



LAU

School of Arts and Sciences

Department of Computer Science & Mathematics

APPLICATION DESIGN FOR A MEDICAL CLINIC



المركز الطبي للجامعة اللبنانية الأميركية - مستشفى رزق
Lebanese American University Medical Center-Rizk Hospital

By

Adam Kanj (202105510)

Ahmad Malak (202105744)

Lionelli Maroun (202102103)

REPORT

Submitted to DR. IBRAHIM EL BITAR

Course “CSC490: Software Engineering” in Computer Science

Phase III

May 1, 2023

Table of Contents:

1. Introduction	3
2. Glossary	4
1- Acronyms	4
2- Definitions	4
3. Background	5
4. User Requirements	6
General Requirements	6
1- USER MANAGEMENT	6
2- PATIENT MANAGEMENT	7
3- BILLING AND CLAIMS MANAGEMENT	7
4- ADMINISTRATIVE TASKS	8
5- FILE SUPPORT	8
6- USER FEEDBACK	9
7- NOTIFICATION SYSTEM	9
8- PRESCRIPTION MANAGEMENT	10
5. System Requirements	10

1- LOGIN AND AUTHENTICATION	11
2- SEARCH AND FILTER	11
3- PATIENT HISTORY	12
4- RATINGS AND REVIEWS	13
5- NOTIFICATIONS	14
6- STAFF AND INVENTORY MANAGEMENT.....	15
6. Software Architecture Design.....	19
7. Software Engineering Tools and Diagrams	20
8. Implementation	26
9. Conclusion	49

1. Introduction

Lebanese American University Medical Center (LAUMC) is the new Clinique-centered software developed as a Web App. This project aims to maximize the efficiency of the clinic by facilitating numerous tasks and organizing the procedure of communicating between patients and staff. This will be done by providing structured guidance through a user-friendly and accessible user interface.

The software focuses on five main objectives:

1. **Streamline Patient Management:** One of the primary objectives of clinic software is to streamline patient management. This includes patient intake, appointment scheduling, electronic medical record (EMR) management, and billing.
2. **Improve Patient Care:** Clinic software can help improve patient care by providing physicians with quick and easy access to patient information, including medical history, lab results, and medication information.
3. **Increase Efficiency:** Clinic software can increase the efficiency of clinic operations by automating routine tasks such as appointment reminders, billing, and claims management.
4. **Ensure Compliance:** Clinic software can help ensure compliance with regulatory requirements such as HIPAA and other privacy laws by securely managing patient information and maintaining proper documentation.
5. **Provide Data Analytics:** Clinic software can provide valuable data analytics to help the clinic better understand their patient population and identify opportunities for improvement. This can include tracking patient outcomes, identifying trends, and identifying areas for cost savings.
6. **Customization and Integration:** Clinic software should allow for customization and integration with other systems. This includes customizing the user interface to fit the needs of the clinic, as well as integrating with other software systems such as pharmacy software, medical imaging systems, and laboratory information systems.
7. **Mobile Compatibility:** With the increasing use of mobile devices, clinic software should be compatible with mobile devices. This includes having a responsive design that adapts to different screen sizes and allowing physicians and staff to access patient information and manage clinic operations from their mobile devices.

2. Glossary

a. Acronyms:

1. Clinic software: A software application that helps manage patient records, appointment scheduling, and billing system for a healthcare facility.
2. Patient record: A digital or paper-based documentation of a patient's medical history, including personal details, diagnosis, and treatments.
3. Appointment scheduling: The process of scheduling appointments for patients to see healthcare providers.
4. Billing system: A software application that manages the billing and invoicing process of healthcare services provided to patients.
5. Login ID: A user identification number to enter the system.
6. Database: Collection of information in a structured form
7. EMR: Electronic Medical Record
8. HIPAA: Health Insurance Portability and Accountability Act

b. Definitions:

- User Requirements: Statements defining the set of functionalities and constraints the system will ensure.
- System Requirements: More detailed statements that describe how the system will operate.
- Domain Requirements: Requirements related to the domain of the software. In this case, requirements related to education and learning.
- Functional Requirements: Requirements related to the functionalities of the system. They state generally what the system should or should not do.
- Non-functional Requirements: Requirements that are not related to a specific behavior or functionality but rather describe the system as a whole. They usually relate to system performance, security, accessibility, interactions with other software, etc.

3. Background

LAUMC is a medical clinic application designed with the aim of streamlining healthcare operations and improving patient care. It is built with the needs of various stakeholders in mind, including medical professionals, clinic administrators, and patients. The LAUMC application offers features that cater specifically to the unique requirements of each stakeholder group, making it a comprehensive solution for healthcare management.

For medical professionals, LAUMC provides tools for managing appointments, patient records, and communication with patients. The application allows doctors and nurses to easily access patient data and medical histories, helping to improve the accuracy of diagnoses and treatment plans. LAUMC also enables medical professionals to communicate with patients remotely, allowing for more frequent check-ins and follow-up appointments.

Clinic administrators can benefit from LAUMC's administrative features, which include appointment scheduling, billing, and inventory management. The application provides real-time data on inventory levels, allowing for more efficient supply management and cost savings. Additionally, LAUMC includes a billing system that integrates with insurance providers, reducing the time and resources required for billing and claims management.

Finally, LAUMC offers features for patients, making it easier for them to access medical care and stay informed about their health. The application allows patients to schedule appointments, access medical records, and receive reminders for upcoming appointments. LAUMC also provides patients with a platform to communicate with their healthcare providers and receive remote medical care when necessary.

In developing LAUMC, the team conducted extensive research on the healthcare industry, including interviews with medical professionals and patients. This feedback was used to identify the pain points and challenges faced by stakeholders in the industry, and to develop features that address these issues. LAUMC aims to provide a comprehensive solution for healthcare management, improving the efficiency of healthcare operations and enhancing the patient experience.

4. User Requirements

Following are the user requirements of the software:

General Requirements:

- The user shall log in to the software using his/her account constituting of a login-ID and password.
- The software will allow patients and physicians to access medical records stored in the database and schedule appointments.

1. User Management:

- Patients should be able to create a profile and register using their email address and personal information.
- The system shall require patients to log in with their credentials to access their profiles and make appointments.
- Doctors should be able to create a profile and register using their email address and personal information.
- The system shall require doctors to log in with their credentials to access their profiles and appointments.

2. Patient Management:

- The software should allow healthcare providers to manage patient records.
- The software should allow healthcare providers to create new records.
- The software should allow healthcare providers to update existing records.
- The software should allow healthcare providers to access patient records.
- The system shall allow healthcare providers to manage patient medical records by recording patient symptoms, diagnosis, treatment plans, and medications prescribed.

2.1. Electronic Medical Record (EMR):

- The software shall include a robust EMR system that allows physicians and staff to easily access and manage patient information.
- The EMR shall include fields for demographic information, medical history, medications, allergies, immunizations, and other relevant patient data.

- The EMR shall be customizable, allowing clinics to add custom fields and data points as needed.
- The EMR shall include an audit trail that tracks all changes to patient records.

2.2 Appointment Scheduling

- The software shall allow for easy appointment scheduling from the EMR.
- The software shall include the ability to view availability of appointments, emergency rooms, hospital beds, surgery rooms, vaccination centers, and locations of mobile clinics.
- The appointment scheduler shall allow for recurring appointments, block scheduling, and other advanced scheduling options.
- Patients should be able to cancel or reschedule appointments.
- Patients should receive reminders for upcoming appointments.
- Doctors should be able to view their schedules and upcoming appointments.
- Doctors should be able to reschedule or cancel appointments.
- The appointment scheduler shall be customizable, allowing clinics to tailor appointment types and duration to their specific needs.

2.3 Patient Communication

- The software shall include a patient portal that allows patients to view their medical records, schedule appointments, and communicate with clinic staff.
- The patient portal shall be accessible from both desktop and mobile devices.
- The software shall include automated patient communication features, such as appointment reminders, prescription refill reminders, and test result notifications.
- The software shall allow for secure messaging between patients and clinic staff.

3. Billing and Claims Management

3.1. Claims Management

- The software shall include billing and claims management functionality, allowing for easy submission and tracking of claims.
- The claims management system shall be customizable, allowing clinics to tailor billing codes and claim submission processes to their specific needs.
- The software shall include real-time claim adjudication, allowing clinics to quickly identify and resolve claim errors.

3.2. Payment Processing

- The software shall include payment processing functionality, allowing clinics to process payments from patients and insurance providers.
- The payment processing system shall be secure and compliant with industry standards, such as PCI DSS.
- The software allow for multiple payment options, such as credit card, debit card, and ACH transfers.

4. Administrative Tasks

4.1. Staff Management

- The software shall include staff management functionality, allowing clinics to manage employee schedules, time off requests, and other personnel information.
- The staff management system shall be customizable, allowing clinics to tailor employee roles and permissions to their specific needs.
- The software shall allow for automated payroll processing and reporting.

4.2. Inventory Management

- The software shall include inventory management functionality, allowing clinics to manage medical supplies, equipment, and other inventory items.
- The inventory management system shall include automatic reorder alerts and real-time inventory tracking.
- The software shall be customizable, allowing clinics to add custom inventory categories and fields as needed.

5. File Support

- The software shall support Microsoft files (excel, word, doc), PDF files such as insurance coverages, image files (JPEG/PNG), HL7 files which are used to exchange healthcare info, and DICOM file format which are used to transmit and process medical images and scans.
- The maximum document size allowed is 50 mb.
- Medicine could be prescribed for patients using URL's which take them to the list of medicine found in the database.

6. User Feedback

- Patients should be able to rate and provide feedback on doctors and services.
- Patients should be able to report any issues or complaints to the clinic.
- Doctors should be able to view patient ratings and feedback on their services.
- Doctors should be able to respond to patient feedback and address any issues or complaints.

7. Notification System

- The system should send notifications to patients and doctors about upcoming appointments, changes in schedules, and new messages.
- Patients should be able to choose their preferred method of receiving notifications, such as email, text message, or push notification.

8. Prescription Management

- Doctors should be able to prescribe medications to patients online.
- Patients should be able to view their prescribed medications and request refills.
- The system should alert doctors when patients request medication refills.

5. System Requirements

Functional Requirements:

Functional requirements refer to the specific features and functions that a software or system must possess in order to meet the needs of its users or stakeholders. These requirements describe what the system should do, how it should behave under different conditions, and what outputs it should produce in response to specific inputs. Functional requirements typically focus on the system's capabilities, performance, security, and usability, and are used to guide the design, development, testing, and implementation of the software or system.

1. Login and Authentication:

- The app shall prompt any user to sign up using their phone number. *
- Users must authenticate themselves using their unique username and password.
- The app shall have a “User Profile” tab that enables users to manage their profiles.
- The app shall prompt users to upload profile pictures matching their IDs. *
- The app should provide doctors with an option to update their room location.
- The app should provide users with an option to delete their profile permanently.
- The app shall require the user to update their verification document when its expiration date is due. *
 - The user will be unable to use the app until the document is updated.
 - Verification documents that differ from the previously uploaded document in name or age will not be accepted.
- Users must have the ability to access their accounts by entering their username, phone number, or email address in addition to providing their password.

2. Search and Filter:

- Users shall be able to search for the doctors using their names.
- The “Filter” button shall open a small popup window to change the order and type of content displayed.
- The app should provide filter option allows the users to view the doctors available based on specialties of the doctors.

- The application must have a feature that allows patients to select their desired gender for the doctor, and only show doctors who meet their preference.
- The app should provide a filter option that displays doctors based on their rating and reviews.
- Patients shall be able to choose doctors based on higher experience by researching the doctor's credentials and work history.
- The application shall allow patients to view doctor's credentials and work history.
- Once the user clicks on one of the results, the application shall open the corresponding file.

3. Patient history:

The system should provide a secure and reliable storage mechanism for storing:

- personal information including name, date of birth, gender, contact information, emergency contact, occupation, and insurance details.
- medical history including information about past and current medical conditions, allergies, surgeries, hospitalizations, medications, and treatments.
- family history including information about the patient's family medical history, including any genetic conditions or hereditary diseases.
- immunization history including a record of immunizations the patient has received, including the dates and types of vaccines.
- lab results including results from lab tests, such as blood work, urine analysis, and imaging tests like X-rays and MRIs.
- vital signs including a record of the patient's vital signs, such as blood pressure, temperature, heart rate, and respiratory rate.
- visit history including a record of the patient's past visits to the medical center, including dates, reasons for the visit, and any treatments or medications prescribed.
- patient notes including any notes made by healthcare providers during a patient's visit, including symptoms, diagnosis, treatment plans, and follow-up instructions.
- patient preferences should including information about the patient's preferences, such as preferred language, cultural or religious considerations, and communication preferences.

4. Ratings and Reviews:

- A ratings and reviews feature will be available for both patients and doctors in the system.
- Access to the ratings and reviews feature will be provided in the app's main menu for both patients and doctors.
- Patients and doctors have the option to rate each other using a five-star scale after a specified period based on their interactions, such as appointments, surgeries, and other activities.
- The system will securely store the ratings and reviews in a database for future reference and analysis.
- An average rating will be provided for each patient and doctor based on the received ratings.
- The average rating will be calculated by summing up all received ratings and dividing them by the total number of ratings received.
- Patients have the option to check ratings and reviews of doctors before scheduling for an appointment.
- An automatic ban feature will be implemented to prohibit users with ratings below two from using the app.

5. Notifications:

- The application allows patients to schedule appointments with doctors and receive notifications about upcoming appointments both one day and one hour in advance.
- The application sends notifications to patients to remind them of upcoming appointments, reducing the likelihood of missed appointments.
- Patients can view their appointment history and cancel or reschedule appointments through the application.
- Doctors can view their appointment schedule through the application and receive notifications about new appointments or changes to their schedule.
- In addition to appointment notifications, the application can send notifications to patients and doctors about other activities, such as test results or prescription renewals.

- The application can be customized to send notifications in different formats, such as push notifications, emails or SMS messages, depending on the preference of the patient or doctor.
- Patients and doctors can communicate through the application, enabling them to discuss any questions or concerns about appointments or other medical issues.
- The application can be integrated with the medical center's electronic health record system, allowing for seamless transfer of patient information and appointment data.
- The patients should be able to receive reminders through the application to take medications at the appropriate times.

6. Staff and Inventory Management:

- After the appointment between the doctor and patients, he could:
 - Prescribe medication to the patient through the application, which can be picked up at the medical center's pharmacy.
 - Order diagnostic tests such as x-rays, MRIs, or blood tests through the application.
 - Send referrals to other specialists or healthcare providers through the application.
 - Schedule follow-up appointments or consultations with the patient through the application.
 - Send secure messages to the patient through the application to provide medical advice or answer any questions they may have.
- For Medicine:
 - Staff should be able to view a list of medications available at the medical center, including their names, dosages, and instructions for use.
 - Pharmacists and doctors should be able to search for specific medications by name, condition, or category (such as antibiotics, pain relievers, or antihistamines).
 - Pharmacists should check whether a particular medication is in stock at the medical center's pharmacy.
 - The application should enable patients to order medications that have been prescribed by their doctor for pickup.
- For Machines:
 - Users should be able to view a list of medical equipment and machines available at the medical center, such as x-ray machines, MRI scanners, or EKG machines.

- Staff should be available to check whether a particular machine is currently available or in use.
- Doctors can schedule appointments to use specific machines through the application.
- Staff should be able to send requests for maintenance or repairs of machines through the application.

7. Payment Management:

- **Payment Options:** The software should offer multiple payment options to the user, such as credit card, debit card, PayPal, and bank transfer.
- **Payment Gateway Integration:** The software must integrate with payment gateway APIs such as Stripe, PayPal, and Braintree to facilitate payment processing.
- **Invoicing:** The software should generate invoices for every transaction, including patient information, payment details, and the items purchased.
- **Billing Statements:** The software should generate billing statements for each patient, including itemized charges, payment details, and any outstanding balances.
- **Insurance Management:** The software should be able to manage insurance information for each patient, including policy details, claim submissions, and insurance payments.
- **Refund Management:** The software should allow for refunds to be processed for cancelled appointments, incorrect payments, or other billing errors.
- **Payment Reminders:** The software should be able to send payment reminders to patients with outstanding balances, including the amount owed and payment due date.
- **Payment Tracking:** The software should track all payments received, including payment method, amount, and date, for accurate financial reporting.
- **Payment Reporting:** The software should be able to generate payment reports, including total revenue, outstanding balances, and payment breakdowns by payment method, patient, or date range.
- **Integration with Electronic Health Records:** The software should integrate with electronic health records (EHRs) to ensure that billing information is accurate and up-to-date.
- **Integration with Accounting Software:** The software should integrate with accounting software such as QuickBooks or Xero for efficient bookkeeping and financial reporting.

- **Compliance with Regulations:** The software should comply with all relevant regulations, such as HIPAA, GDPR, and PCI DSS, to ensure that patient payment and billing information is secure and protected.

Non-Functional Requirements:

Non-functional requirements, also known as quality requirements, are the characteristics that describe how a software system should behave, rather than what it should do. They are related to the system's performance, reliability, scalability, security, maintainability, and usability, and are often critical to the success of the system. Non-functional requirements are often qualitative in nature, and are typically specified as constraints or criteria that the system must meet, rather than specific features or functionality.

1. Usability:

- The software system should have an intuitive and user-friendly interface, making it easy for users to navigate and find what they need.
- The software system should be responsive and accessible from a range of devices and screen sizes, to accommodate different user preferences and needs.
- The software system should be designed to minimize user errors and ensure efficient workflows.

2. Security:

- The software system should be designed with robust security measures to protect patient data and comply with HIPAA regulations and other privacy laws.
- The software system should include features for user authentication and authorization, to prevent unauthorized access and ensure that data is only accessible to authorized personnel.
- The software system should include a backup and recovery system to ensure that data is not lost in the event of a system failure or other disaster.

3. Performance:

- The software system should be designed to perform well, with fast load times and minimal lag, even when handling large amounts of data.
- The software system should be scalable to accommodate the growing needs of a clinic, including the ability to handle an increasing number of patients and users.
- The software system should be reliable and available for use 24/7, with minimal downtime for maintenance and upgrades.

4. Integration:

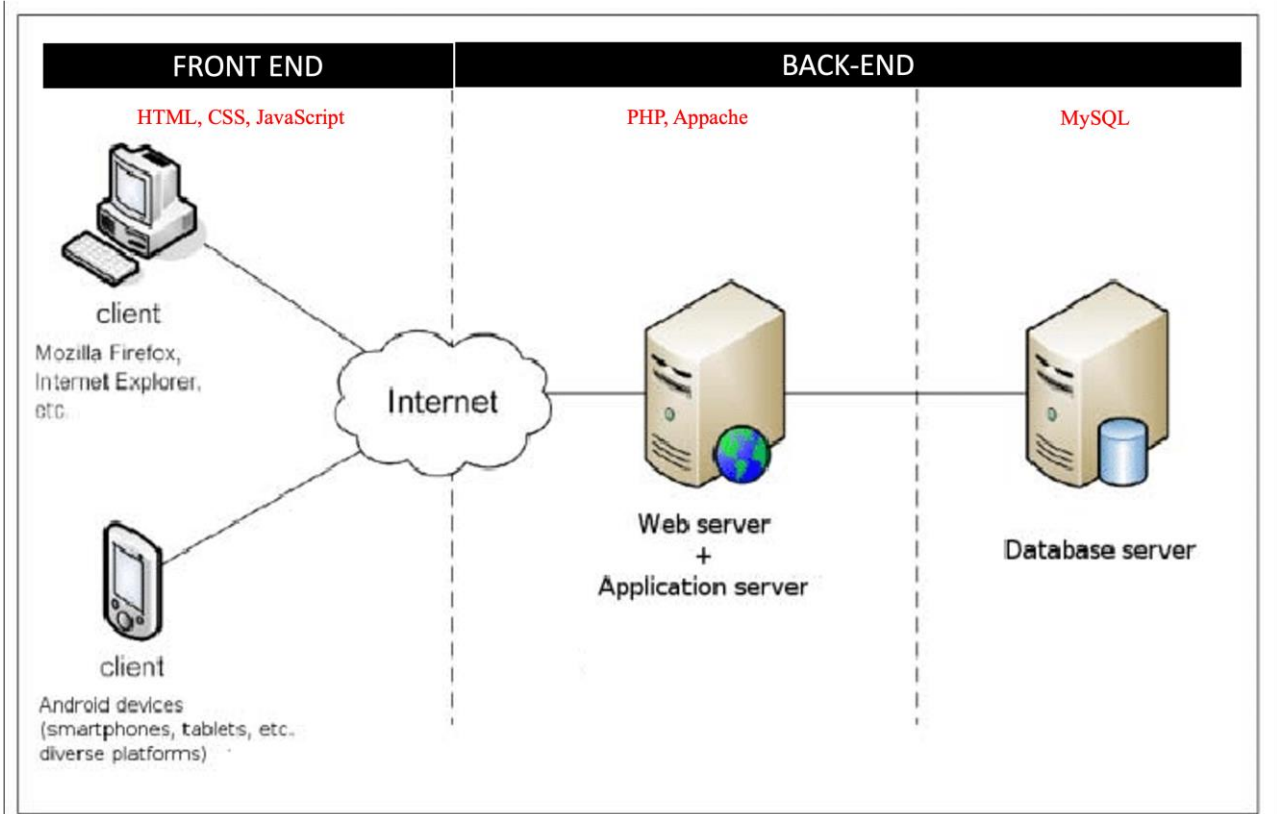
- The software system should be designed to integrate with other healthcare systems and software, such as electronic prescribing systems, lab information systems, and imaging systems.
- The software system should be designed to import and export data in a standard format, to facilitate interoperability with other systems and improve data sharing.

5. Support and Maintenance:

- The software system should include comprehensive user documentation and training resources, to ensure that users are able to make full use of the software's features and capabilities.
- The software system should include ongoing technical support and maintenance, to address any issues or bugs that arise and ensure that the software remains up-to-date and functional.
- The software system should be designed with flexibility and adaptability, to accommodate changing user needs and evolving healthcare regulations and standards.

6. Software Architecture Design

Software architecture refers to the structures of a software system and the discipline of creating such structures and systems. Our platform is a web-based application, therefore the software architecture is defined as follows:



The program is split into two parts:

- **Front-end:** The presentation and visuals of the software. In other words, what the user sees. These are done using HTML, CSS and JavaScript as programming languages.

- **Back-end:** Involves all servers and databases that the software utilizes. Here, we're mainly using PHP and SQL.

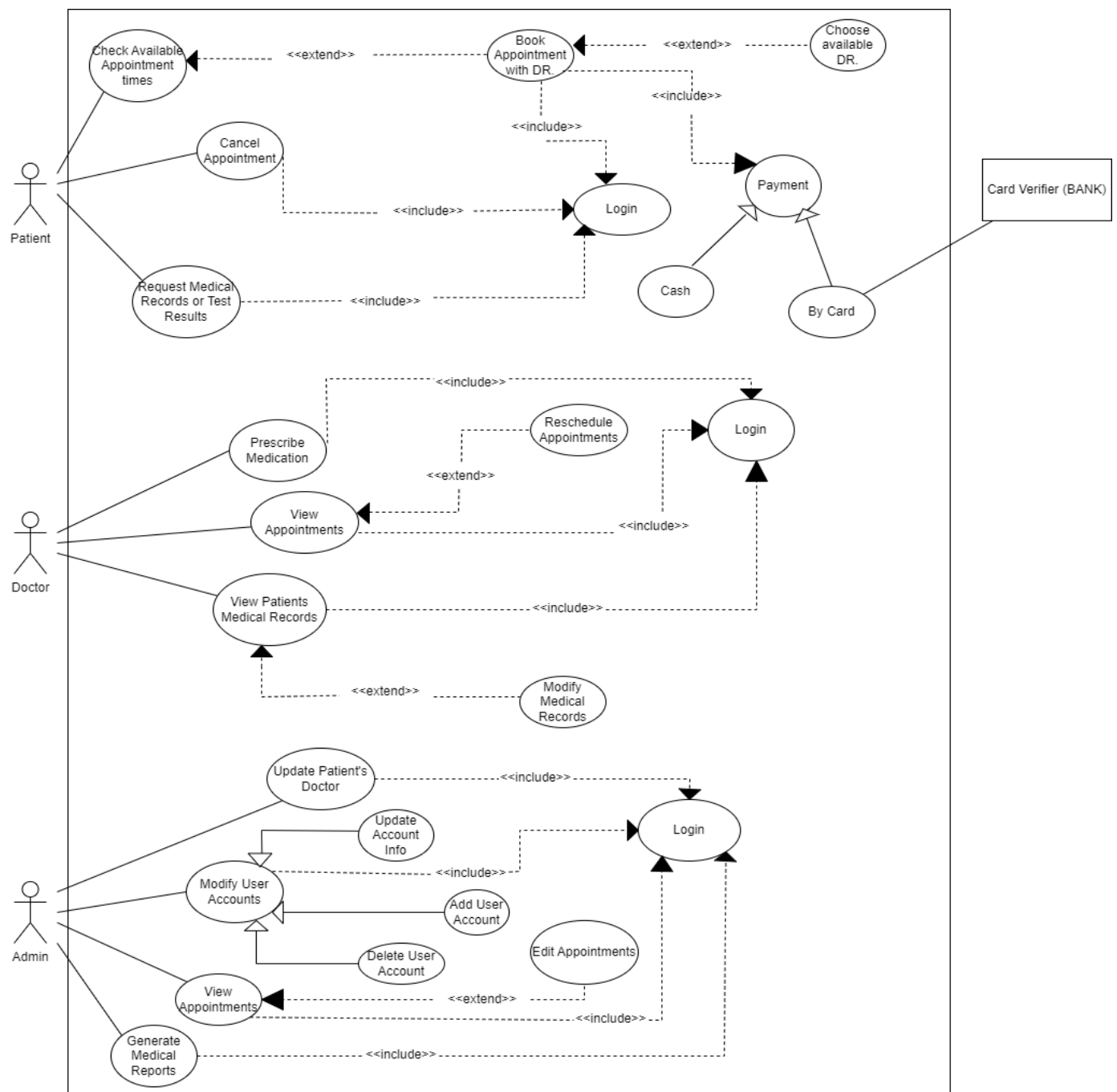
Since our software is web-based, we decided to go with the traditional 3-tier software architecture which involves the clients, server, and database. The clients can be from any device. The server is a midway communication between the clients and the database.

7. Software Engineering Tools and Diagrams

This section will discuss the different diagrams that the team used in order to produce the software. Each diagram was designed for a specific purpose, which will be mentioned in the document.

7.1 Use Case Diagram

The first diagram to be presented is the use case diagram, which shows the user's possible interactions (Called use cases) with a system. The figure below shows the use case diagram for the waterfall booking system.



In the use-case diagram previously shown, we include 3 primary actors: the Patient, the Admin, and the Doctor. These are the types of users that will access and use our system. As a secondary actor, we have the card verifier (BANK), which will be discussed later.

The Patient has one of three options to do in our system, either to check available appointment times, to cancel booked appointments, or request medical reports/test results. In order to cancel already booked appointments or request a medical report/test result the patient is asked to log-in to our system, however, any user, even if not logged-in, can check the available appointment times at the clinic. If a customer chooses to book an appointment, he/she is asked to log-in to the system, and may choose the available Doctor at that time, then they are directed to either pay by card or choose to pay by cash. If the user chooses to pay by card, the card information and payment is verified and checked by the bank.

Another actor in the system is the admin. The admin is responsible for viewing all user appointments, update patient's doctor, generate medical reports and modify user accounts. All these use cases require that this user logs in as admin. However, to modify the user accounts the admin will be asked to choose one of the following update account info, add user account, or delete user account. In addition, when viewing all clinic appointments, the admin may be asked if he/she wants to edit an appointment time.

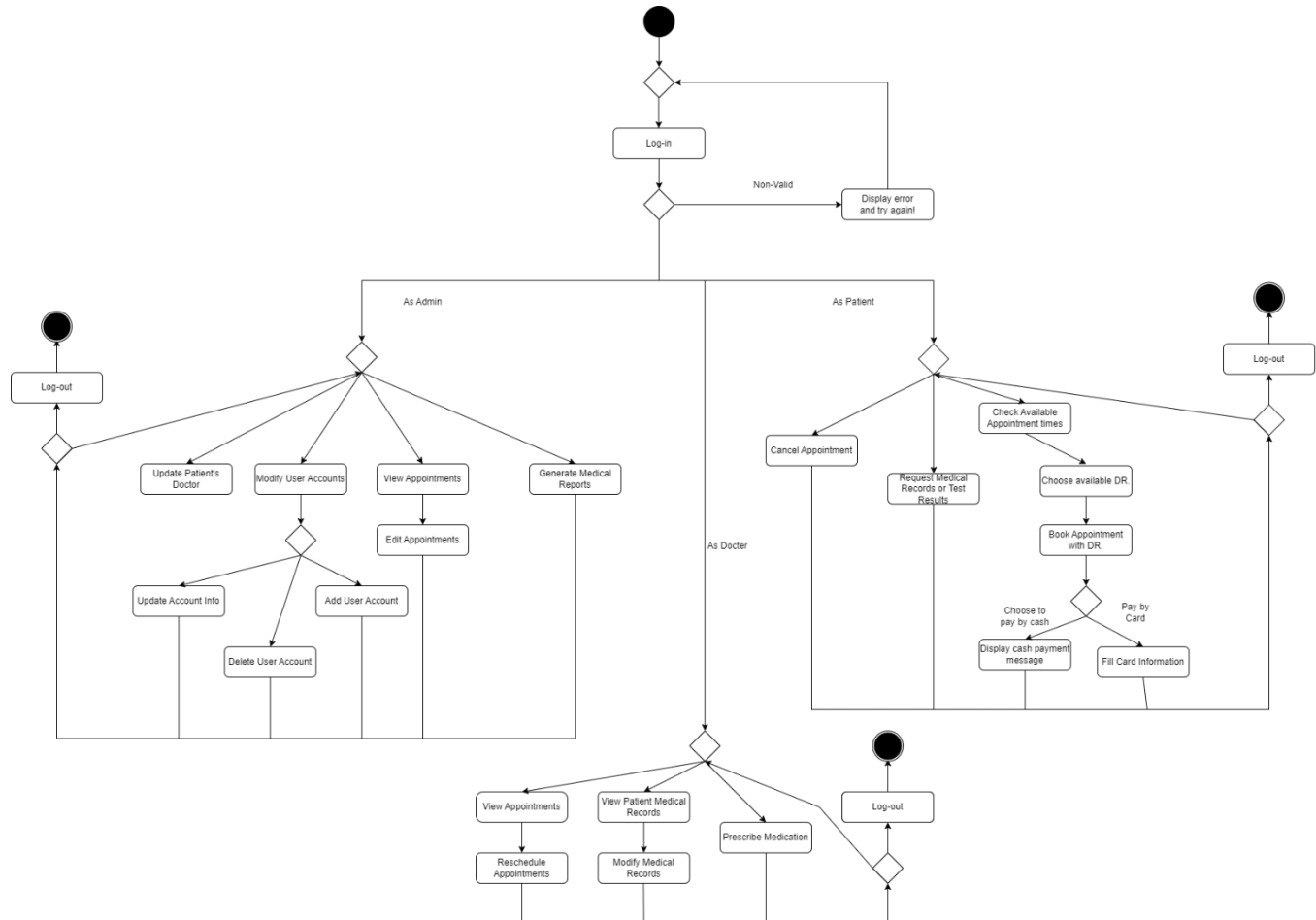
Finally, the third actor is the Doctor which can view his/her appointments with patients and can reschedule to any other time of his availability. Moreover, the Doctor can prescribe medication for his/her patients or can view patients medical records so the doctor can be prepared before the appointment with the patient in case of a checkup, the doctor can also modify his/her patient's medical record providing them with the newest version. These use cases require that the user logs in as Doctor.

The use case diagram allows us to differentiate the different features specific to each user and helps us implement these user-specific features. The only downside to the use case diagram is that it does not specify any flow of execution, nor does it give details of how the features are implemented in the system.

This is why we use other diagrams for this information, which are presented in later sections.

7.2 Activity Diagram

The next diagram to be discussed is the activity diagram, that shows the flow of activities and actions within our system. The figure below shows the activity diagram for the medical clinic application system.

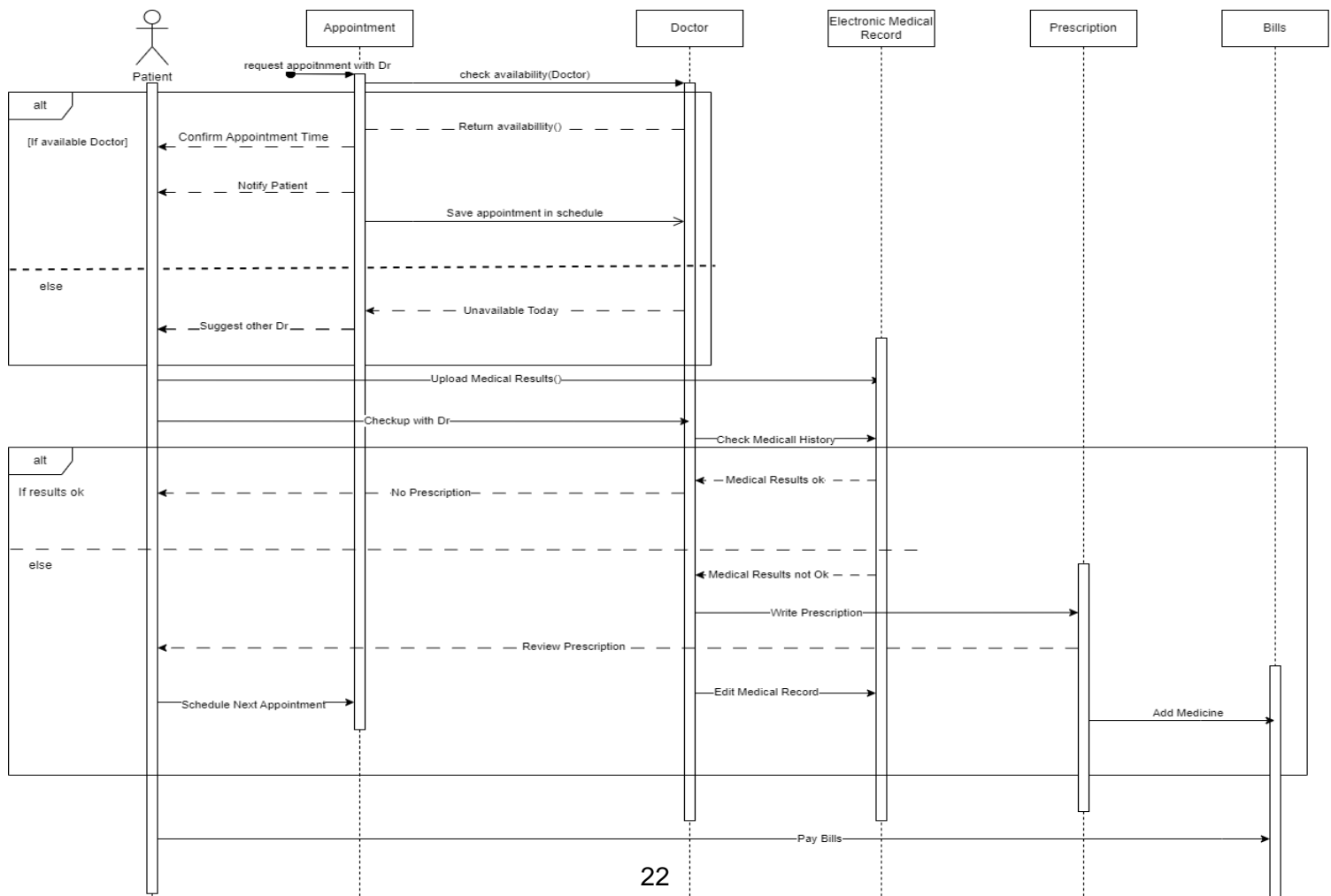


The activity diagram gives much more information than the use case diagram presented in section 7.1. This diagram gives a general overview of how the system will execute and gives the decision at each point along with the options it may lead. At first, a user logs-in to the system, either as an admin, doctor, or patient. There is an option of a non-valid user, which is either a user that is not stored in our database, or a user who has entered incorrect credentials. This is why this option will yield to an error message and prompts the user to try to log-in again. As an admin, the user can generate medical reports, view or edit appointments, update patient's doctor or modify user accounts (either add, delete or update user account). All these features then direct to a decision node, either to log-out from the session, repeat a process or start a new process. Another log-in feature is to log-in as a doctor. This feature provides the user with the authority to view and

manage appointments, access patient medical reports, and prescribe medication. With this feature, doctors can easily view their schedules and make any necessary changes or reschedule appointments. They can also access patient medical reports, which allows them to review and modify information as needed. Additionally, the prescribing function enables doctors to prescribe medication directly through the system. The doctor is then directed back either to log-out or to execute a new process again. As a patient, he/she can check available appointment times and available doctors. If the patient decides to book an appointment with the Dr., he/she is directed to the paying screen where he/she chooses to display the cash payment message, or to fill in the card information. Moreover, patients can request previous medical record or results and can cancel appointments. The customer is then re-directed back to either logout or to execute a new process.

7.3 Sequence Diagram

The following shows the sequence diagram for our system (for the patient user). The sequence diagram shows the exact sequence of events and messages, in chronological order.



The sequence diagram above depicts the sequence of events and messages for a patient, so we assume that the user already logged-in as patient. When a patient makes an online reservation, they use the `availability(date)` method to check the doctor's availability at that time. If the doctor is available, the appointment is confirmed, and the patient is notified. The scheduled appointment is then saved in both the patient and doctor's schedules using `schedAppointment()` function. If the doctor is not available, a doctor with the same specialty is suggested if available on the required date. After the appointment with the doctor, the doctor will upload the medical records. During the check-up, the doctor will review the medical history. If the medical results indicate a problem, the doctor will prescribe the necessary medicine using the `Prescribe(Patient P, Prescription P2)` function, reschedule another appointment, and edit the medical records accordingly. Finally, the patient will pay for the bills using the `payBills()` method.

A class diagram shows the different classes and objects used in a system that is implemented by an Object-Oriented Programming Language. The diagram below shows the class diagram for our system.

Medication

Prescription



The class diagram above shows the eight main classes in our system: Patient, Admin, Doctor, Medication, Medical Record, Appointment, Bills, and Prescription. Each class shown includes its corresponding attributes and methods. The relationship between classes is described below:

- Medication and Prescription are a many-to-many relationship, in the sense that many prescriptions can contain many medication.
- Prescription and Doctor is a one-to-many relationship, in the sense that a doctor can prescribe 0 to many prescriptions for patients.
- Patient and Medication is a many-to-many relationship, where many patients can order many medications.
- Patient and Medical record is a one-to-one relationship, in the sense that an patient has only one medical record.
- Doctor and Admin is a one-to-many relationship, where an admin can view/modify/add/delete many doctors' accounts.
- Appointment and Doctor is a one-to-one relationship, where one appointment is assigned with one doctor.
- Patient and Appointment is a one-to-many relationship, in the sense that a patient can schedule one or more appointments with different doctors.
- Patient and Bills is a one-to-many relationship, where a patient pay many bills given by the clinic.
- Patient and Admin is a one-to-many relationship, where one admin can view/modify/add/delete many patients' accounts.
- Admin and Appointment is a one-to-many relationship, in the sense that an admin organizes 0 to many appointments.

8. Implementation:

LAUMC is a cross-platform mobile application aimed to help Patients manage their reservations and payments, Doctors to update patients' records and manage appointments, and Administrators to better oversee and run the application. Customers have a multitude of functionalities such as looking for a medicine, selecting a doctor and a service (Blood test, PCR, ...), booking an appointment, editing an appointment, and many other options. Moreover, Flutter software along with the Dart programming language was utilized to build the LAUMC application with the database being stored and managed by Firebase. In this document, we will be showcasing eight use cases of our software, with six functions from the patient-side app, and two functions from the admin-side:

Patient Side:

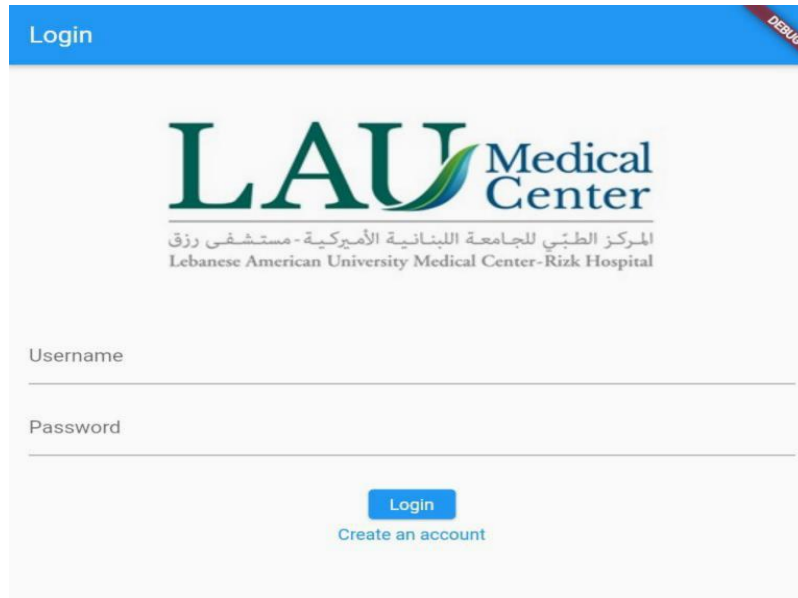
- Logging in
- Signing up
- Scheduling Appointment
- Purchase Medicine
- View Clinic Doctors
- Choose Lab Test

Admin Side:

- Add User
- Delete User

Patient Side Functions:

Logging In:



```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  await Firebase.initializeApp();  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'LAUMC Clinic',  
      theme: ThemeData(  
        primarySwatch: Colors.blue,  
        visualDensity: VisualDensity.adaptivePlatformDensity,  
      ),  
      home: LoginPage(),  
    );  
  }  
}
```

At first, the main function will run and will initialize the firebase and then automatically show the user the Login Page.

```

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}

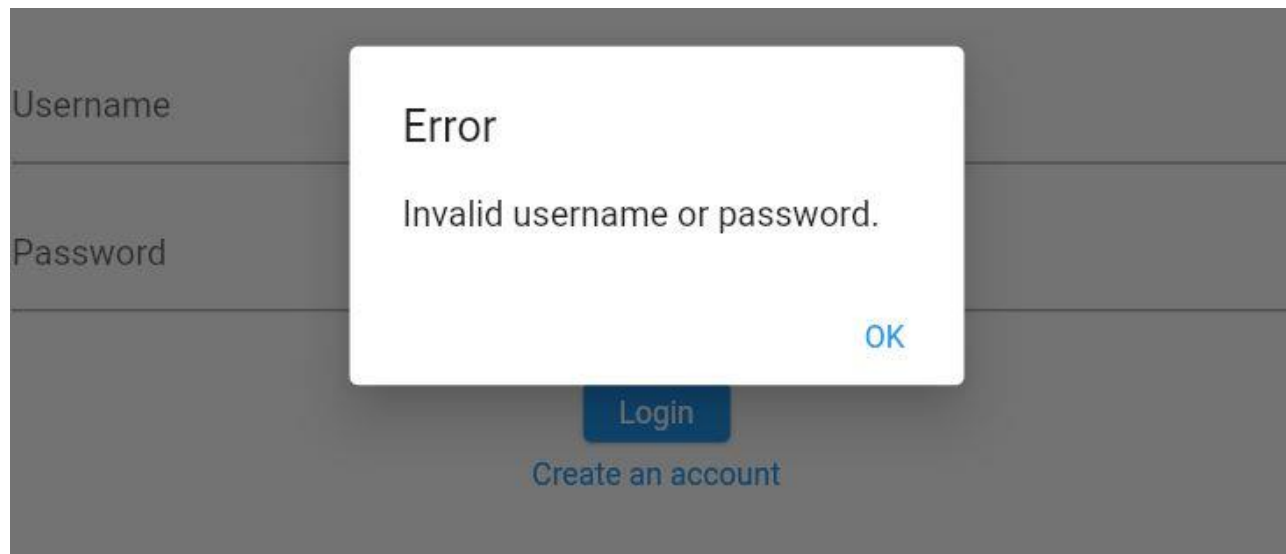
class _LoginPageState extends State<LoginPage> {
  final TextEditingController _usernameController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<void> _login() async {
    // Check if username and password are correct
    String email = _usernameController.text.trim();
    String password = _passwordController.text.trim();

    try {
      // Check if user exists in the database
      bool? userExists = await getUser(email, password);
      if (userExists==false) {
        showDialog(
          context: context,
          builder: (BuildContext context) => AlertDialog(
            title: Text("Error"),
            content: Text("Invalid email or password."),
            actions: [
              TextButton(
                onPressed: () => Navigator.pop(context),
                child: Text("OK"),
              ),
            ],
          ),
        );
      }
      return;
    }
  }
}

```

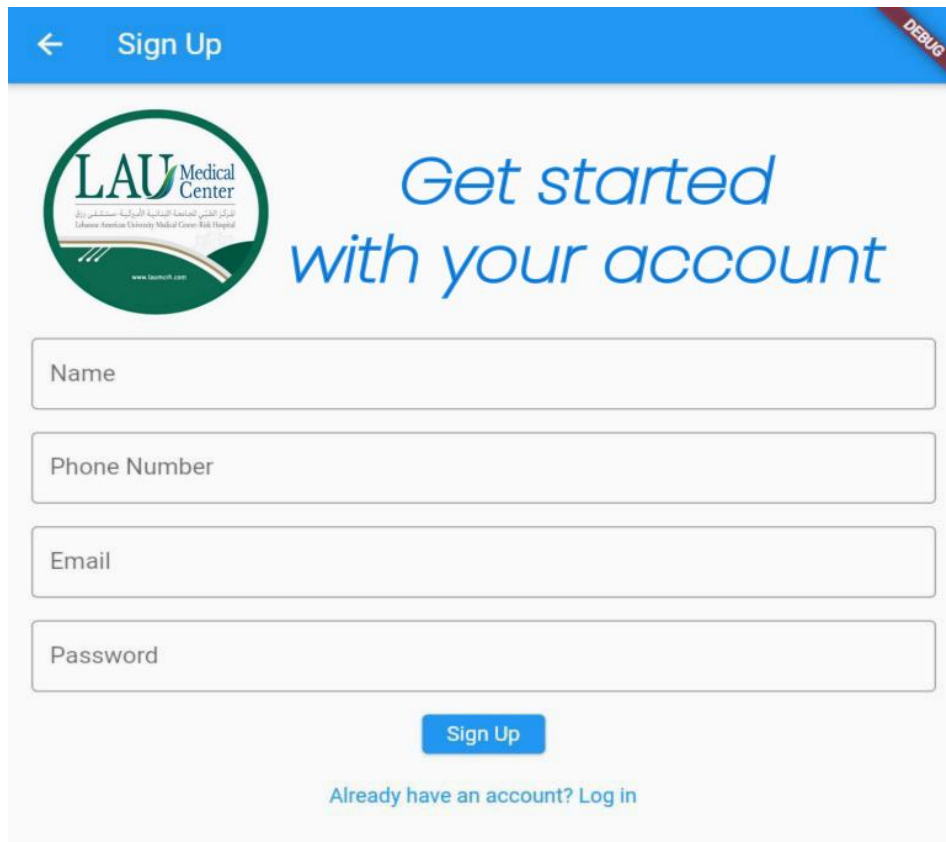
Here two inputs are going to be taken from the user (patient). His/her email and password are going to be then checked if they exist in the database to continue with the login and go to the home page otherwise a message is going to pop-up showing ‘Error-Invalid email or password.’ And the user will remain in the login page.



```
// Sign in the user using Firebase Authentication
UserCredential userCredential = await _auth.signInWithEmailAndPassword(
  email: email,
  password: password,
);
// Navigate to home page if login is successful
Navigator.pushReplacement(
  context,
  MaterialPageRoute(builder: (BuildContext context) => HomePage()),
);
} on FirebaseAuthException catch (e) {
  // Show an error message if login is unsuccessful
  String errorMessage = e.message ?? "An error occurred";
  showDialog(
    context: context,
    builder: (BuildContext context) => AlertDialog(
      title: Text("Error"),
      content: Text(errorMessage),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text("OK"),
        ),
        TextButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SignUpPage()),
            );
          },
          child: Text('Create an account'),
        ),
      ],
    ),
  );
}
```

There is also an option in the login Page if a user does not yet have an account to log in. A small option written in blue saying ‘Create an Account’ can be found under the Login button, which takes the user to the Sign-up Page to create an account so he could later log in easily.

Sign Up:



```
class SignUpPage extends StatefulWidget {  
  @override  
  _SignUpPageState createState() => _SignUpPageState();  
}  
  
class _SignUpPageState extends State<SignUpPage> {  
  final TextEditingController _usernameController = TextEditingController();  
  final TextEditingController _passwordController = TextEditingController();  
  final TextEditingController _nameController = TextEditingController();  
  final TextEditingController _phoneController = TextEditingController();  
  final FirebaseAuth _auth = FirebaseAuth.instance;
```

```

Future<void> _signUp() async {
  // Create a new user account using Firebase Authentication
  String email = _usernameController.text.trim();
  String password = _passwordController.text.trim();

  try {
    UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );

    // Add user to the Firestore database
    String name = _nameController.text.trim();
    String phone = _phoneController.text.trim();
    await addUser(email, password, name, phone);

    // Navigate to home page if sign up is successful
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (BuildContext context) => HomePage()),
    );
  }

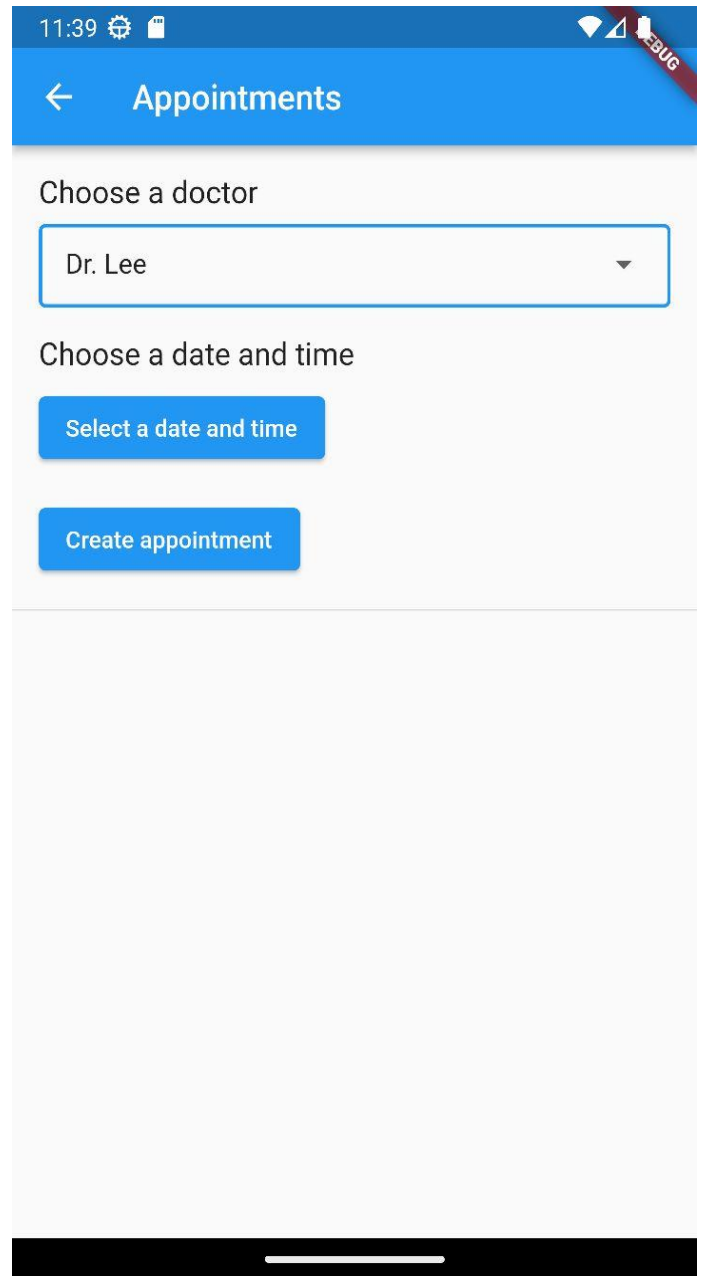
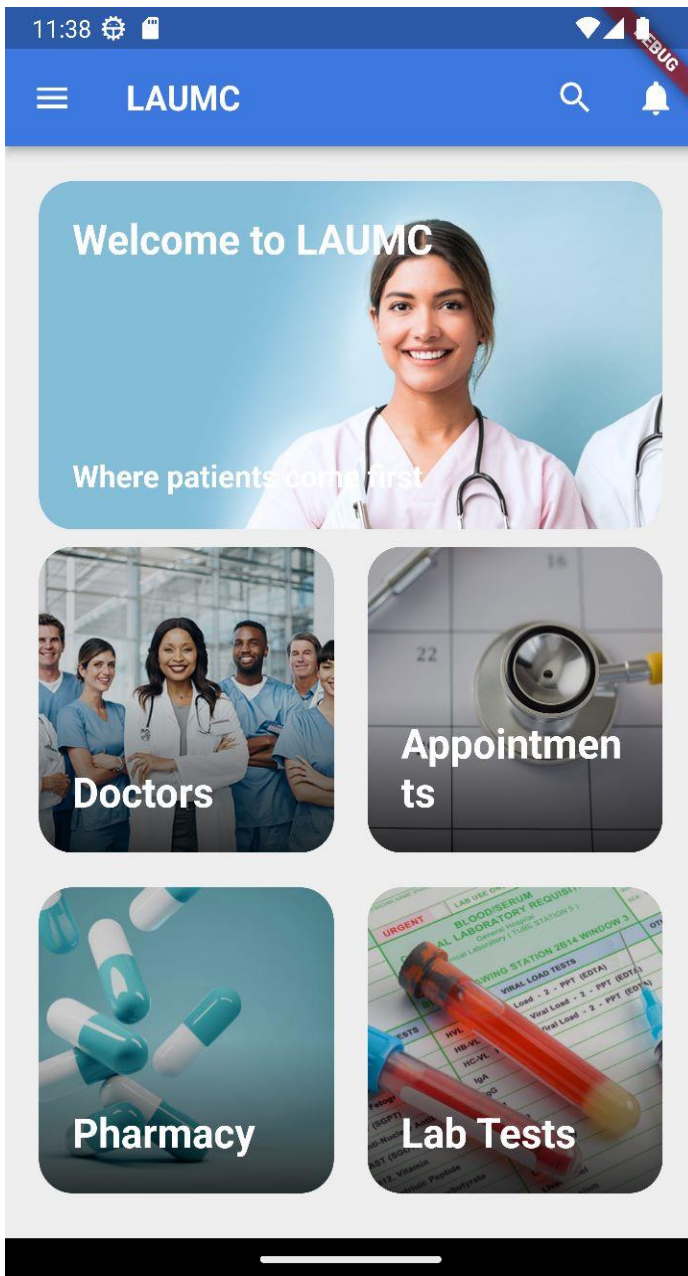
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Sign Up"),
      ),
    ),
  },
}

```

The Sign-Up Page basically takes 4 inputs (email, password, name, and Phone number). Once the input are taken and the user presses the button, his/her credentials are going to be added to the database using the addUser function which exists in the ‘Database.dart’ file and imported here. Note that when a user tries to sign up with an email already being used by another user an error is going to pop-up slightly different from the previous pop-up where this shows ‘Username already taken.’.

Scheduling Appointment:

After signing in, the user will first see the home page and can now access a lot of options by gently clicking on the buttons. One of these options is the appointments page.




```

import 'package:cloud_firestore/cloud_firestore.dart';

class Appointment {
  final String doctor;
  final DateTime dateTime;

  Appointment({required this.doctor, required this.dateTime});

  Map<String, dynamic> toMap() {
    return {
      'doctor': doctor,
      'dateTime': Timestamp.fromDate(dateTime),
    };
  }
}

class AppointmentPage extends StatefulWidget {
  @override
  _AppointmentPageState createState() => _AppointmentPageState();
}

class _AppointmentPageState extends State<AppointmentPage> {
  final _formKey = GlobalKey<FormState>();
  String? _selectedDoctor;
  DateTime? _selectedDateTime;
  List<Appointment> _appointments = [];
  CollectionReference? _userAppointmentsCollection;

  @override
  void initState() {
    super.initState();
    // Replace 'user_id' with the actual ID of the logged in user
    _userAppointmentsCollection = FirebaseFirestore.instance
      .collection('users')
      .doc('user_id')
      .collection('appointments');
  }

  @override

```

After pressing on the ‘Appointments’ widget in the home page. The user will arrive at the appointments page which shows him/her booked appointments if there is any. If not, the user can book a new appointment which will be stored into the user’s record in the database.

```

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Appointments')),
    body: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: const EdgeInsets.all(16.0),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text('Choose a doctor', style: TextStyle(fontSize: 18)),
                SizedBox(height: 8),
                DropdownButtonFormField<String>(
                  decoration: InputDecoration(
                    border: OutlineInputBorder(),
                    contentPadding: EdgeInsets.symmetric(horizontal: 16),
                  ),
                  value: _selectedDoctor,
                  .map((doctor) => DropdownMenuItem(
                    value: doctor,
                    child: Text(doctor),
                  ))
                  .toList(),
                  onChanged: (value) {
                    setState(() {
                      _selectedDoctor = value;
                    });
                  },
                  validator: (value) {
                    if (value == null) {
                      return 'Please choose a doctor';
                    }
                    return null;
                  },
                ),
                SizedBox(height: 16),
                Text('Choose a date and time', style: TextStyle(fontSize: 18)),
                SizedBox(height: 8),
                ElevatedButton(
                  child: Text(_selectedDateTime == null
                    ? 'Select a date and time'
                    : 'Selected date and time: ${_selectedDateTime!.toString()}'),
                  onPressed: () async {

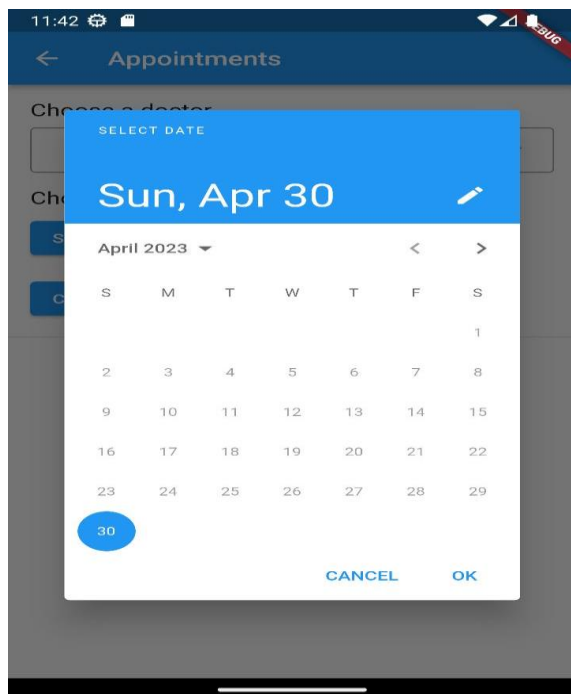
```

```

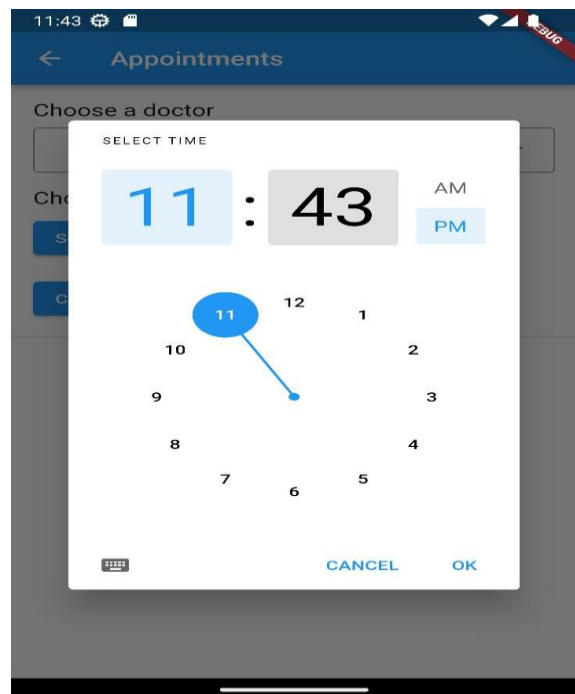
final selectedDateTime = await showDatePicker(
  context: context,
  initialDate: DateTime.now(),
  firstDate: DateTime.now(),
  lastDate: DateTime.now().add(Duration(days: 365)),
);
if (selectedDateTime != null) {
  final selectedTime = await showTimePicker(
    context: context,
    initialTime: TimeOfDay.now(),
  );
  if (selectedTime != null) {
    setState(() {
      _selectedDateTime = DateTime(
        selectedDateTime.year,
        selectedDateTime.month,
        selectedDateTime.day,
        selectedTime.hour,
        selectedTime.minute,
      );
    });
  }
},
),

```

In this section of the code, the user is selecting the Dr. he/she wants to schedule an appointment with, and choosing the date and time available. Note that if a booked appointment is scheduled with a Dr. that already has an appointment at that time an error message will pop-up. Below is the testing part for it.



35



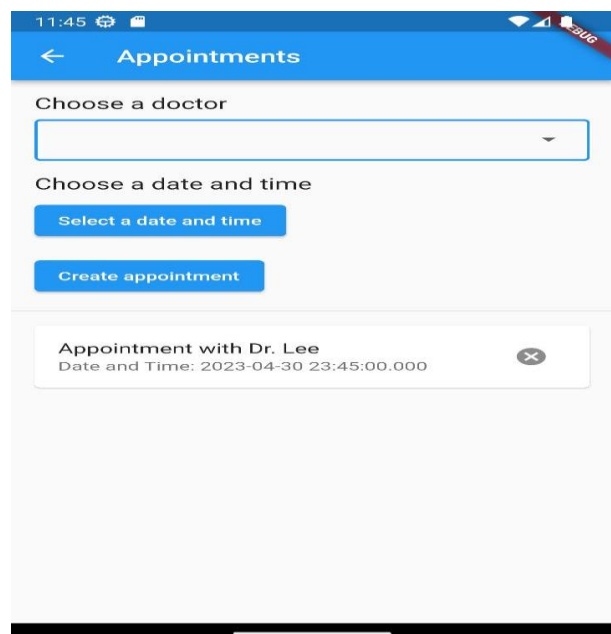
```

        SizedBox(height: 16),
        ElevatedButton(
          child: Text('Create appointment'),
          onPressed: () async {
            if (_formKey.currentState!.validate() &&
                _selectedDateTime != null) {
              final newAppointment = Appointment(
                doctor: _selectedDoctor!,
                dateTime: _selectedDateTime!,
              );
              await _userAppointmentsCollection!.add(newAppointment.toMap());
              setState(() {
                _appointments.add
              if (result != null) {
                _appointments.add(Appointment(
                  doctor: _selectedDoctor!,
                  dateTime: _selectedDateTime!,
                )
              );
              _selectedDoctor = null;
              _selectedDateTime = null;

              // Save the appointment to the user's collection in Firestore
              await addAppointmentToFirestore(Appointment(
                doctor: result['doctor'],
                dateTime: result['dateTime'].toDate(),
              )
            );
          });

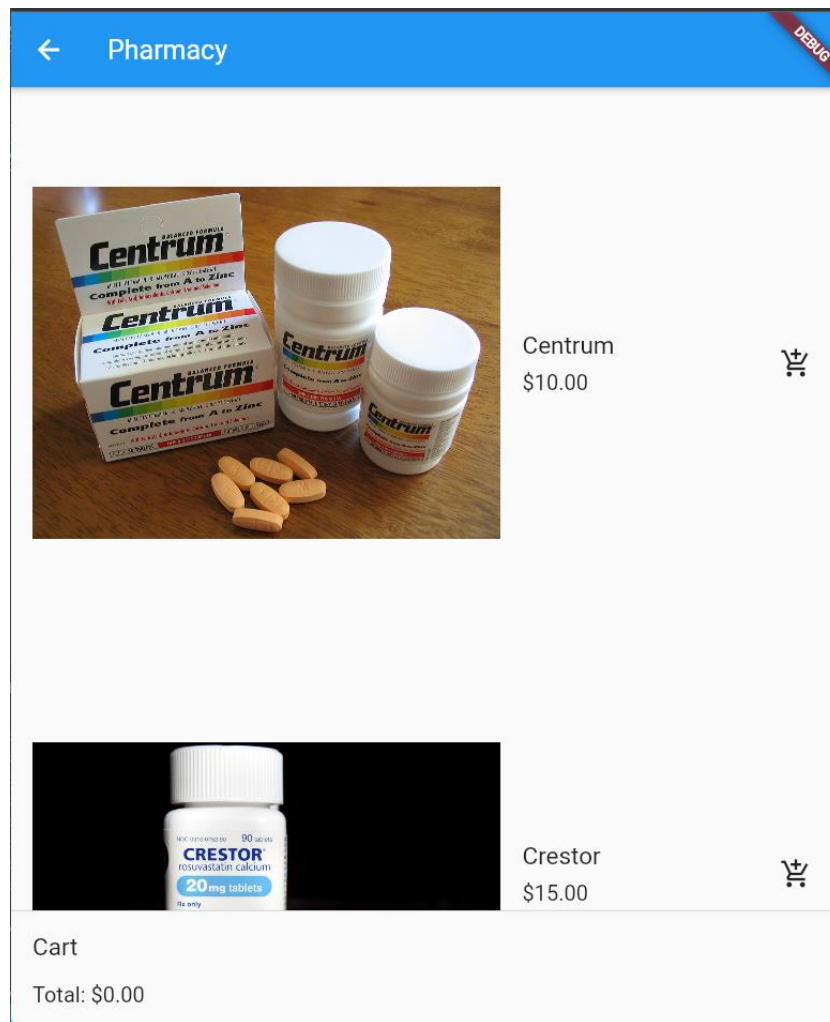
```

Finally, a widget is going to appear showing the appointment details with the Dr. at the bottom.



Purchase Medicine:

By pressing on the Pharmacy widget on the home page you can arrive at this page which shows all drugs available in the clinic. These drugs are all stored in the Database.



```
class Medicine {  
    final String name;  
    final String imageUrl;  
    final double price;  
  
    Medicine({required this.name, required this.imageUrl, required this.price});  
}  
  
class PharmacyPage extends StatefulWidget {
```

```

@override
_PharmacyPageState createState() => _PharmacyPageState();
}

class _PharmacyPageState extends State<PharmacyPage> {
  List<Medicine> _medicines = [];
  List<Medicine> _cartItems = [];

  double _getCartTotal() {
    double total = 0.0;
    for (var item in _cartItems) {
      total += item.price;
    }
    return total;
  }

  @override
  void initState() {
    super.initState();
    _loadMedicines();
  }

  void _loadMedicines() async {
    final medicines = await
FirebaseFirestore.instance.collection('medicines').get();
    setState(() {
      _medicines = medicines.docs.map((doc) => Medicine(
        name: doc['name'],
        imageUrl: doc['imageUrl'],
        price: doc['price'],
      )).toList();
    });
  }
}

```

The List of drugs with their names, pictures, and price are all stored in the database. In this section of code the stored data is retrieved and displayed in the pharmacy page for the user to view the drugs. Also, the patient can add the drug of their choice to his/her cart to sum up the total price.

```

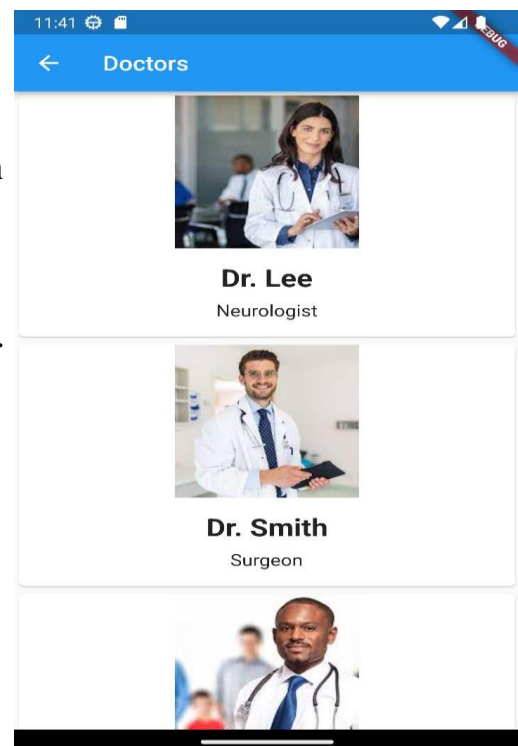
Divider(height: 1, thickness: 1),
  Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text('Cart', style: TextStyle(fontSize: 18)),
        SizedBox(height: 8),
        ListView.builder(
          shrinkWrap: true,
          itemCount: _cartItems.length,
          itemBuilder: (context, index) {
            final cartItem = _cartItems[index];
            return ListTile(
              title: Text(cartItem.name),
              subtitle: Text('\${cartItem.price.toStringAsFixed(2)}'),
            );
          },
        ),
        SizedBox(height: 8),
        Text('Total: \${_getCartTotal().toStringAsFixed(2)}',
          style: TextStyle(fontSize: 16)),
      ],
    ),
  ),
),

```

View Clinic Doctors:

By pressing on the Doctors widget on the home page you can arrive at this page which shows all types of Doctors in the clinic.

These Doctors are all stored in the Database.



```

class Doctor {
    final String name;
    final String specialty;
    final String imagePath;

    Doctor({required this.name, required this.specialty, required this.imagePath});
}

class Doctors extends StatefulWidget {
    @override
    _DoctorsPageState createState() => _DoctorsPageState();
}

class _DoctorsPageState extends State<Doctors> {
    List<Doctor> _doctors = [];

    Future<void> _fetchDoctors() async {
        final response = await http.get(Uri.parse('https://example.com/api/doctors'));
        final data = json.decode(response.body);

        setState(() {
            _doctors = List.generate(
                data.length,
                (index) => Doctor(
                    name: data[index]['name'],
                    specialty: data[index]['specialty'],
                    imagePath: data[index]['imagePath'],
                ),
            );
        });
    }

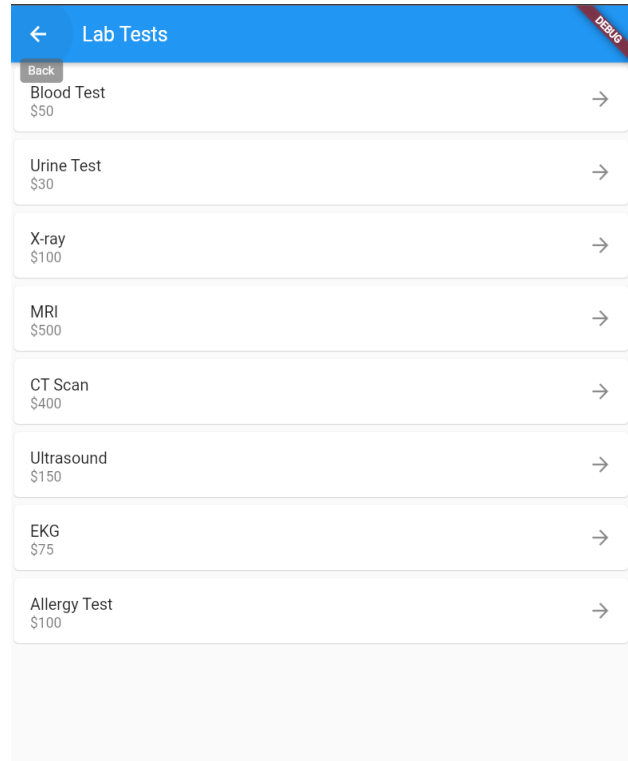
    @override
    void initState() {
        super.initState();
        _fetchDoctors();
    }

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("Doctors"),
            ),
            body: _doctors.isEmpty
                ? Center(
                    child: CircularProgressIndicator(),
                )
                : ListView.builder(
                    itemCount: _doctors.length,

```


Choose Lab Test:

By pressing on the Lab Tests widget on the home page you can arrive at this page which shows all types of medical tests that are offered in the clinic with their details (names, description, and price). These Tests and their details are all stored in the Database.



```
class LabTest {
    final String name;
    final String price;

    LabTest({required this.name, required this.price});
}

class LabTestsPage extends StatefulWidget {
    @override
    _LabTestsPageState createState() => _LabTestsPageState();
}

class _LabTestsPageState extends State<LabTestsPage> {
    List<LabTest> labTests = [];

    @override
    void initState() {
        super.initState();
        _getLabTests();
    }

    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text("Lab Tests"),
            ),
        ),
    }
}
```

```

body: ListView.builder(
  itemCount: labTests.length,
  itemBuilder: (BuildContext context, int index) {
    return GestureDetector(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) =>
              SelectedTestPage(selectedTest: labTests[index]),
          ),
        );
      },
      child: Card(
        child: ListTile(
          title: Text(labTests[index].name),
          subtitle: Text(labTests[index].price),
          trailing: Icon(Icons.arrow_forward),
        ),
      ),
    );
  },
  child: Card(
    child: ListTile(
      title: Text(labTests[index].name),
      subtitle: Text(labTests[index].price),
      trailing: Icon(Icons.arrow_forward),
    ),
  ),
),
class SelectedTestPage extends StatefulWidget {
  final LabTest selectedTest;
  SelectedTestPage({required this.selectedTest});
  _SelectedTestPageState createState() => _SelectedTestPageState();
}
class _SelectedTestPageState extends State<SelectedTestPage> {
  bool _isChecked = false;
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.selectedTest.name),
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          SizedBox(height: 16),
          Text("Price: ${widget.selectedTest.price}"),
          SizedBox(height: 16),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("Add to cart"),
            ],
          ),
        ],
      ),
    );
  }
}

```

Also, the patient can add the Lab test of their choice to his/her cart to sum up the total price.

Admin Side Functions:

An Admin user is basically an entrusted administrator of the firebase database system. In which he can access the database to use any CRUD operation in order to better improve the database and secure it.

Adding User:

The image shows two screenshots of the Firebase Admin console. The top screenshot shows the 'Add user' form with the email 'lm@mail.com' and password 'pass12' entered. The bottom screenshot shows the resulting user list with two users: 'lm@mail.com' and 'ak@mail.com', both created on May 1, 2023.

Add user form:

Search by email address, phone number, or user UID Add user ↻ ⋮

Identifier	Providers	Created ↓	Signed In	User UID
------------	-----------	-----------	-----------	----------

Add an Email/Password user

Email: Password:

Cancel Add user

User list:

Identifier	Providers	Created ↓	Signed In	User UID
lm@mail.com	✉	May 1, 2023		GX1GcbSBcTVaAp1SfSE6svk54Ti1
ak@mail.com	✉	May 1, 2023		ubvcg2HJOpRabTihqfix5eOu4o42

Rows per page: 50 1 – 1 of 1 < >

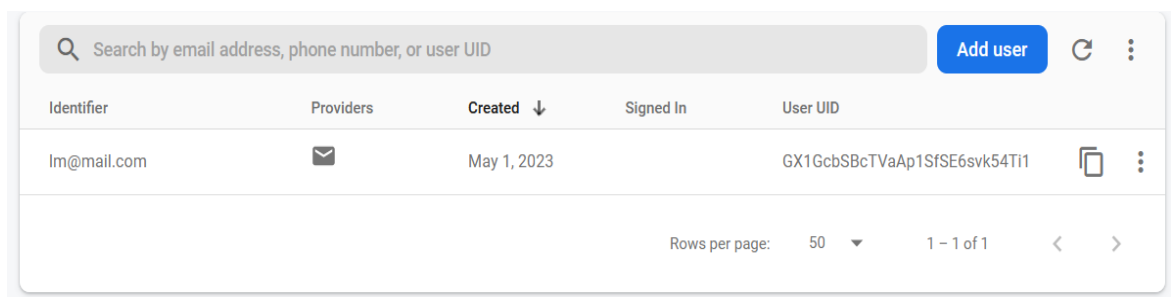
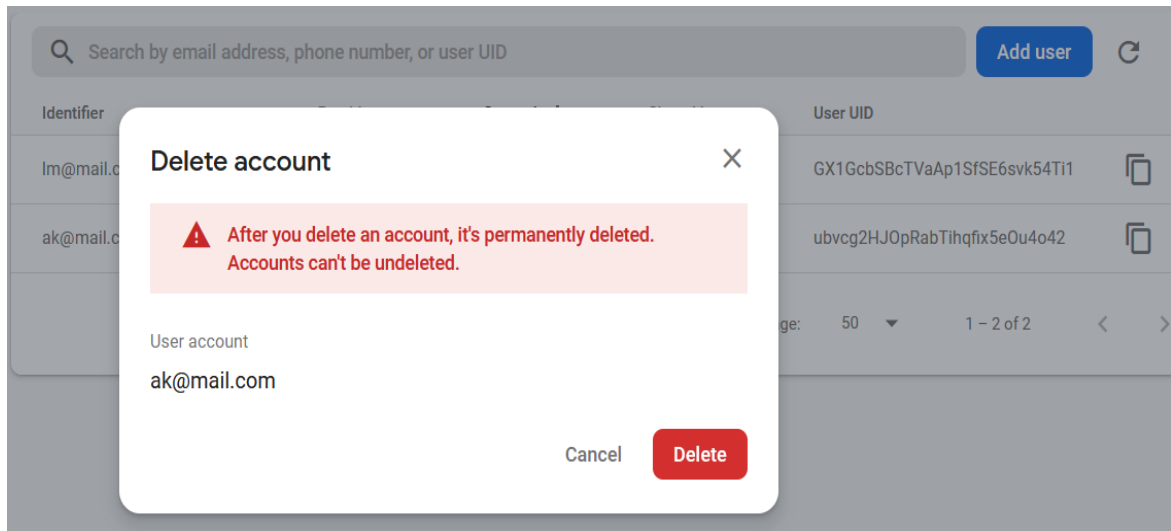
User list (after adding):

Identifier	Providers	Created ↓	Signed In	User UID
lm@mail.com	✉	May 1, 2023		GX1GcbSBcTVaAp1SfSE6svk54Ti1
ak@mail.com	✉	May 1, 2023		ubvcg2HJOpRabTihqfix5eOu4o42

Rows per page: 50 1 – 2 of 2 < >

An admin can also edit the real-time database if he/she has access to it depending on the account privilege and the data's security level.

Deleting User:



Doctor Side Functions:


Log In and Sign Up (Same as the patient)

Other Functions take more time to develop.

Unit Testing:

Sign Up:

1) Successful Signup



Get started
with your account

← Sign Up DEBUG

Name
Adam

Phone Number
12345

Email
ak@mail.com

Password
.....

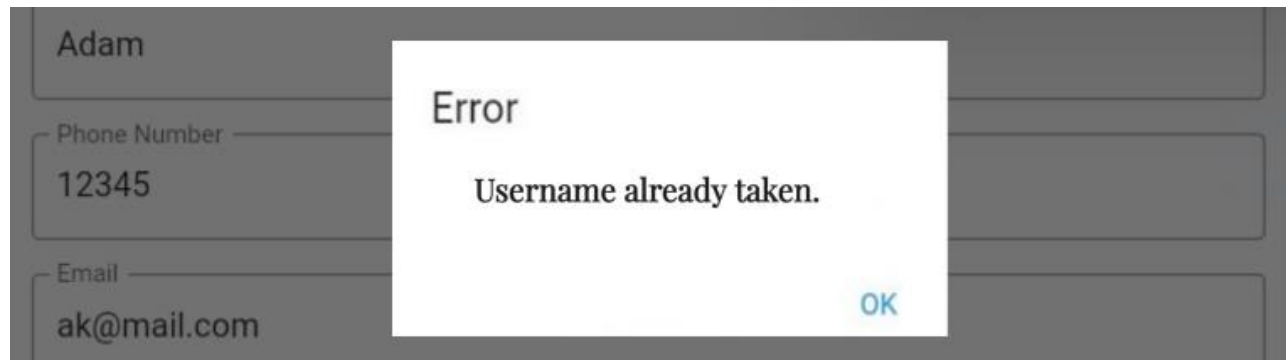
Sign Up

Already have an account? Log in

After signing up

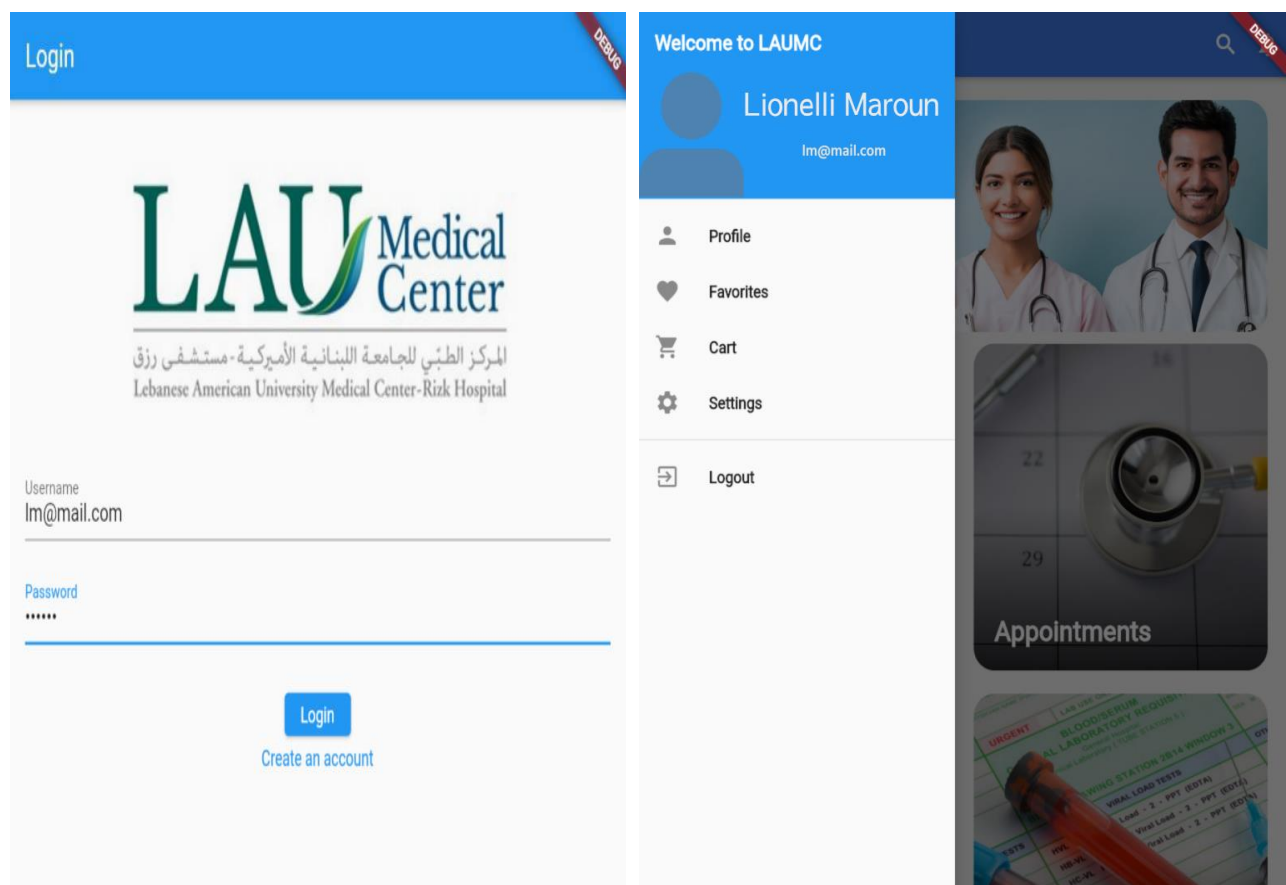
laumc-844c9	userscollection	1
+ Start collection	+ Add document	+ Start collection
userscollection >	1 >	+ Add field
		email: "ak@mail.com"
		name: "Adam"
		password: "pass123"
		phone: "12345"

2) Unsuccessful sign up



Log in:

1) Successful Log in



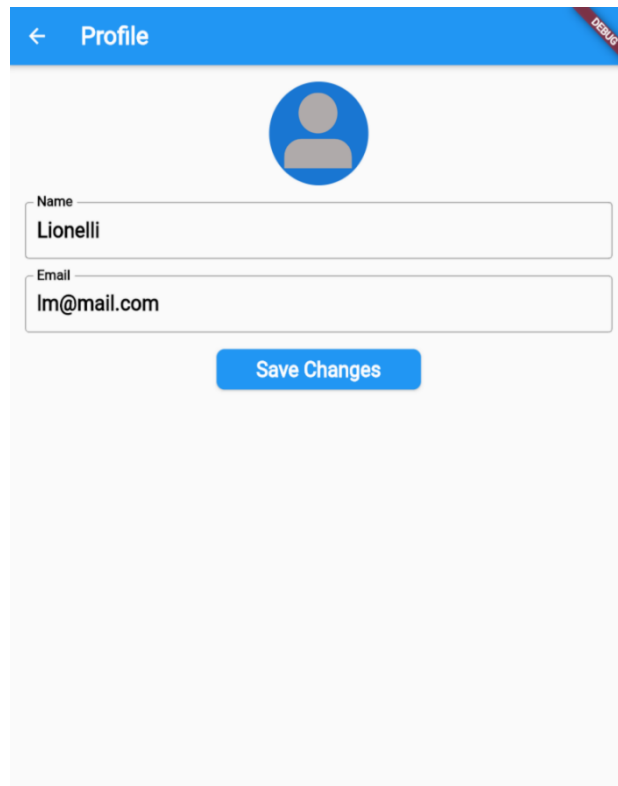
2) Unsuccessful Login is shown before in the LOG IN Page where it shows an error. (Page 29)

Edit Username and email:

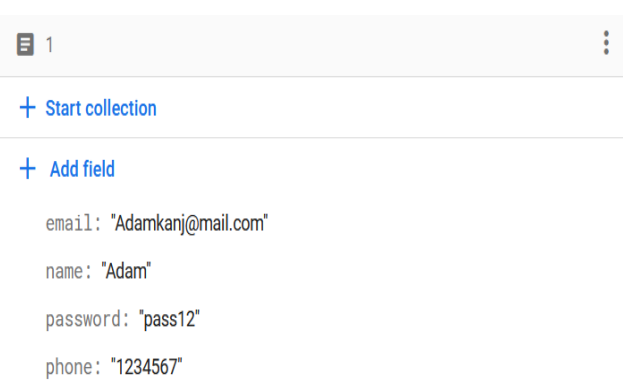
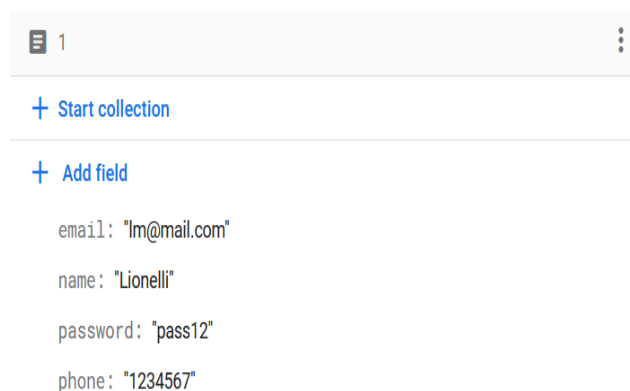
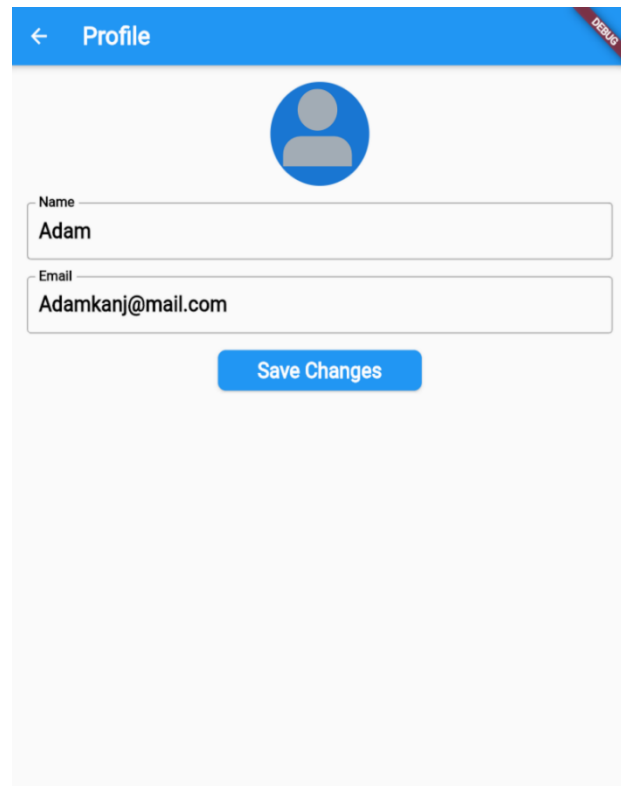
1)Successful update

Click on the menu button inside the home page, then press on profile. Now the user can edit and save his/her altered name and email address respecting all constraints.

Before



After



9. Conclusion

The LAUMC app is a valuable tool for healthcare providers and patients alike, as it offers several features that can streamline the healthcare process and improve the overall patient experience. One of the key benefits of the app is its ability to securely store and manage patient information and medical history. This ensures that healthcare providers have access to critical information when they need it, allowing them to make more informed decisions about patient care.

In addition to managing patient information, the app also offers a range of other features that can enhance the patient experience. For example, patients can use the app to schedule appointments, view test results, and communicate with their healthcare providers. This can help to reduce wait times, improve communication, and ensure that patients receive the care they need in a timely and efficient manner.

Another key benefit of the LAUMC app is its billing and claims management functionality. This helps to simplify the administrative side of healthcare, allowing clinics to focus more on patient care. The app can help to automate many of the administrative tasks associated with healthcare, such as billing and claims processing, which can save time and reduce errors.

Overall, the LAUMC app is an innovative and effective solution for modern healthcare needs. Its range of features and functionality make it a valuable tool for both healthcare providers and patients, and its ability to streamline the healthcare process can improve the overall quality of care. As technology continues to play an increasingly important role in healthcare, apps like LAUMC are likely to become even more important in the years to come.