

**Department of Computer Science and Mathematics**

**CSC 599: Capstone Project**

Instructor: Dr. Faisal Abu Khzam

Students: Abdullah Jrad and Ibrahim Mabrouki

Date: 21/ Apr/ 2025

Table of Contents

[I. Abstract: 3](#_Toc196766860)

[II. Introduction: 3](#_Toc196766861)

[I. Home Page Activity 4](#_Toc196766862)

[II. Login Page Activity 7](#_Toc196766863)

[III. Create Account 10](#_Toc196766864)

[IV. OTP Activity 15](#_Toc196766865)

[VII. Change Password Activity 20](#_Toc196766866)

[VIII. Patient Home Page Activity 21](#_Toc196766867)

[Doctor Main Activity 28](#_Toc196766868)

[ Doctor Overview Fragment 28](#_Toc196766869)

[ Doctor Patient Records Fragment 29](#_Toc196766870)

[ Doctor Schedule Fragment 31](#_Toc196766871)

[ Doctor Messaging System (This activity is shared across staff) 33](#_Toc196766872)

[Pharmacist Main Activity 35](#_Toc196766873)

[ Pharmacist Overview Fragment 35](#_Toc196766874)

[ Pharmacist Storage Fragment 36](#_Toc196766875)

[ Pharmacist Prescriptions Fragment 38](#_Toc196766876)

[Lab Technician Main Activity 39](#_Toc196766877)

[ Lab Technician Overview Fragment 40](#_Toc196766878)

[ Lab Test Orders Fragment 41](#_Toc196766879)

[ Lab Test Results Fragment 42](#_Toc196766880)

[General Features of the Application 44](#_Toc196766881)

[Conclusion 45](#_Toc196766882)

[References 46](#_Toc196766883)

# Abstract:

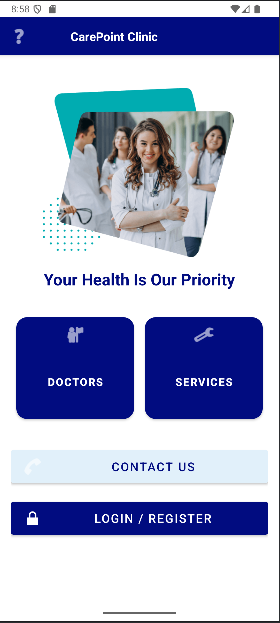
This care point clinic mobile application was designed to optimize a healthcare facility in everyday operations. This application was developed as part of the CSC 599 Capstone Project. Generally, this application integrates real-time reliable messaging, appointments management, prescription handling, and secure authentication. By combining Android development with custom-built TCP server and backend services. This project illustrates how mobile technology can play a vital role in enhancing the health care service delivery. This report gives a thorough overview of the application functioning and room for future development by showing its key features.

# Introduction:

The care point clinic mobile application was created as a comprehensive solution for healthcare operations in response to the demand for more quiet and easily accessible, and real-time management tools integrated with clinics. We aimed to design and developed a role-based android application that speeds up the crucial and regular clinic operations into portable devices. These operations could be appointment scheduling, inventory control, lab test processing, patient record management, and internal staff communication. By providing customized dashboards for lab workers, physicians, pharmacists, and patients, the program makes sure that users only engage with the aspects that are pertinent to their positions. We used the PHP scripts, a customized TCP server for handling real-time messaging, and local server hosting for testing. The focus maintained on the project was objected toward delivering a smooth and safe user experience. This report details the major functionalities, and the lessons learned during the creation of Care Point Clinic.

# Home Page Activity

**General Overview (Home\_page.java):**

****

This Activity serves as the entry point of the Care Point Clinic android application. Regardless of what type of user (including patients, doctors, pharmacists, and laboratory staff) trying to access the application, following the splash screen they will be directed to the central navigation interface “home page activity” before attempting to login or access any provided service/functionality.

The objective of this activity is to offer intuitive, welcoming UI that generally introduces the application features and allows different users to proceed to their next step based on their needs (it reflects the clinic’s professionalism and patient-first approach.). It encourages ease access and guidance users to whether they are booking appointments (for patients), viewing records (for doctors), exploring services and viewing contact information (any user).

1. Header Section:
   * Login Button: this login button allows users of different types to be directed to the login Activity (and access their personalized dashboards).
   * We also have the application title and a certain logo that represents a health center
2. Navigation Cards:
   * **Doctors:** This will allow the users to view the doctors working in the clinic by running a method that calls backend PHP to retrieve their name from the doctor’s table.
   * **Services:** This will allow the user to navigate to a scrollable section (another layout) that displays all the services offered by the Clinic, in addition we have at the end of the scroll a button to show back the home page layout.



1. **Contact:** As in any application we integrated the contact information to allow users or visitors to contact the owner institution directly.

A close-up of a blue and white rectangular sign

AI-generated content may be incorrect.

Clicking the Phone card will launch the device’s dialer with the clinic’s number prefilled to allow the user to contact the clinic directly through phone call.

Clicking the Email card will open the default email agent will the email address of the clinic allowing user to contact the clinic through mailing.

1. Promotional Banner:

This gives a visual and motivational perspective as it includes an image of the medical profession and text (**Your Health is Our Priority**).

1. **Permission handling for notifications:**

To align with latest Android polices, we implemented a runtime permission handling for the notifications as we are required to request “POST\_NOTIFICATIONS”

<uses-permission android:name="android.permission.POST\_NOTIFICATIONS"/>

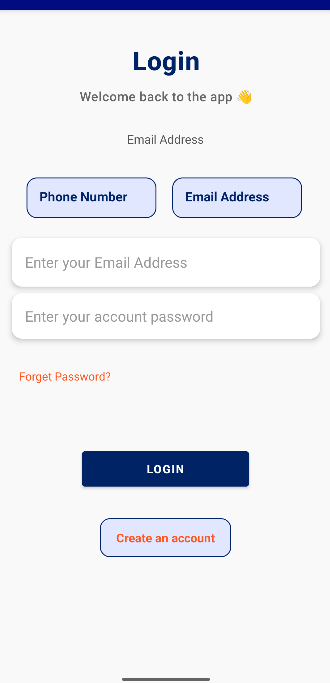
before sending notifications to users. In the home page activity java code and explicitly in the onCreate () we added the following code where it first checks is the API level greater than 33, if so, it then checks if the application grants permission to receive the notification or not and if not, it asks for permission.

// Request notification permission on Android 13+  
if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*TIRAMISU*) {  
 if (ContextCompat.*checkSelfPermission*(this, Manifest.permission.*POST\_NOTIFICATIONS*) != PackageManager.*PERMISSION\_GRANTED*) {  
 ActivityCompat.*requestPermissions*(this, new String[]{Manifest.permission.*POST\_NOTIFICATIONS*}, 101);  
 }  
}

This approach is necessary in healthcare settings where users may receive very critical alerts appointment reminders, new prescriptions, or lab result updates.

# Login Page Activity

**General Overview (LoginActivity.java):**



The login page of the Care Point Clinic ensures secure authentication step to all users. Based on the role of each (account type) it enables patients, doctors, pharmacists, and laboratory staff to access their respective dashboards.

It provides dual mode for logging in, where users are allowed to either use their email address or phone number as an authenticator input. In this activity we tried to make a very smooth flow of validation of the inputs, custom feedback and redirection based on each user type.

This activity handles:

1. **Getting inputs (username/phone + password):**

When the login activity is launched, the user is subjected to UI that offers a **login mode toggle**, allowing the user to choose between logging in with their **phone number** or **email address**. And the interface adapts dynamically based on the selected login mode. If **Phone Number** is selected, the input field updates to accept numeric values, if **Email Address** is selected, the input field updates to expect a valid email format.

A screenshot of a phone

AI-generated content may be incorrect.A screenshot of a phone number

AI-generated content may be incorrect.

Both input fields are subject to real time feedback while the user trying to fill them for example the password is checked for a minimum length.

1. **Validation of inputs:**

As mentioned, we created real-time input validation using the **Text Watcher:**

* **Email mode:** The app validates against known email patterns. Feedback is shown if the format is incorrect or too short.
* **Phone mode**: The system expects 8-digit Lebanese numbers, and users are warned if their input is invalid.
* **Password**: The password must meet a minimum length requirement (8 characters). Users are instantly alerted if they fall short of this requirement.

1. **Role based backend authentication and specific redirection to set of dashboards:**
   * After checking the validity of the inputs written, this activity communicates with the back end via PHP to verify the submitted credentials. Following the successful verification the login page will redirect the user to his personal and specific dashboard based on the value of the user type (echo) by the PHP script
   * In addition, each user type is assigned with a success screen briefly appears with a Visual indicator that matches the user type with customized toast notification.
   * This role-based is beneficial since it makes it possible to maintain a single login screen and at the same time it supports multiple users’ flows within the app.
   * The app also includes error-handling logic for network failures and unexpected backend responses, preventing crashes and improving reliability.

setContentView(R.layout.*activity\_success\_screen*);  
ImageView successImage = findViewById(R.id.*success\_image*);  
Intent intent = null;  
  
if (userType.equals("Doctor")) {  
 successImage.setImageResource(R.drawable.*doctor\_toast\_bg*);  
 showCustomToast("Logged in as Doctor");  
 intent = new Intent(getApplicationContext(), DoctorActivity.class);  
  
} .

.

.

.

final Intent finalIntent = intent;  
final String finalUserId = userId;  
  
new Handler(Looper.*getMainLooper*()).postDelayed(new Runnable() {  
 @Override  
 public void run() {  
 finalIntent.putExtra("user\_id", finalUserId);  
 startActivity(finalIntent);  
 finish();  
 }  
}, 2000);

the above code is found in the function **“private void loginUser (String input, String password)” that** communicates with backend and verify the account.

1. **Account creation and password recovery flows:**

In addition to authentication of the authorized users, this activity also provides direct links to two important features:

* **Create Account:** Those users who are not already registered can be directed to the registration screen to create account and then redirected back to the login page to login (note that this feature is assumed to be used by patients only as we assumed that other user types created by admin) to ensure that not everyone can take a whatever role he/she wants.
* **Password Recovery**: for those users who forget their password they are allowed to change it back by redirecting them to the **OTP** activity as a first step.

# Create Account

**General Overview (CreateAccount.java):**

A screenshot of a login form

AI-generated content may be incorrect.

This activity allows new users (only patient users) to register and account in Care Point Clinic application. This UI supports dual-mode account creation where users are subjected to choose whether they want to create an account using **Email** or Phone **Number** (user-based preference).

Multiple validation methods are used to make sure that the user enters only unique (username and email/phone number), and well-formed credentials are sent to the database.

This Activity handles:

1. **Username Input Field:**

This field allows users to enter an available, valid username field, the entered username is prompted to dynamically (real-time) validation and availability checking.

* The username should be between 4 and 20 characters long
* It contains only letters, digits, or underscores.
* Only valid inputs will be subjected to POST request to the backend (check\_username.php) that checks if the username is already in use, or if it is available. As mentioned, the result is dynamic and immediate feedback is shown.
* @Override  
  public void onTextChanged(CharSequence s, int start, int before, int count) {  
   String username = s.toString().trim();  
    
   if (!username.isEmpty()) {  
   if (!*isValidUsername*(username)) {  
   create\_ed\_username.setError("Invalid username!");  
   }else {  
   checkUsernameAvailability(username, new UsernameCheckCallback() {  
   @Override  
   public void onResult(boolean isAvailable) {  
   if (!isAvailable) {  
   create\_ed\_username.setError("Username already used!");  
   }else {create\_ed\_username.setError(null);}  
   }  
   });  
   }  
   }else {create\_ed\_username.setError(null);}  
  }

1. **Contact Method Selection:**

* Based on his preference can choose the contact info he/she wants to use upon logging in, so we create a toggled UI that allows user to switch between the Email and Phone number.
* By default, the app assumes that the user will register using email address. However, the user can toggle to use phone numbers instead. (note: very similar to the approach used in loginActivty). Choosing either option will dynamically change the setting of the edit text receiving value and place holder.

1. **Contact Value Validation and Availability Check:**

Once the contact method is selected and the user begins to enter the field this activity makes sure that the value is valid using two operations in sequence.

* **If email is selected:**
  + Check if the input shows a regex pattern using the following built-in function.
* **if the phone is selected:**
  + The input must match the pattern of exactly 8 digits using phone.matches("\\d{8}")

After performing the validation, the app sends a POST request to (check\_contact\_value.php), to check whether the submitted user contact information is alread used in the database or not. Thus, we are ensuring by two stages check (format and availability).

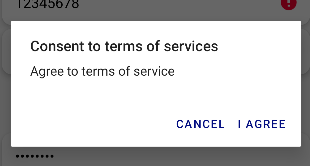
1. **Password and Confirmation Handling:**

In this part the user is required to retype the chosen password for confirmation. As the user types, the password is being checked dynamically and shows real-time errors if occurred. Additionally in password confirmation we compare the confirmation password dynamically with the chosen one using **Text Watcher** and provide the new user with instant feedback.

1. **Sign-Up Button Logic with Consent Dialog:**

The button is very significant as it performs very sensitive logic and serves as the final gatekeeper before initiating the registration. Since here all logic comes together validating input fields, checking availability via the backend, displaying a terms-of-service confirmation dialog, and finally calling the signup script.

* A Consent Dialog is shown when the user taps on the Sign-Up button:



* Handling the “I Agree” Response:

If the user agrees, the app will first retrieve all the input fields: contact method (Phone or Email), contact value, username, and password. And start validating them. And if validation fails a custom toast is shown with a proper message and stopping immediately of the process.

final String type = tag[0];  
final String input = create\_ed\_input\_cred.getText().toString().trim();  
final String username = create\_ed\_username.getText().toString().trim();  
final String password = create\_ed\_password.getText().toString().trim();

* Backend Availability Checks:

Once the input (username, Email/Phone Number) passes the validation we use helper methods to check their availability.

checkUsernameAvailability(username, new UsernameCheckCallback() {  
 @Override  
 public void onResult(boolean isUsernameAvailable) {  
 if (!isUsernameAvailable) {  
 create\_ed\_username.setError("Choose another username");  
 return;  
 }  
  
 checkContactAvailability(input, new ContactCheckCallback() {  
 @Override  
 public void onResult(boolean isContactAvailable) {  
 if (!isContactAvailable) {  
 create\_ed\_input\_cred.setError("Choose another contact Value");  
 }  
 else{  
 signUpUser(type, input, password, username);  
 }  
 }  
 });  
 }  
});

These nested callbacks and we used the callbacks to perform asynchronous backend checks. And finally, after passing the checks we call the signUpUser(type, input, password, username); in order to insert the registered user in the table using the backend script.

@Override  
public void onResponse(String response) {  
 // Handle the response from the server  
 //Log.d("SERVER\_RESPONSE", response); used for debugging  
 if (response.contains("Success")) {  
  
 setContentView(R.layout.*activity\_success\_screen*);  
 ImageView successImage = findViewById(R.id.*success\_image*);  
 successImage.setImageResource(R.drawable.*patien\_toast\_bg*);  
  
 showCustomToast("User registered successfully", R.drawable.*ic\_check*);  
  
 new Handler().postDelayed(new Runnable() {  
 @Override  
 public void run() {  
 startActivity(new Intent(getApplicationContext(), LoginActivity.class));  
 }  
 }, 2000);  
 }

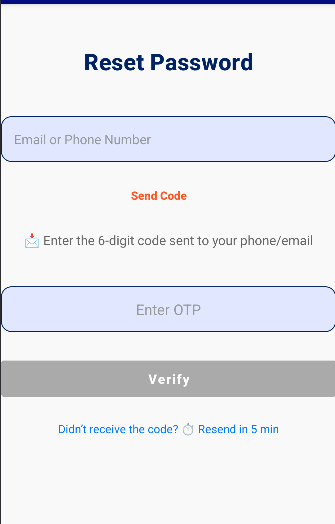
This code is part of the private void signUpUser(String type, String input, String password, String username) where the user will be redirected to login page after a custom toast with different layout (showing the Success layout).

1. **User Feedback and Navigation:**

Throughout the Create Account process, users are continuously guided and informed via real-time feedback mechanisms.

# OTP Activity

**General Overview (OTP\_page.java):**



After the user presses the forget password in the login activity, he/she will be directed to this activity (OTP Verification Activity). This activity serves as the core security step required to reset the password of Care Point Clinic account. This activity is designed to verify the identity of (the ownership of the contact info) to allow users to proceed with password reset.

This includes a specific logical flow. First the user enters their already used contact information (**Email Address or Phone number**), the application tests the existence of the entered value in the database backend. If it exists a randomly generated 6-digit authentication code will be set to the contact information (in our application, we applied it only for the email). The user then enters the code received through the email to the application. After submitting the code through script run by the server that checks the correctness of the code and if it is expired or not, if it is valid then the user will be redirected toto the next activity which user ChangePassword Activity. And we applied a certain logic to prevent the abuse of OTP code sending. Below is the detailed explanation of each functionality.

1. **Collect the User’s Contact (Email or Phone Number):**

At the top of this UI the user is asked to enter their used contact value whether it was Email Address of Phone Number to verify that he/she is the real owner of the account and to know which account the user is trying to reset the password.

1. **Check if the Contact Exists:**

Once the user enters their contact info, it will be tested to be valid and checked whether it is available (registered in the backed) by calling a method (when Send code is clicked) that calls a backend PHP code that fetches the users table for the existence of the user in the users’ table.

private void checkIfContactExists(String contactValue)

to make sure that we don’t send private security codes (**OTP**) to people who don’t even have accounts.

if (isset($\_POST['contact\_value'])) {

    $contact = $\_POST['contact\_value'];

    $stmt = $conn->prepare("SELECT id FROM users WHERE contact\_value = ?");

    $stmt->bind\_param("s", $contact);

    $stmt->execute();

    $stmt->store\_result();

    // Row exists = contact is registered

    if ($stmt->num\_rows > 0) {

        echo "true";

    } else {

        echo "false";

    }

The above is part of the PHP script.

1. **Send a 6-Digit Code (OTP):**

* By passing the above two steps means the contact entered is valid, inside the method:

private void checkIfContactExists(String contactValue)

* we call another method which is responsible for calling another PHP script that generated the random code and send it to the contact value:

public void onResponse(String response) {  
 if (response.trim().equalsIgnoreCase("true")) {  
 requestOtp(contactValue);  
 contactExists[0] = true;}

1. **Limit How Often the Code Can Be Resent:**

* **To not abuse requesting the OTP code we applied the following logic; we included the countdown timer that disables the resending for 5 minutes where neither** Send code at the top of the UI nor the Resend in 5 min will be clickable. The countdown method is directly called after successfully sending the code via requestOtp() method:

private void requestOtp(String contactValue) {  
 String url = "http://10.0.2.2/testfyp/send\_otp.php";  
  
 StringRequest stringRequest = new StringRequest(Request.Method.*POST*, url,  
 new Response.Listener<String>() {  
 @Override  
 public void onResponse(String response) {  
 //making send code disabled  
 sendcontactinfo.setEnabled(false);  
 sendcontactinfo.setTextColor(getResources().getColor(android.R.color.*darker\_gray*));  
 startResendCountdown();

* The below image show how will the UI look like:

**A screenshot of a phone

AI-generated content may be incorrect.**

* We also make sure at the finish of the countdown to reset the UI to its initial state.
* public void onFinish() {  
   resendText.setEnabled(true);  
   resendText.setTextColor(getResources().getColor(android.R.color.*holo\_blue\_dark*));  
   resendText.setText("Resend Code");  
   resendText.setAlpha(1f);  
    
   sendcontactinfo.setEnabled(true);  
   sendcontactinfo.setText("Send Code");  
   sendcontactinfo.setTextColor(getResources().getColor(android.R.color.*holo\_red\_dark*));  
   sendcontactinfo.setTypeface(null, Typeface.*BOLD*);  
   sendcontactinfo.setAlpha(1f);  
    
   otpInput.setText("");  
   verifyBtn.setEnabled(false);  
  }

1. **Let the User Enter the OTP:**

* The user then enters the value of the received code. We added a dynamic check of the code length and as soon as it meets the specified length the “Verify” button becomes activ**e**

**A close-up of a number

AI-generated content may be incorrect.**

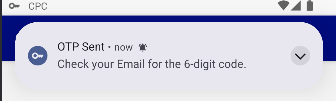
1. **Verify the OTP:**

When the user taps“Verify” it will call the **verifyOtp()** method by passing the received code and the contact value. This method sends POST Request to the PHP script (**verify\_otp.php**) on the server. At the server side it fetches the OTP table to check if there is a record matching the provided contact and OTP and ensures that the OTP has not expired.

* If the java code is true after finding a valid match the application will display a successful screen and redirect the user to the Change Password activity
* If the OTP is incorrect or expired, the server responds with "false", and the app notifies the user with a custom toast message indicating that the code is invalid or has expired.

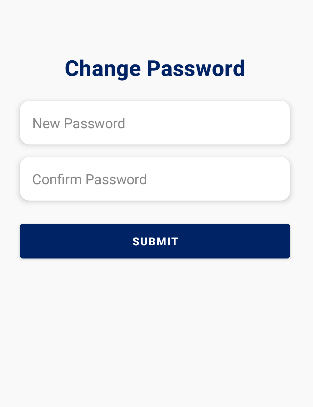
1. **Notify the User That the Code Was Sent:**

Additional feature added is to notify the user with **“OTP Sent – Check your email for the 6-digit code.”** Only if the user allows notifications on the home page.



# VII. Change Password Activity

**General Overview (CreateAccount.java):**



The purpose of this activity is to allow users of the Care Point Clinic application to securely reset their passwords after verifying their ownership of the account based on contact information using the OTP Activity. In addition, this activity ensures that the new password to be used meets the requirements, at the same time it provides visual feedback through the notification alerts and the custom toast.

First the user will fill the required fields (New Password, Confirm Password). Before communicating with the backend, the activity will perform the local validations, followed by pressing the (SUBMIT) button this will call the necessary PHP script that will update the user table and change the password.

Note that the validation of the password is the same as the validation of the CreateAccount Activity.

# VIII. Patient Home Page Activity

**General Overview (Patient Home Page Activity.java):**

The **Patient Home Page Activity** is one of the centrals activities in the Care Point Clinic application. It is well designed to provide seamless and smooth navigation for the users. After the patient logs in to the application, we mentioned that they will be directed to a specific dashboard based on their user type. This activity servs as the entry point to patients after successful login. Below is the overview of activity explaining the main functionalities.

1. **Bottom Navigation Bar**:

This activity uses a navigation bar at the bottom to allow users to navigate easily between different fragments of the activity.

1. **Setting Up Fragments**:

The activity starts by setting the **patient\_overview** fragment as the default fragment when the activity is opened. This gives the user an immediate view of the relevant details, such as their daily health updates or appointments.

* + **Overview fragment (section):** This servs as the default fragment that is displayed when the activity first launched (It shows an overview of the patient's relevant information, such as their scheduled appointments or medical announcements.).
  + **Appointments fragment(section):** This fragment shows a specific UI that allows the patient to reserve an appointment by specifying some attributes such as Doctor they want to visit, Reason for the visit, Day and time.
  + **Records fragment(section):** As we know the patients might receive records when they visit a clinic, these records might be prescription, medical records or Lab Test. This fragment shows all the records related to the patient and he/she might delete them.

Each fragment is replaced without changing the current activity screen. This ensures that the user’s experience is smooth and intuitive.

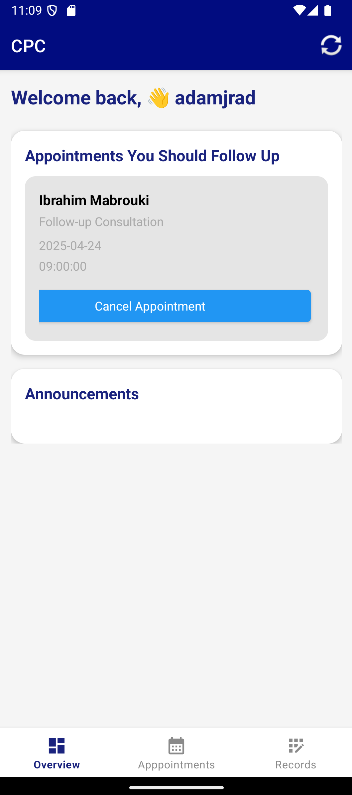
if (savedInstanceState == null) {  
 getSupportFragmentManager().beginTransaction()  
 .replace(R.id.*patient\_fragment\_container*, new patient\_overview())  
 .commit();  
}  
  
BottomNavigationView bottomNav = findViewById(R.id.*bottom\_navigation*);  
bottomNav.setOnItemSelectedListener(item -> {  
 Fragment selectedFragment = null;  
 int id = item.getItemId();  
  
 if (id == R.id.*nav\_overview*) {  
 selectedFragment = new patient\_overview();  
 }  
 else if (id == R.id.*nav\_appointments*) {  
 selectedFragment = new patient\_appointments();  
 }  
 else if (id == R.id.*nav\_records*) {  
 selectedFragment = new patient\_records();  
 }

1. **Refresh Option**:

At the top of **PatientHomePageActivity** we included the refresh opotion in the action bar (accessible via the menu). This will give the users the ability to refresh the currently displayed fragment and if some data has been changed to be updated. Deeply it works as follows checking the activity which is visible and if the fragment implements the abstract class **RefreshableFragment** it will trigger the refresh state method on the that fragment.

Now each fragment will be explained in detail on its own.

1. Patient Overview Fragment:



This fragment is an essential part of the Care Point Clinic, it defaults page launched after redirecting users of type patients and it displays summary of their information and upcoming appointments. It includes the following functionalities.

* 1. **Personalized Greeting:** When a patient logs in, we extract the userId of the patient from the user table and save it in the shared preference to retrieve it later and use it in another method that requires to pass the userId to the backend when calling other PHP scripts. The following is part of the code of the login method of in the Login activity.

SharedPreferences sharedPreferences = getSharedPreferences("MyAppPreferences", *MODE\_PRIVATE*);  
SharedPreferences.Editor editor = sharedPreferences.edit();  
editor.putString("user\_id", userId);  
editor.apply();

In the overview fragment we retrieve the userId from the shared preference to call another PHP script to retrieve the username of the patient based on the userId and the reset the value of the greeting TextView.

SharedPreferences sharedPreferences = requireContext().getSharedPreferences("MyAppPreferences", Context.*MODE\_PRIVATE*);  
userId = sharedPreferences.getString("user\_id", null);

The idea might seem redundant as we could have handled it by sending the username through the intent directly when directing the user to specific dashboard, but this id will be used in multiple activities and fragments.

* 1. **Upcoming Appointments**:
* The **patient\_overview fragment** fetches and displays a list of the patient’s upcoming appointments. this is done by calling an PHP script that fetched the table specified for (appointment reserved by the patients) and display them using customized row that will flattend into a **ListView** through the AppointmentsCustomeArrayAdapter.
* Each appointment is shown with important details: the doctor’s name, the reason for the visit, the date of the appointment, and the time. And they dynamically added to a **ListView**, allowing the patient to see all their upcoming visits in one place.
* So, in general we have a java method called **fetchAppointents** and takes the userId in the **onCreateView**. The **fetchAppointents** receive the data in JSON form from the backend and then pass it to the. The **AppointmentsCustomArrayAdapter** takes this data and populates it inside the **ListView**, which will display each appointment as a list item.

private void fetchAppointments(String userId) {  
 String url = "http://10.0.2.2/testfyp/get\_appointments\_by\_patient.php?patient\_id=" + userId;

1. Patient Appointments Fragment

A screenshot of a calendar

AI-generated content may be incorrect.

This is another important fragment that allows users to manage and book a new appointment, so it allows the patient to schedule appointments with the Doctor, Data, Time they want and the specify the reason for the medical visit. Below is an explanation of its functionalities and how it works in the Care Point Clinic application.

* 1. It is known that one of the problems in scheduling appointments is that they might interleave and this is wrong. We solved this problem by assuming that the working hours of the doctors in the clinic are distributed into equal time slots. So, we display only the available time slots to the user based on the Doctor he/she will proceed with and based on the already available time slots. And this is done by a PHP script the takes as input the date the patient chose to visit the doctor and the doctor’s name. The script will fetch appointments table and remove the time slots that are reserved from the set of all available time slots and then send it to the java **updateTimeSlots()** method in the **patient\_appointments.java** this method will repopulate the spinner after updating the List each time the user updates the chosen doctor from the displayed doctors or changes the data from the calendar**.** Using this approach, we make sure that the user will see only the available times to book appointments and not time interleaved.

$sqlDoctorId = "SELECT doctor\_id FROM doctors WHERE first\_name = ? AND last\_name = ?";

$stmt = $conn->prepare($sqlDoctorId);

$stmt->bind\_param("ss", $first\_name, $last\_name);

$stmt->execute();

$result = $stmt->get\_result();

if ($row = $result->fetch\_assoc()) {

    $doctor\_id = $row['doctor\_id'];

    $sqlAppointments = "SELECT time FROM appointments WHERE doctor\_id = ? AND date = ?";

    $stmt2 = $conn->prepare($sqlAppointments);

    $stmt2->bind\_param("is", $doctor\_id, $date);

    $stmt2->execute();

    $result2 = $stmt2->get\_result();

    $booked = [];

    while ($row2 = $result2->fetch\_assoc()) {

        $booked[] = substr($row2['time'], 0, 5);

    }

    $available = array\_values(array\_diff($all\_slots, $booked));

    echo json\_encode($available);

} else {

* 1. **Doctor Section:** This is similar to the one used in the Home\_Page Activity we used the same java and backed script here but with slight changes where doctor is being displayed in (dropdown menu).
  2. **Submit appointment:** After entering and specifying all the fields they send to PHP backend through POST request to insert them in the database.

1. **Patient Records Fragment:**

* The **patient\_records** fragment is responsible for displaying the personalized medical information of a patient in Care Point Clinic application. These records can be categorized as prescriptions, medical records, or lab tests. The fragment fetches the patient's medical records from the server and presents them in a list view. It works very similar to the overview fragment. Where it retrieves the userId from the shared preference the same way then this userId is passed to the fetchMedicalRecords() function is responsible for fetching the medical records of the patient from the backend. It sends a GET request to the get\_records\_by\_patient.php. The data retrieved from the table holding the records are then passed to the custom adopter the inflates a custom row.

// Query to get records for the specific patient

$sql = "SELECT r.record\_id, r.doctor\_id, r.type, r.content, r.created\_at, r.appointment\_id,

               CONCAT(d.first\_name, ' ', d.last\_name) AS doctor\_name

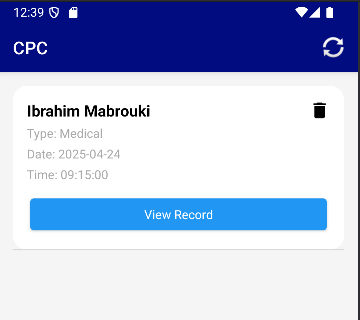
        FROM medical\_records\_pt r

        JOIN doctors d ON r.doctor\_id = d.doctor\_id

        WHERE r.patient\_id = ?";

$stmt = $conn->prepare($sql);

* Since each record might include content that might be long, we made them hidden initially and the user can later press the button that toggles the visibility



A screenshot of a phone

AI-generated content may be incorrect.

* The top right basket icon is responsible for deleting the record by passing the record\_id of the records the patients want to delete to a simple php Script that deletes a row of id = record\_id from the database table.

btn\_delete.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 try {  
 int recordId = record.getInt("record\_id");  
 deleteRecord(recordId);  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
});

# **Doctor Main Activity**

The Doctor Main Activity serves as the central control panel for doctors once they log into the Care Point Clinic app. It acts as a gateway to all other features a doctor needs to manage patients, view schedules, receive messages, and generate medical documents. The design is clean and focused on usability, ensuring that every function is just one tap away using the bottom navigation bar.

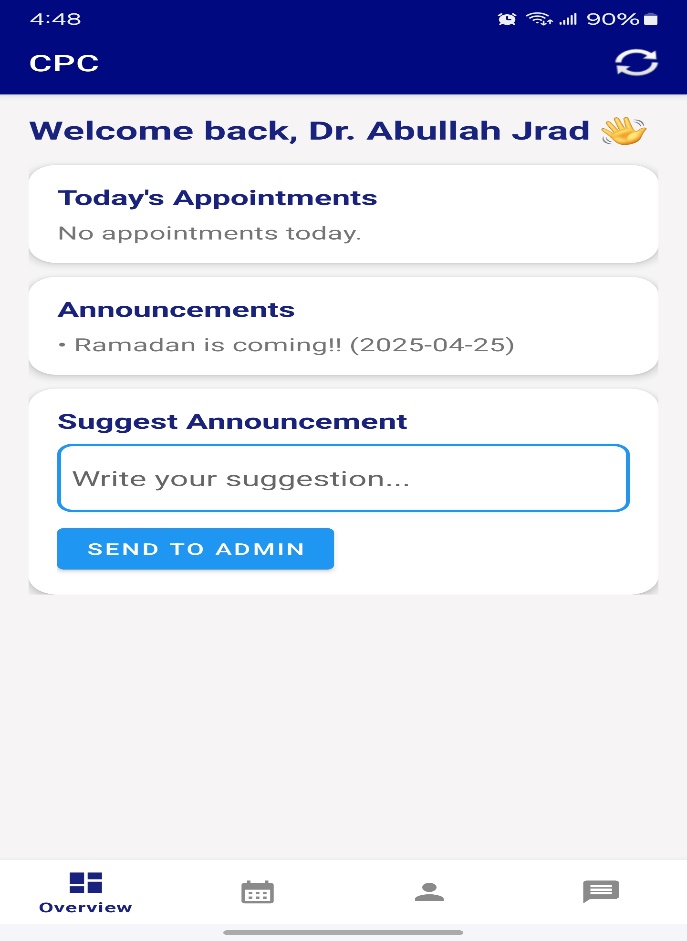
**Key Features:**

* Displays **four main fragments**: Overview, Patient Records, Schedule, Notifications.
* Passes the doctor\_id dynamically to each fragment from the login screen.
* Integrates a **global refresh icon** on the top bar that reloads all the active fragments.
* All fragments are preloaded efficiently using FragmentTransaction for smooth switching.

This activity makes it easier for doctors to manage their day-to-day tasks with minimal friction, reducing the time spent navigating and improving focus on patient care.

## **Doctor Overview Fragment**

Figure 1.1: Overview Fragment (Doctor)



The Overview Fragment is the first screen the doctor sees after logging in. It offers a greeting and a quick glance at important clinic updates. It also provides a suggestion box that allows doctors to send ideas or feedback to the clinic admin, supporting transparency and improvement.

**Main Features:**

* Personalized greeting using the doctor's name.
* Dynamic list of announcements fetched from the backend (Day by Day).
* Input field for submitting suggestions to the admin.

## **Doctor Patient Records Fragment**

Firgure 1.2.1: Records Fragment (Doctor) Figures 1.2.2 and 1.2.3:

A screenshot of a phone

Description automatically generatedA screenshot of a medical form

Description automatically generatedA screenshot of a phone

Description automatically generated

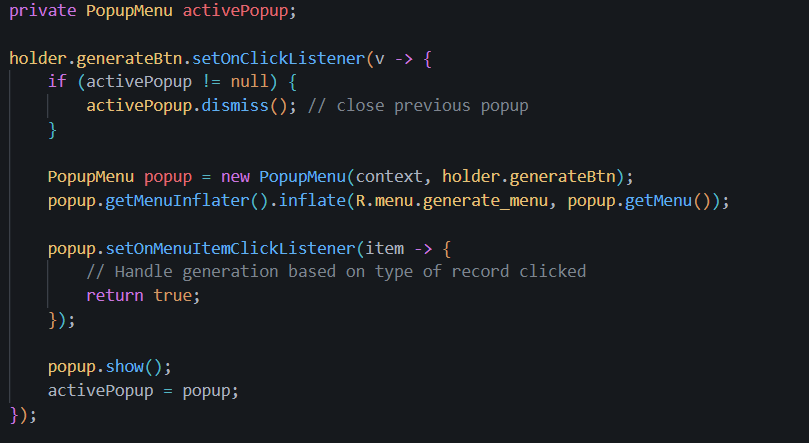
This fragment allows the doctor to view all current patients they are handling. Each patient appears in a card view, showing their basic info with quick-access options. From this screen, the doctor can either **view the patient's full record** or **generate new documents** like prescriptions, medical records, or lab tests.

**Main Features:**

* Dynamic list of patients fetched from the backend, filtered by doctor ID.
* “View Record” button: opens a detailed activity showing the patient’s full medical history.
* “Generate” button: opens a popup menu with options to create:
  + Prescription
  + Medical Record
  + Lab Test
* “Delete” icon: removes a patient from view (soft delete).

The UI is intuitive, and each interaction is fast and role-specific. This fragment helps the doctor manage patient data quickly and securely.

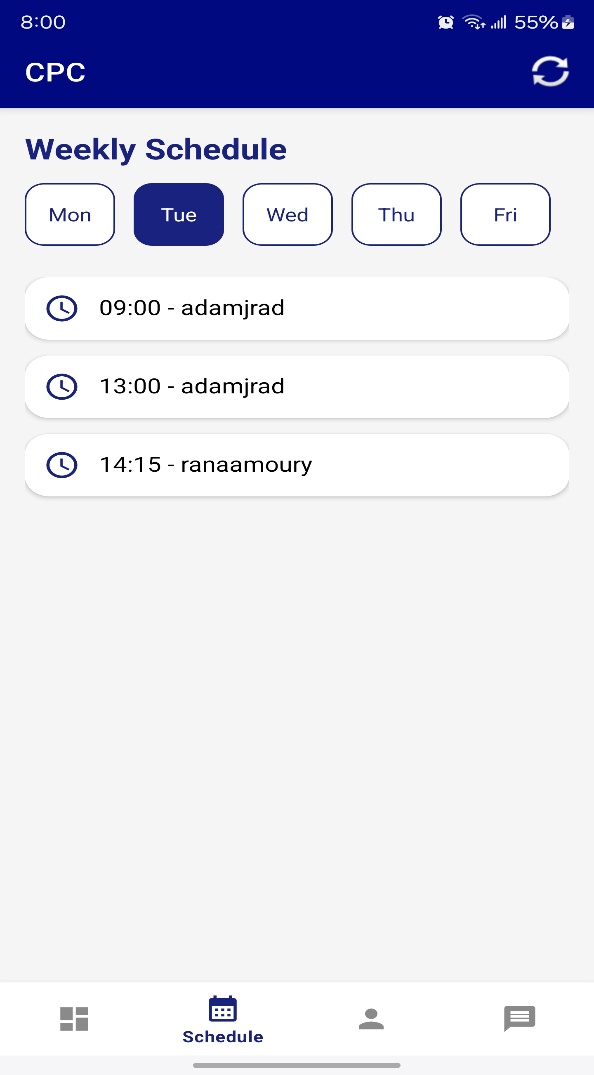
**Code Snippet:**



This logic ensures that only one popup menu is visible at a time. When a doctor taps the “Generate” button on a patient card, it opens a menu with options to create a prescription, medical record, or lab test.

If another popup was already open from a previous card, it automatically closes it before opening the new one. This improves the experience by keeping the interface clean and preventing clutter or confusion.

## **Doctor Schedule Fragment**

Figure 1.3.1: Schedule Fragment (Weekend day) Figure 1.3.2: (Working day)

A screenshot of a phone

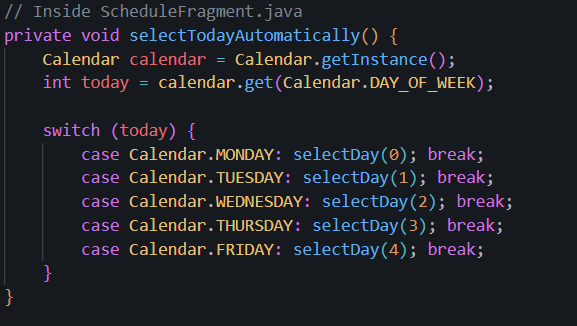
Description automatically generated

The Schedule Fragment provides doctors with a clear overview of their upcoming appointments for the week. The design uses a **horizontal day selector** at the top and dynamically updates the list of appointments below based on the selected day.

**Main Features:**

* Displays days of the week in a horizontal scrollable format.
* Dynamically loads appointments based on the doctor’s ID and selected day.
* Refreshes appointment data when switching days or pressing the global refresh button.
* Organized view showing patient name, appointment time, and reason for visit.
* Automatically selected day

**Code Snippet:**



This method automatically selects the current day when the doctor opens the Schedule screen. It improves usability by directly showing today's appointments without needing extra clicks. It makes the app feel smarter and saves time for doctors checking their daily schedule.

## **Doctor Messaging System (This activity is shared across staff)**

Figure 1.4.1: Messaging System Figure 1.4.2: Inside a chat

A screenshot of a phone

Description automatically generatedA screenshot of a phone

Description automatically generated

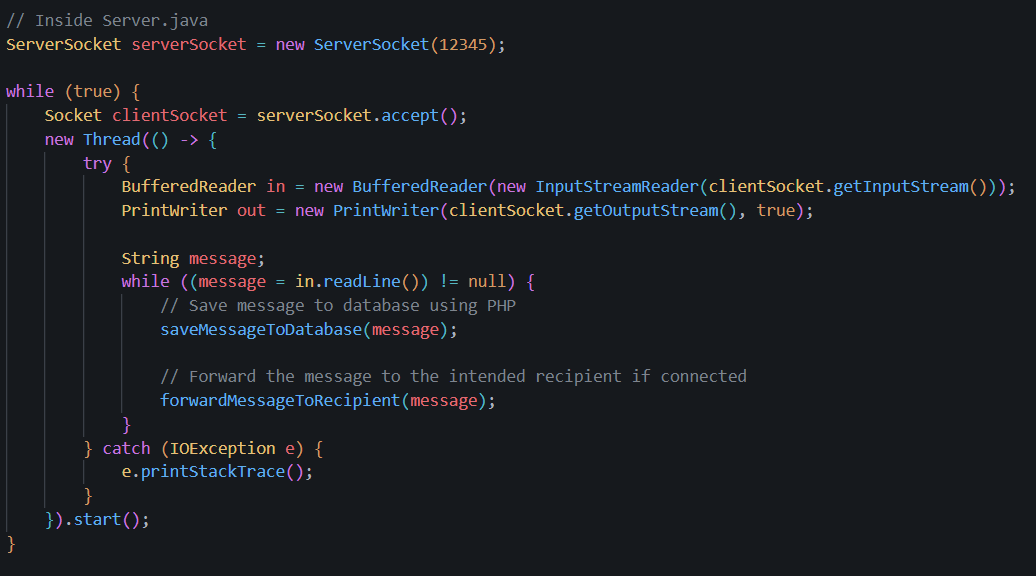
The Messaging System is inside the Notifications Fragment, providing secure 1-to-1 real-time chat between clinic staff (doctors, pharmacists, lab technicians). It also integrates a notification system to alert users about new messages even when the chat is closed. **All roles in the CPC application have this system.**

**Main Features:**

* Displays a list of active chats (contacts) for the doctor.
* Real-time socket communication when connected to the server.
* Polling mechanism activated automatically if the socket disconnects, ensuring message delivery continuity.
* Messages are saved to the backend database with **delivered** and **read** statuses.
* Notifications triggered when a new message is received and the chat is not currently open. With a greed dot on the chat card showing unread message.
* Smooth UI to open individual chat threads, send and receive messages instantly.

This messaging system ensures doctors can communicate with each other, the lab, or the pharmacy team instantly inside the clinic app without needing external apps. It improves collaboration, speeds up service delivery, and ensures that important updates are not missed.

**Code Snippet:**



To power the real-time messaging inside the Care Point Clinic app, we built a custom TCP server entirely in Java. This server listens for incoming socket connections on a fixed port. When a doctor, pharmacist, or lab technician connects, a new thread is created to handle their messages individually.

Each incoming message is processed by:

* Saving it securely into the database using PHP scripts.
* Instantly forwarding it to the correct recipient if they are online.

This custom-built messaging server using **TCP sockets** was a major step in the project. It eliminated the need for third-party services, making the system fully independent, fast, and secure. With this approach, the clinic's internal communication became instant and reliable, similar to professional messaging platforms.

# **Pharmacist Main Activity**

The Pharmacist Main Activity acts as the control panel for pharmacists inside the Care Point Clinic app. After login, pharmacists are redirected to their own dashboard, which provides streamlined access to features needed for medicine management, prescription handling, and clinic communication.

**Key Features:**

* Displays four main fragments: Overview, Storage, Prescriptions, and Notifications.
* Passes the pharmacist\_id dynamically from login to all fragments.
* Clean and role-specific navigation, preventing confusion with other user types.
* Integrated real-time chat and polling system similar to the doctor role.

The main activity was designed to focus specifically on pharmacist workflows, making the app faster and easier to use for pharmacy staff.

## **Pharmacist Overview Fragment**

Figure 2.1: Overview Fragment (Pharmacist)

A screenshot of a phone

Description automatically generated

The Overview Fragment for pharmacists provides a simple and welcoming interface when the pharmacist logs in. It shows important clinic announcements and allows pharmacists to submit suggestions directly to the admin for improvements or feedback.

**Main Features:**

* Personalized greeting using the pharmacist’s name.
* List of low stock medicines in storage.
* List of announcements dynamically fetched from the backend.
* A suggestion submission box, allowing pharmacists to propose ideas or report issues.

This fragment keeps pharmacy staff informed and engaged with the clinic’s activities and ensures they have a voice in operational improvements.

## **Pharmacist Storage Fragment**

Figure 2.2: Storage Fragment (Pharmacist)

A screenshot of a phone

Description automatically generated

The Storage Fragment allows pharmacists to view and manage the clinic’s medicine inventory directly from the app. Medicines are displayed in an organized list, showing their name, quantity available, and an image for easier identification.

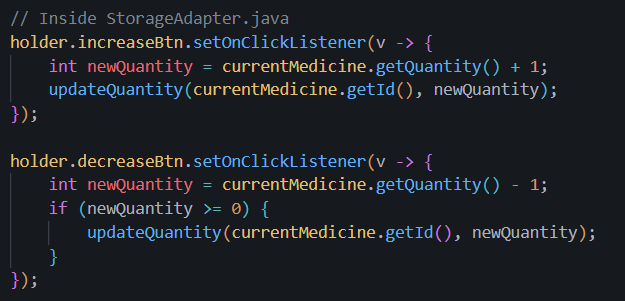
Pharmacists can easily update stock levels by using simple “+” and “–” buttons next to each medicine, ensuring real-time management of medicine quantities.

**Main Features:**

* Dynamic list of medicines fetched from the backend.
* Each item displays medicine name, quantity, and an associated image.
* "+" button to increase the quantity after restocking.
* "–" button to decrease quantity after dispensing medicine.
* Changes are instantly reflected both locally and in the backend database.

This feature greatly simplifies inventory management, making pharmacy operations faster, more accurate, and better organized.

**Code snippet:**



When the pharmacist taps the "+" or "–" button, the app automatically adjusts the quantity in the database using an API call.

The system ensures that stock levels stay accurate in real time without requiring manual data entry, reducing errors and saving time during busy pharmacy hours.

## **Pharmacist Prescriptions Fragment**

Figure 2.3: Prescriptions Fragment (Pharmacist)

A screenshot of a phone

Description automatically generated

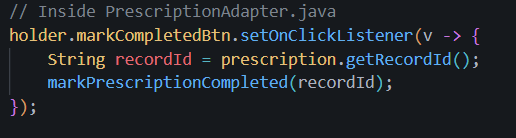
The Prescription Fragment provides pharmacists with a structured list of pending prescriptions that need to be processed. Each prescription is assigned to a specific patient and doctor, ensuring that the pharmacist can handle requests quickly and accurately.

Pharmacists can either view the full prescription details or mark the prescription as completed once the medicine has been prepared and handed over.

**Main Features:**

* Dynamic list of pending prescriptions fetched from the backend.
* “View Record” button: opens a detailed screen showing the full prescription.
* “Mark as Completed” button: removes the prescription from the pending list and archives it into a submitted prescriptions table.
* Confirmation feedback shown after marking as completed to ensure the pharmacist knows the action was successful.

**Code snippet:**



When the pharmacist taps "Mark as Completed," the app sends a request to the backend to move the prescription record from the pending table (**medical\_records**) to the **submitted\_prescriptions** table. This step ensures that only active prescriptions are shown, and completed ones are safely archived for future reference (**hidden\_records**). It supports better pharmacy tracking and reduces clutter in the working interface.

# **Lab Technician Main Activity**

The Lab Technician Main Activity acts as the main dashboard for laboratory staff inside the Care Point Clinic app. It provides quick and structured access to all the features needed to manage lab test orders, generate results, and stay connected with other staff members.

**Key Features:**

* Displays four main fragments: Overview, Lab Test Orders, Lab Test Results, and Notifications.
* Receives the lab\_technician\_id dynamically from the login flow and passes it to the fragments.
* Clean, role-specific navigation to access features related to lab tasks.
* Integrated real-time messaging and notification system similar to other roles.

The design focuses on making lab staff workflows faster, reducing manual paperwork, and ensuring a better experience for patients waiting for results.

## **Lab Technician Overview Fragment**

Figure 3.1: Overview Fragment (Lab Tech)

A screenshot of a phone

Description automatically generated

The Overview Fragment for lab technicians provides a welcoming screen after login, showing any important announcements and allowing lab staff to communicate feedback or ideas to the clinic administration.

**Main Features:**

* Personalized greeting using the lab technician’s name.
* Pending lab tests count for better personalization.
* List of latest announcements dynamically loaded from the backend.
* Suggestion submission box for reporting improvements or issues.

## **Lab Test Orders Fragment**

Figure 3.2: Lab Orders Fragment (Lab Tech)

A screenshot of a phone

Description automatically generated

The Lab Test Orders Fragment allows lab technicians to view all incoming lab test requests assigned to them. Each request comes from a doctor and is linked to a specific patient, ensuring clear tracking and organization of laboratory tasks.

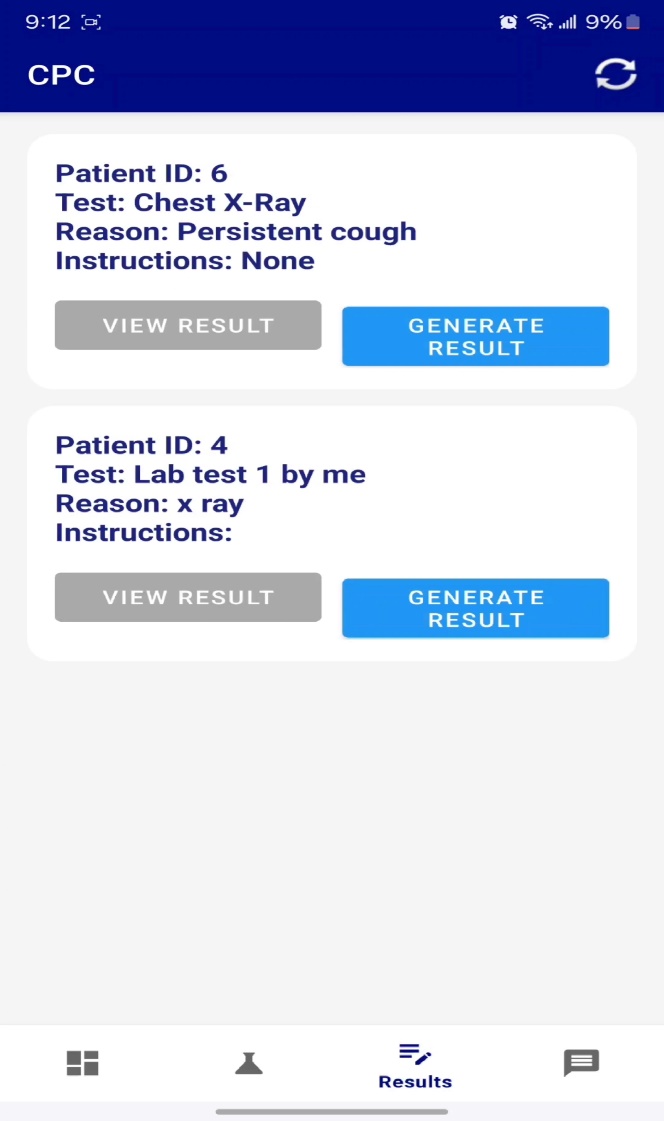
**Main Features:**

* Dynamic list of pending lab test orders fetched from the backend.
* Each order displays the patient’s name, doctor’s name, and requested test details.
* “View” button: allows the technician to view the request for generating a result.

## **Lab Test Results Fragment**

Figure 3.3.1: Lab Results Fragment (Lab Tech) Figure 3.3.2: After generating

A screenshot of a test results

Description automatically generated

The Lab Test Results Fragment allows lab technicians to manage the generation and final submission of lab test results. After receiving a lab test order, technicians can generate detailed reports, review them, edit them if necessary, and then officially submit the finalized results into the clinic’s database.

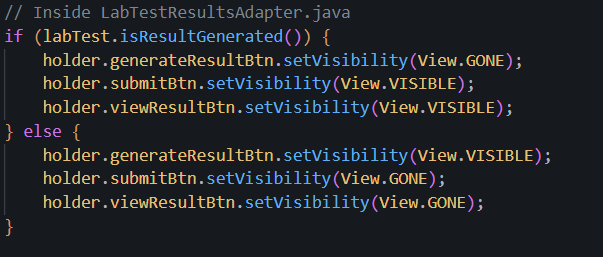
**Main Features:**

* Displays a list of lab test requests that are ready for result generation.
* “Generate Result” button: allows technicians to input lab test findings.
* “Submit Result” button: saves the finalized report permanently in the backend and marks the original order as completed.
* “View Result” button: lets technicians preview the generated result before submission.
* “Edit Result” button (inside the view screen): allows technicians to update or correct the lab result if needed before finalizing.

**Dynamic Button Behavior:**

* If a lab test has **no generated result**, only the **“Generate Result”** button is visible.
* Once a result is generated, the **“Generate Result”** button is hidden, and the **“Submit Result”** and “**View Result**” buttons become visible.
* Inside the result viewing screen, an **“Edit”** button allows technicians to modify or correct the report if needed before submission.
* This flexibility ensures that only accurate and reviewed lab results are saved permanently into the clinic system.

**Code snippet:**



This logic dynamically switches the visibility of action buttons based on the lab test status.  
Before generating a result, only the "Generate Result" button is shown. After generation, the app hides it and shows the "View Result" and "Submit Result" buttons instead, helping guide the technician through the correct workflow.

# **General Features of the Application**

Besides the main modules for patients, doctors, pharmacists, and lab technicians, the Care Point Clinic application includes several system-wide features that make it more flexible, efficient, and professional.

**Key General Features:**

* **Global Refresh System:**  
  Every main screen includes a universal refresh button on the top bar. This allows users to reload updated data instantly without restarting the app, keeping everything dynamic and real-time.
* **Splash Screen:**  
  The app displays a clean and welcoming splash screen with the clinic’s branding and a cool animation. This improves the user experience and adds a better first impression.
* **Dynamic Role-Based System:**  
  After login, each user is redirected automatically to the correct dashboard based on their role (patient, doctor, pharmacist, or lab technician), ensuring that users only access the features intended for their workflow.
* **Real-Time Messaging System:**  
  The application includes a real-time 1-to-1 messaging system between users (staff) using a custom-built TCP socket server. If real-time connection is lost, a background polling system ensures that messages are still delivered reliably from the database.
* **Hosting Server Locally for Testing:**  
  To test and demonstrate the app in a real network environment, the laptop was set up as a **local host server**.  
  The laptop’s local IP address was used, and the phone was connected to the same Wi-Fi network.  
  This allowed opening the app on the phone and communicating with the server just like a live deployed system.  
  Example:

# **Conclusion**

The Care Point Clinic mobile application was successfully developed to address the daily needs of patients and clinic staff including doctors, pharmacists, and lab technicians.

The project achieved its goal of offering a clear, organized, and real-time solution for medical record handling, inventory management, appointment tracking, and internal communication.

Building real-time messaging, dynamic role-based dashboards, and flexible record generation tools inside a single Android application showed how mobile technology can directly improve healthcare service quality.

**What Could Have Been Better:**

* **Socket Server Reliability:**  
  While the TCP server performed well, it could be improved by adding heartbeat signals to better detect dropped connections.
* **User Interface (UI) Enhancements:**  
  Although the app is functional and clean, a more polished design (with animations or dark mode) could improve the user experience even more.
* **Cloud Deployment:**  
  Deploying the server to a cloud platform (like AWS or Azure) would make the system globally accessible and ready for larger production environments.

Overall, the project demonstrated the strong potential of mobile technology for professional clinic management and offered a fully working system that is ready for real-world use.

# **References**

* Github application url – [link](https://github.com/ibrahimmabrouki/CPC_App/tree/Abdullah)
* Github backend url (full php and java server) – [link](https://github.com/Abedsay/Capstone-Project-Backend)
* Android Developers Documentation – [link](https://developer.android.com/studio?gad_source=1&gbraid=0AAAAAC-IOZkLOiHs28vNT5HYy0hd-z6uU&gclid=Cj0KCQjw8cHABhC-ARIsAJnY12xdebjFUQtTome77ZdAoXmyjCki_dderijXLrReka613RrRBRarfYUaAqpXEALw_wcB&gclsrc=aw.ds)