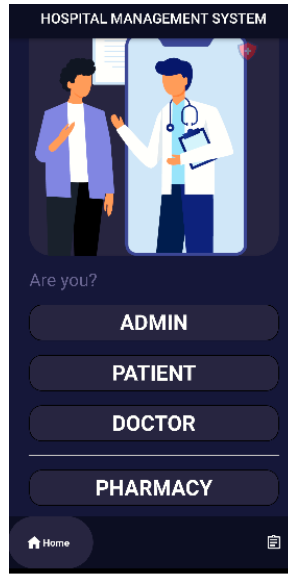
4. Test Case: Admin Login Description: Check if the admin can log in with valid credentials. Steps:
 1. Open the application.
 2. Enter valid admin username and password.
 3. Click on the Login button.Expected Result: Admin login is successful.
5. Test Case: Doctor Login Description: Verify that a doctor can log in with valid credentials. Steps:
 1. Open the application.
 2. Enter valid doctor username and password.
 3. Click on the Login button.Expected Result: Doctor login is successful.
6. Test Case: Patient Login Description: Ensure that a patient can log in with valid credentials. Steps:
 1. Open the application.
 2. Enter valid patient username and password.
 3. Click on the Login button.Expected Result: Patient login is successful.
7. Test Case: Book Appointment Description: Test the ability to book an appointment as a patient. Steps:
 1. Log in as a patient.
 2. Click on the Book Appointment button.
 3. Select a doctor and choose a suitable date and time.
 4. Click on the Book button.Expected Result: Appointment booking is successful.
8. Test Case: Cancel Appointment Description: Verify that a patient can cancel an appointment. Steps:
 1. Log in as a patient.
 2. Go to the My Appointments section.
 3. Select the appointment to be cancel.
 4. Click on the Cancel Appointment button.Expected Result: Appointment cancellation is successful.
9. Test Case: Update Doctor Details Description: Check if a doctor can update their personal information. Steps:
 1. Log in as a doctor.
 2. Go to the My Profile section.
 3. Update the desired fields.
 4. Click on the Update button.Expected Result: Doctor details are successfully updated.
10. Test Case: Update Stock Description: Ensure that the pharmacist can update the medicine stock. Steps:
 1. Log in as a pharmacist.
 2. Go to the Stock Management section.
 3. Add or subtract the quantity of a medicine.
 4. Click on the Update Stock button.Expected Result: Medicine stock is successfully updated.

Chapter 6

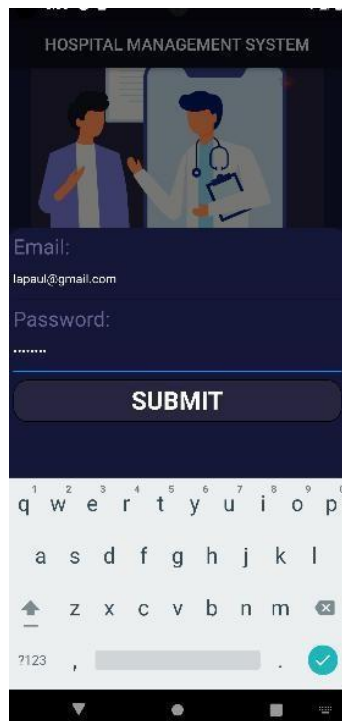
SNAPSHOTS

This chapter consists of working screenshots of the project.

Front Page:

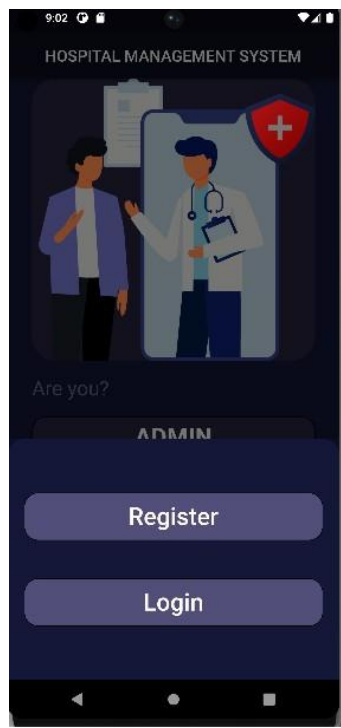


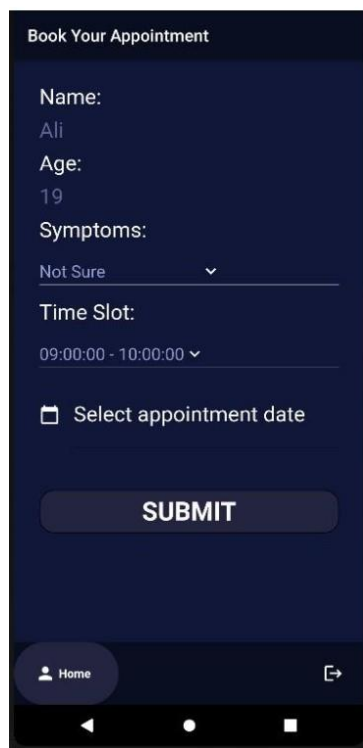
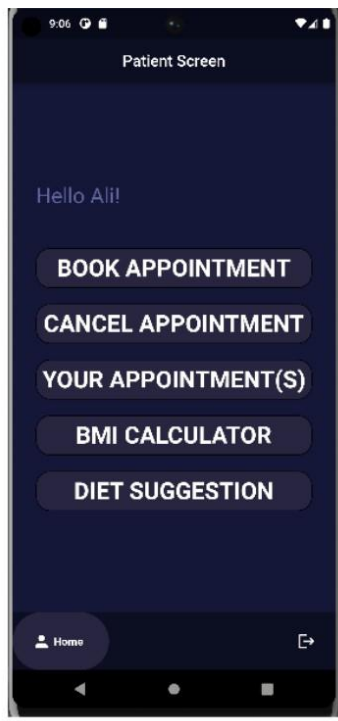
Admin:

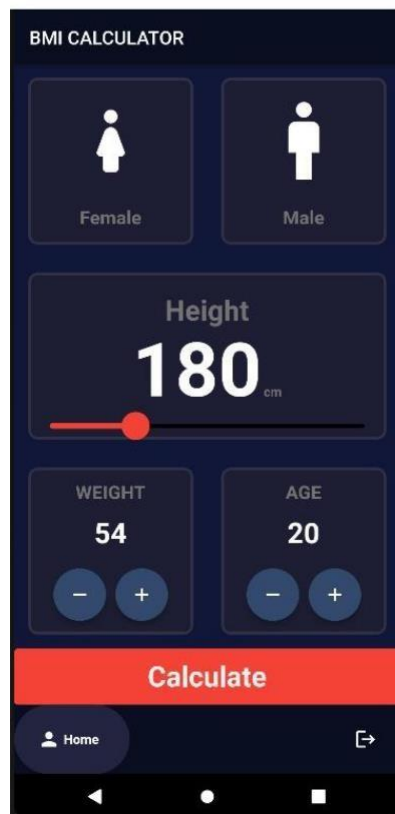




Patient

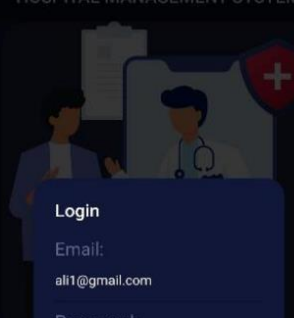






Doctor:

HOSPITAL MANAGEMENT SYSTEM



Login

Email:
ali1@gmail.com

Password:

SUBMIT

Register

Login

Update Your Details

Name :

Specialization :

Email :

New Password:

Confirm Password :

Note: Please reload the application so that changes could take effect.

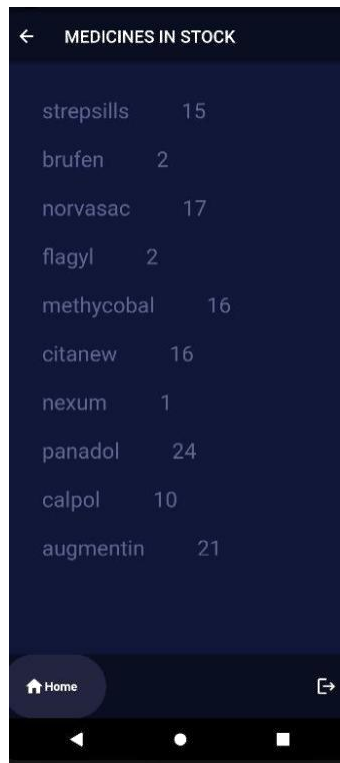
SUBMIT

Home

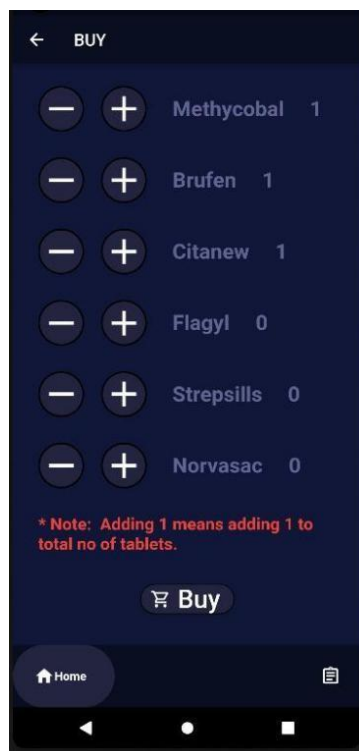


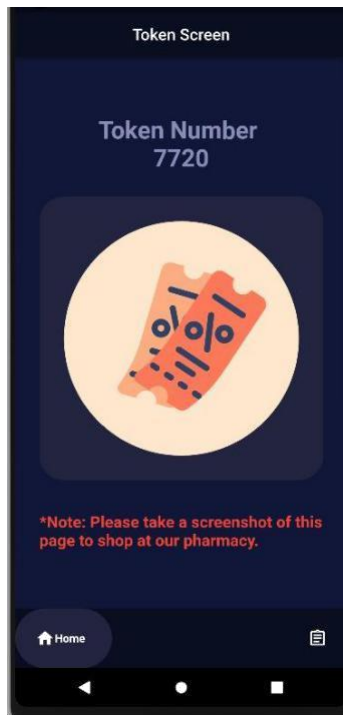
Pharmacist:





Random User:





About Us:



Chapter 7

CONCLUSION

The Hospital Management System provides easier maintenance of doctor registration and stock maintenance. It allows simplified operation and is a time saving platform with the ability to view and update data and easily book and view appointments and buy medicines. The application has been completed successfully and tested with suitable test cases. It is user friendly and contains suitable options for admin, doctors and patients. This is developed using Flutter (DART) and SupaBase. The goals achieved by this project are:

- Centralized database
- Easier buying medicines and booked appointments
- User friendly environment.
- Efficient management of stocks.
- Ability to view data and also update it

Chapter 8

FUTURE ENHACEMENTS

Future upgrades to this project will implement:

Give notification of appointments to user when is appointment date time is near. Buy medicine online checkout payment feature can be added

Online consultation through video chat. Adding more medicine stock.

REFERENCES

Supabase Documentation: <https://supabase.com/docs/reference/dart/upgrade-guide>

Dart Documentation: <https://dart.dev/>

Flutter Full Course Angela Yu: <https://mega.nz/folder/3WhSkBjJ#YSDbnDegckd9-xSQo4WoqA/folder/HXxyyDiQ>

Flutter Documentation: <https://docs.flutter.dev/>