By: Henrietta, Jairo, Scott

Medical Appointment System Report

- Project Scope

- The Medical Appointment System is a web based application that is designed to manage and facilitate the scheduling and cancelling of a user's appointment. It also acts as an interface for doctors to see their patients' appointment schedules. The web based application provides a simple yet user friendly environment to guide users into scheduling an appointment with a doctor while protecting the patients data and identity.

- Objectives

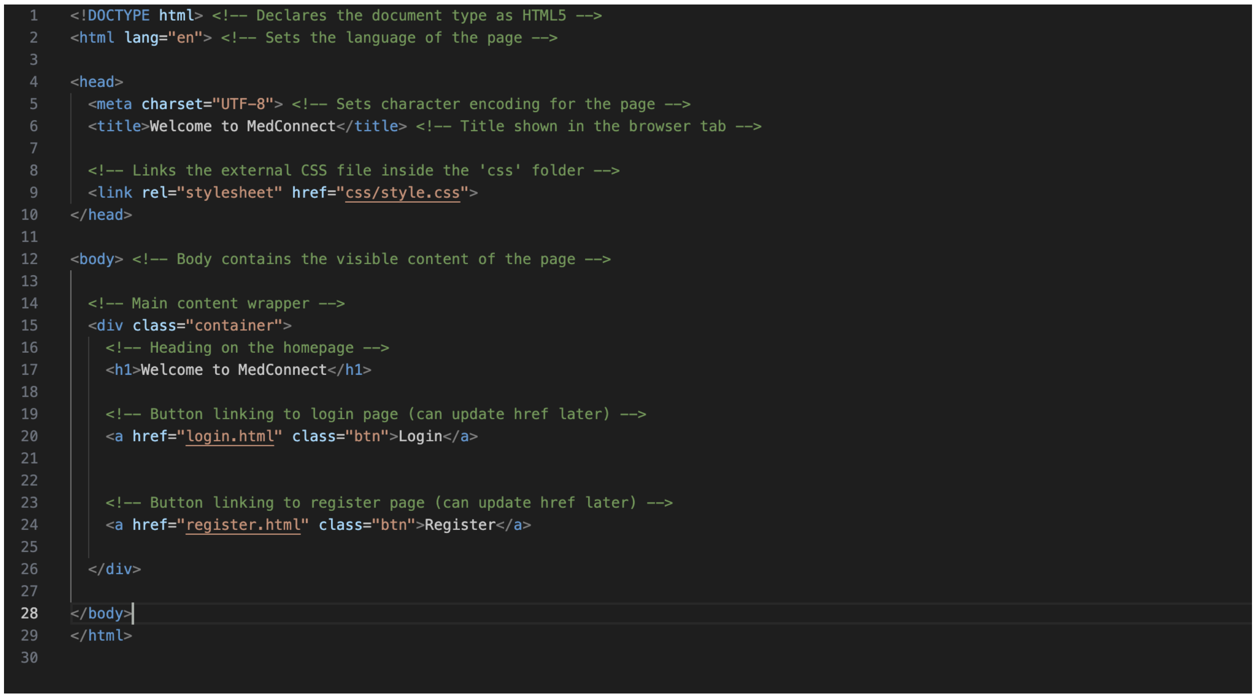
- We split up the group work in terms of two. Henrietta handles the PHP and SQL portion. Jairo handles the HTML and CSS portion. Scott will handle the project report and Github organizing portion. We split it this way because each member can work with something they are capable and comfortable of doing.
- The goal of this project was to make a web server source that allowed a user to enter their information or sign up to create an account in order to log-in and access the services provided by the web server. Services include such as booking appointments and seeing their appointment info. If necessary, the user can also cancel their appointment on the web server itself.
- We implement the use of CSS and HTML for front end coding and use PHP for server side processing
- MySQL for creating the tables between the users/patients and the doctors and for storing information about appointment dates
- Set up a server side processing database to record and store appointment information

- Initial Design Plans

- Front-end: HTML and CSS for user interface and pages
- Back-end: PHP for server-side processing
- Database: MySQL for keeping records of patients and doctors and keeping records of appointments

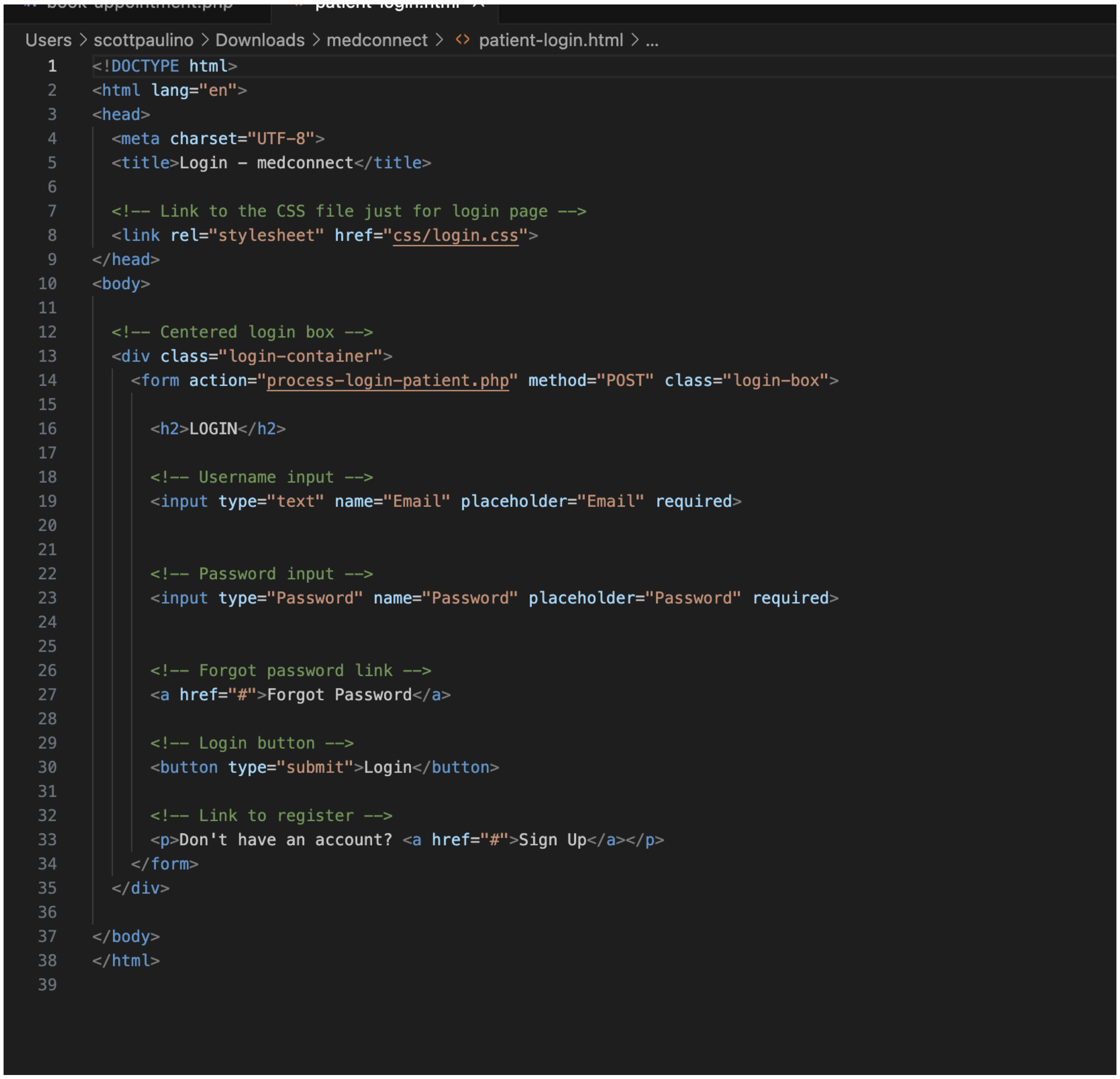
- Functionalities: Users can log into the website or register an account in order to access the web server, Doctors can log in and get special access to their dashboard and view different kinds of information

- Overview



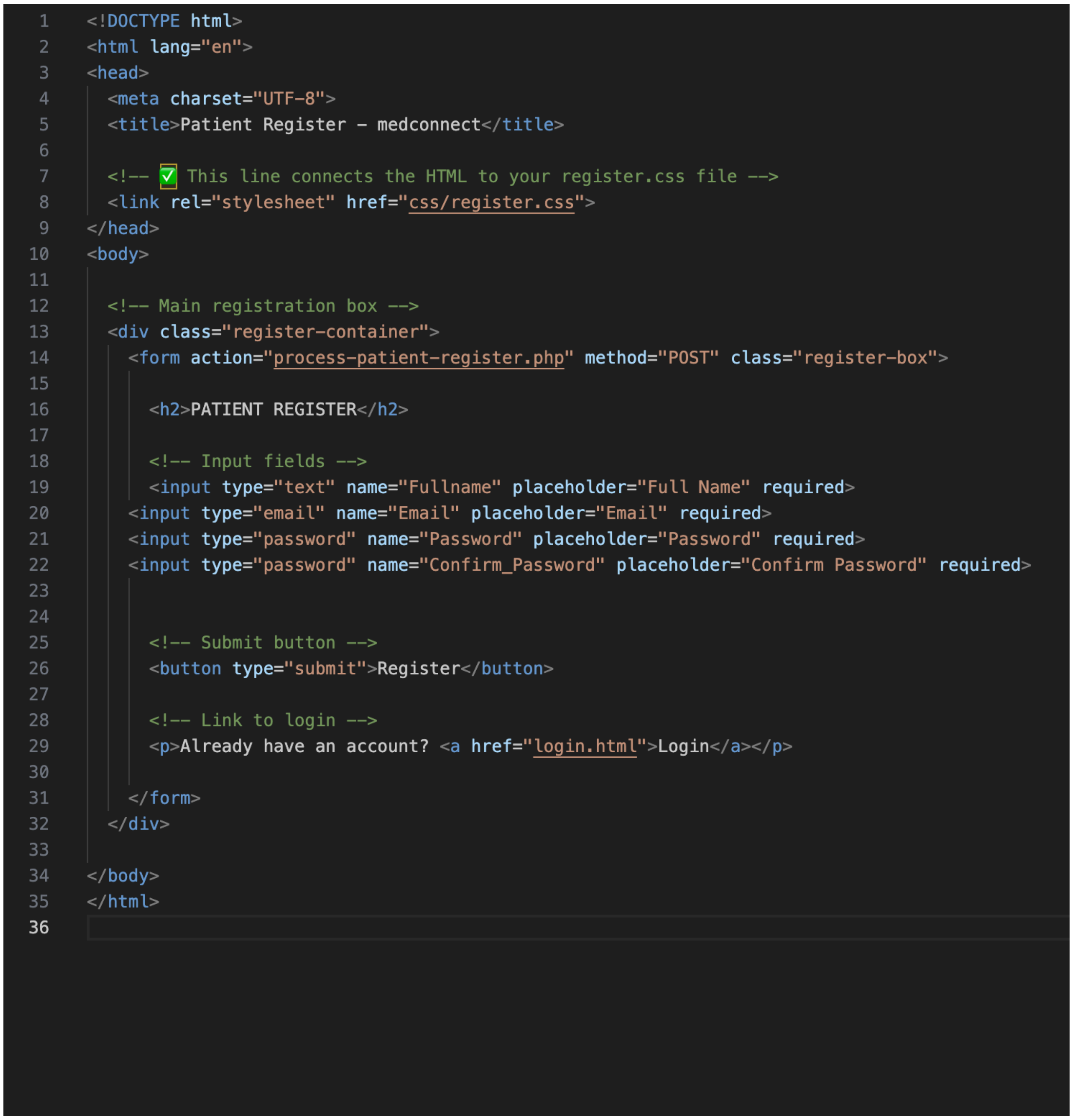
```html
<!DOCTYPE html> <!-- Declares the document type as HTML5 -->
<html lang="en"> <!-- Sets the language of the page -->

<head>
  <meta charset="UTF-8"> <!-- Sets character encoding for the page -->
  <title>Welcome to MedConnect</title> <!-- Title shown in the browser tab -->

  <!-- Links the external CSS file inside the 'css' folder -->
  <link rel="stylesheet" href="css/style.css">
</head>

<body> <!-- Body contains the visible content of the page -->

  <!-- Main content wrapper -->
  <div class="container">
    <!-- Heading on the homepage -->
    <h1>Welcome to MedConnect</h1>

    <!-- Button linking to login page (can update href later) -->
    <a href="login.html" class="btn">Login</a>


    <!-- Button linking to register page (can update href later) -->
    <a href="register.html" class="btn">Register</a>

  </div>

</body>
</html>
```

- The initial step our group did is to make the index section for the web server which is displayed here. A member in our group even took the effort to label each individual piece of code and what it does in order to increase efficiency between him and the PHP scripter. The file that holds all the codes are made to make the web server run and active but for this portion will show some of the more important scripts and files needed to make this mini-world scenario applicable.


Users > scottpaulino > Downloads > medconnect > patient-login.html > ...

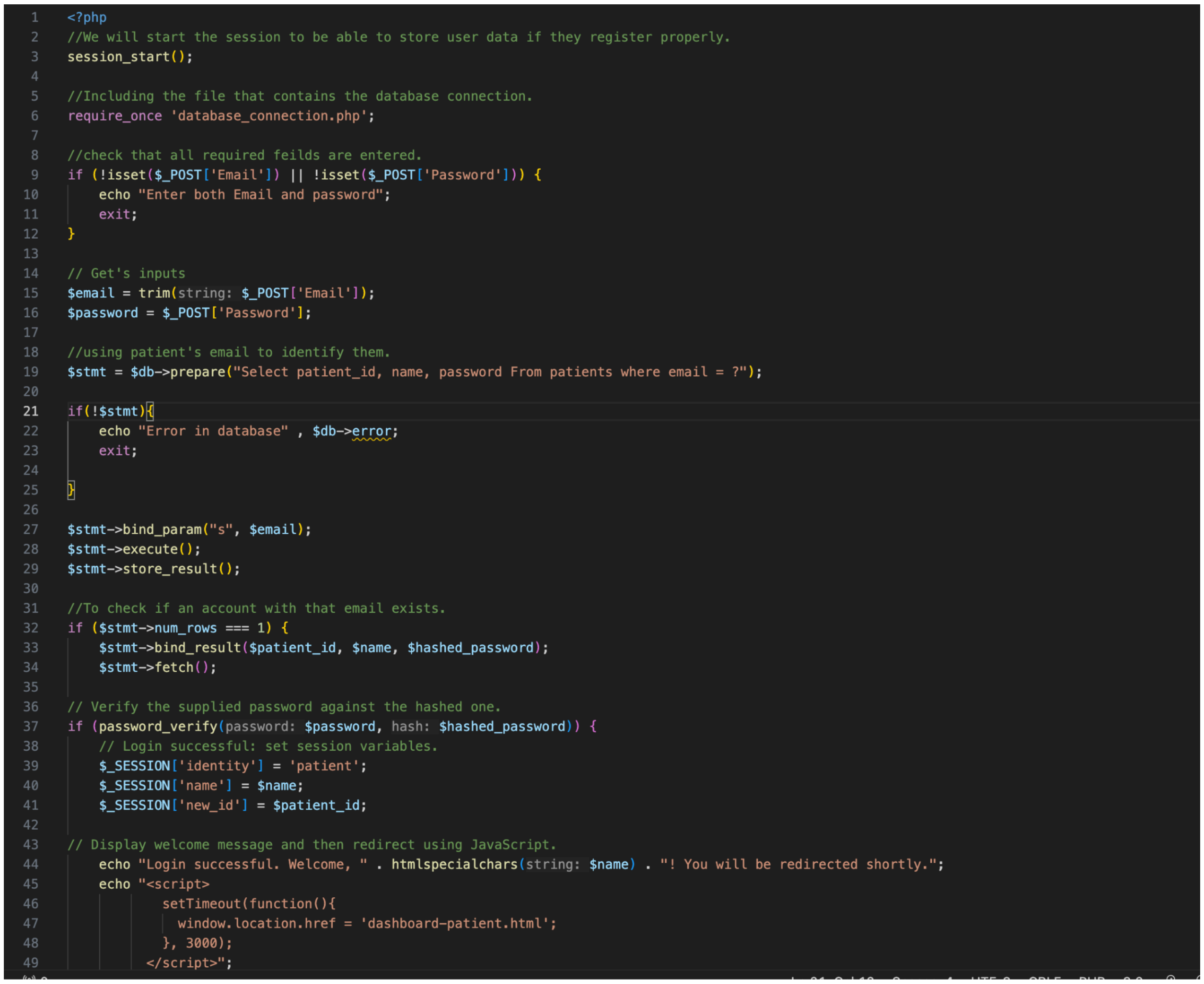
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Login - medconnect</title>

  <!-- Link to the CSS file just for login page -->
  <link rel="stylesheet" href="css/login.css">
</head>
<body>

  <!-- Centered login box -->
  <div class="login-container">
    <form action="process-login-patient.php" method="POST" class="login-box">

      <h2>LOGIN</h2>

      <!-- Username input -->
      <input type="text" name="Email" placeholder="Email" required>


      <!-- Password input -->
      <input type="Password" name="Password" placeholder="Password" required>


      <!-- Forgot password link -->
      <a href="#">Forgot Password</a>

      <!-- Login button -->
      <button type="submit">Login</button>

      <!-- Link to register -->
      <p>Don't have an account? <a href="#">Sign Up</a></p>
    </form>
  </div>

</body>
</html>

```

- This HTML code defines a login page for **MedConnect.** It includes a structured form that submits user credentials via POST to process-login-patient.php. The form consists of an email input, a password input, a login button, a forgotten password link, and a sign-up link for new users. The page is styled using an external CSS file (login.css), with a centered layout inside a <div class="login-container"> and the form wrapped in <div class="login-box">. The form elements include placeholders and required attributes for basic validation.



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Patient Register - medconnect</title>

  <!-- ✅ This line connects the HTML to your register.css file -->
  <link rel="stylesheet" href="css/register.css">
</head>
<body>

  <!-- Main registration box -->
  <div class="register-container">
    <form action="process-patient-register.php" method="POST" class="register-box">

      <h2>PATIENT REGISTER</h2>

      <!-- Input fields -->
      <input type="text" name="Fullname" placeholder="Full Name" required>
    <input type="email" name="Email" placeholder="Email" required>
    <input type="password" name="Password" placeholder="Password" required>
    <input type="password" name="Confirm_Password" placeholder="Confirm Password" required>


      <!-- Submit button -->
      <button type="submit">Register</button>

      <!-- Link to login -->
      <p>Already have an account? <a href="login.html">Login</a></p>

    </form>
  </div>

</body>
</html>

```

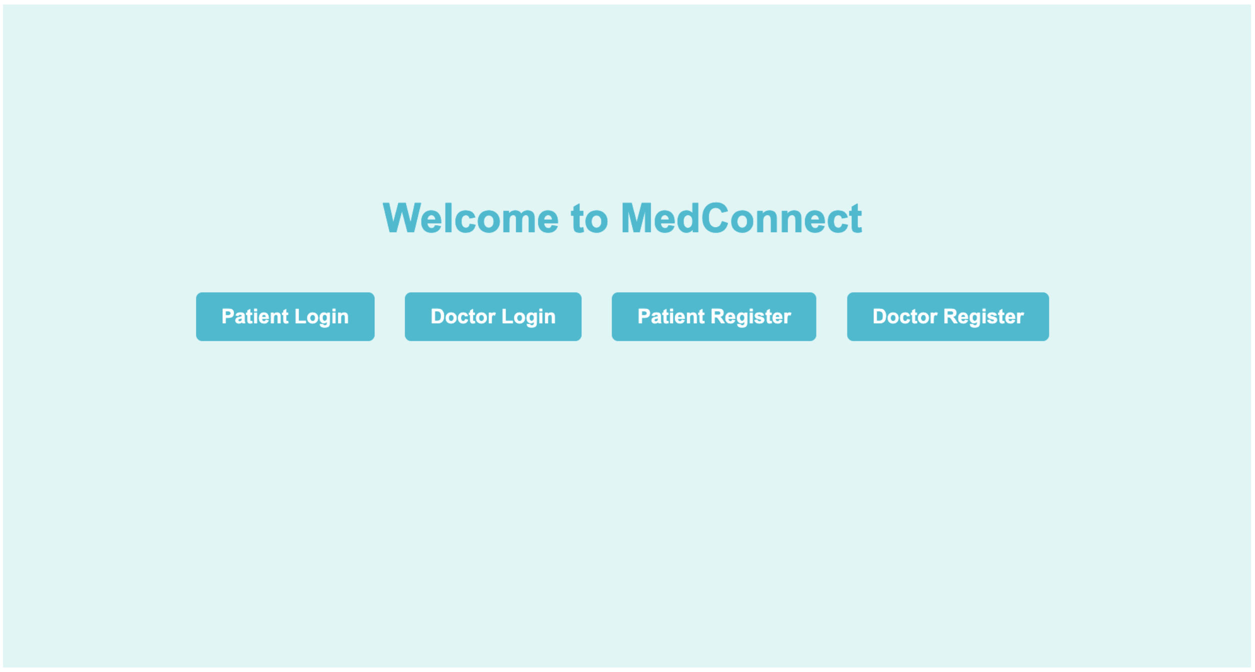
- This HTML code features a structured form that submits user details via POST to process-patient-register.php. The form includes input fields for full name, email, password, and password confirmation, each marked as required for basic validation. A register button allows users to submit their information, while a login link directs existing users to login.html. The page is styled using an external CSS file (register.css), with a centered layout inside a <div class="register-container"> and the form wrapped in <div class="register-box">.



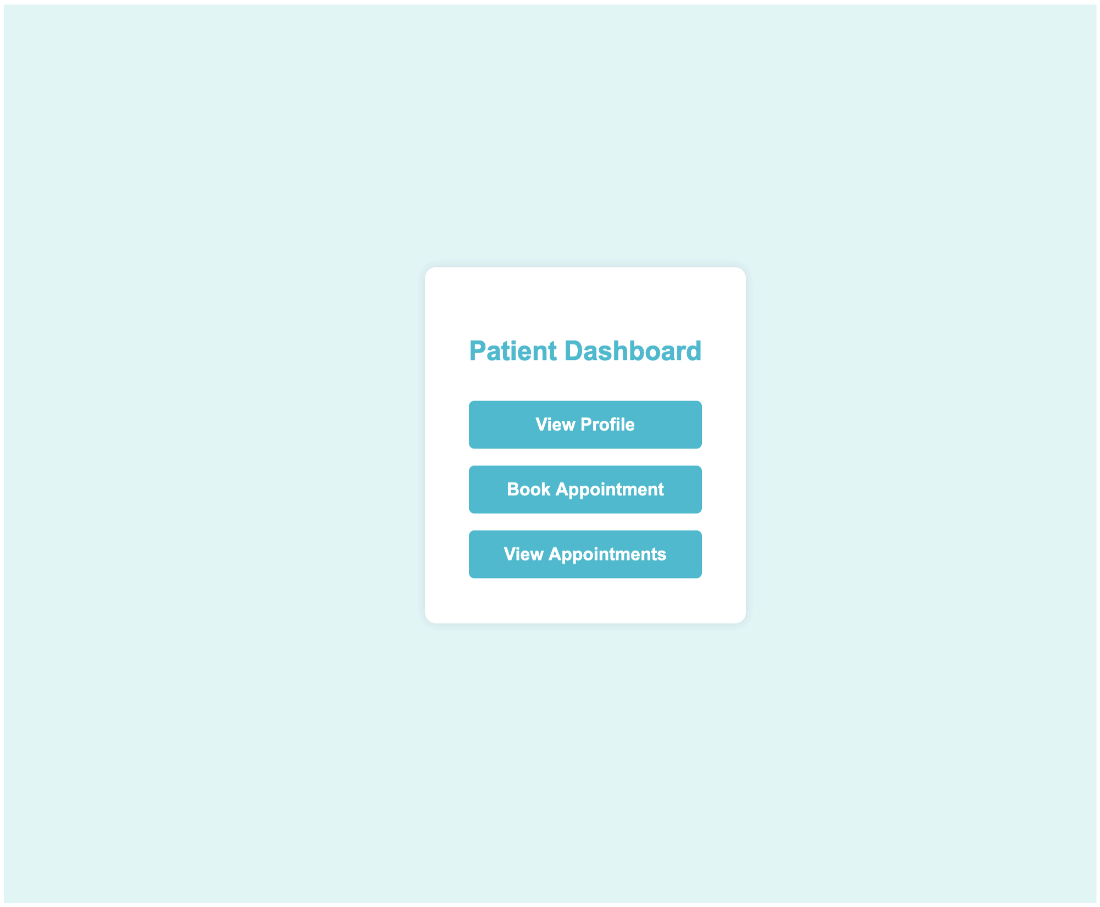
```php
<?php
//We will start the session to be able to store user data if they register properly.
session_start();

//Including the file that contains the database connection.
require_once 'database_connection.php';

//check that all required feilds are entered.
if (!isset($_POST['Email']) || !isset($_POST['Password'])) {
    echo "Enter both Email and password";
    exit;
}

// Get's inputs
$email = trim(string: $_POST['Email']);
$password = $_POST['Password'];

//using patient's email to identify them.
$stmt = $db->prepare("Select patient_id, name, password From patients where email = ?");

if(!$stmt){
    echo "Error in database" , $db->error;
    exit;

}

$stmt->bind_param("s", $email);
$stmt->execute();
$stmt->store_result();

//To check if an account with that email exists.
if ($stmt->num_rows === 1) {
    $stmt->bind_result($patient_id, $name, $hashed_password);
    $stmt->fetch();

// Verify the supplied password against the hashed one.
if (password_verify(password: $password, hash: $hashed_password)) {
    // Login successful: set session variables.
    $_SESSION['identity'] = 'patient';
    $_SESSION['name'] = $name;
    $_SESSION['new_id'] = $patient_id;

// Display welcome message and then redirect using JavaScript.
    echo "Login successful. Welcome, " . htmlspecialchars(string: $name) . "! You will be redirected shortly.";
    echo "<script>
            setTimeout(function(){
              window.location.href = 'dashboard-patient.html';
            }, 3000);
          </script>";
```

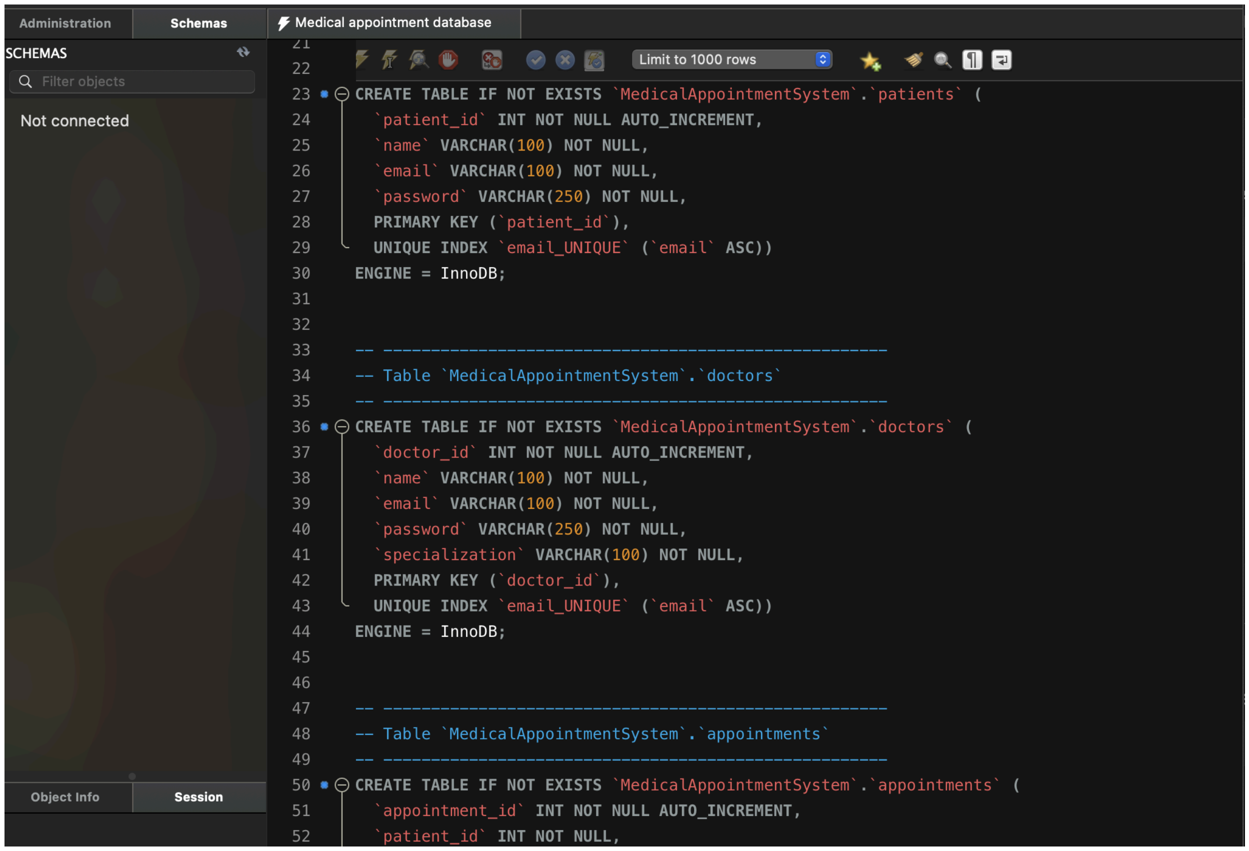
- This PHP script handles patient login for MedConnect by verifying user credentials against a database. It starts a session to store user data upon successful login and includes the database connection via require_once 'database_connection.php'. It checks if both email and password fields are provided before retrieving the user's details using a prepared statement to prevent SQL injection. The script queries the patients table for a matching email and, if found, verifies the hashed password using password_verify(). If authentication succeeds, it sets session variables (identity, name, new_id) and displays a welcome message before redirecting the user to dashboard-patient.html via JavaScript. If the email doesn't exist or the password is incorrect, appropriate error messages are shown.

- When running the code, it opens up the home page of our medical appointment system. The user can then click on either login or register once inside the server.

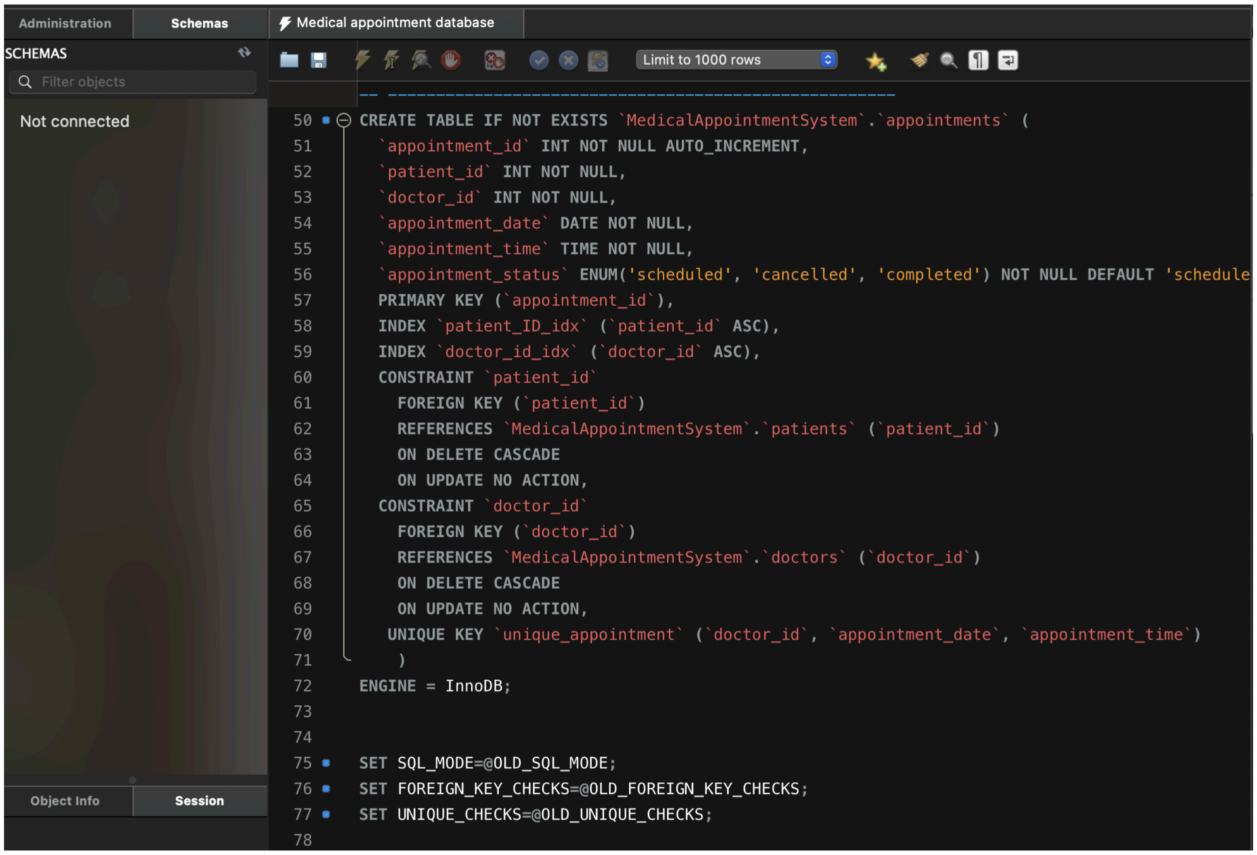


- Once the user can make an account, they can log in and come across this page. They can then select to view their profile, book an appointment or view any appointments they have booked.





```sql
CREATE TABLE IF NOT EXISTS `MedicalAppointmentSystem`.`patients` (
  `patient_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `password` VARCHAR(250) NOT NULL,
  PRIMARY KEY (`patient_id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `MedicalAppointmentSystem`.`doctors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MedicalAppointmentSystem`.`doctors` (
  `doctor_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `password` VARCHAR(250) NOT NULL,
  `specialization` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`doctor_id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `MedicalAppointmentSystem`.`appointments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MedicalAppointmentSystem`.`appointments` (
  `appointment_id` INT NOT NULL AUTO_INCREMENT,
  `patient_id` INT NOT NULL,
```



```sql
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MedicalAppointmentSystem`.`appointments` (
  `appointment_id` INT NOT NULL AUTO_INCREMENT,
  `patient_id` INT NOT NULL,
  `doctor_id` INT NOT NULL,
  `appointment_date` DATE NOT NULL,
  `appointment_time` TIME NOT NULL,
  `appointment_status` ENUM('scheduled', 'cancelled', 'completed') NOT NULL DEFAULT 'schedule
  PRIMARY KEY (`appointment_id`),
  INDEX `patient_ID_idx` (`patient_id` ASC),
  INDEX `doctor_id_idx` (`doctor_id` ASC),
  CONSTRAINT `patient_id`
    FOREIGN KEY (`patient_id`)
    REFERENCES `MedicalAppointmentSystem`.`patients` (`patient_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `doctor_id`
    FOREIGN KEY (`doctor_id`)
    REFERENCES `MedicalAppointmentSystem`.`doctors` (`doctor_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
   UNIQUE KEY `unique_appointment` (`doctor_id`, `appointment_date`, `appointment_time`)
    )
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

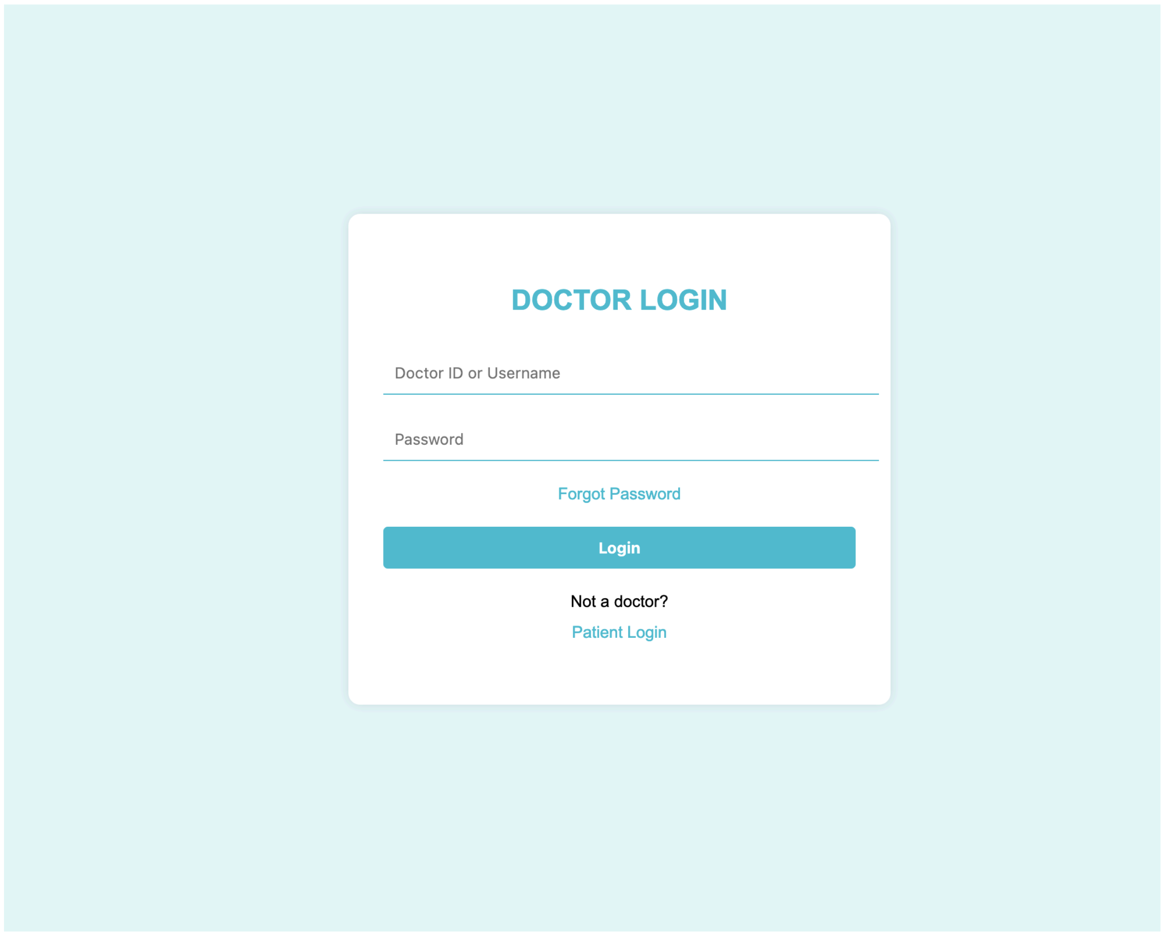
- The database schema goes as follows:
    1. Patients
        - Patient_id INT (primary key)
        - name VARCHAR(100)
        - email VARCHAR(100)
        - password VARCHAR(250)
    2. Doctors
        - doctor_id INT (primary key)
        - name VARCHAR(100)
        - email VARCHAR(100)
        - password VARCHAR(250)
        - specialization VARCHAR(100)
    3. Appointments
        - appointment_id INT (primary_key)
        - patient_id INT
        - doctor_id INT
        - appointment_date DATE
        - appointment_time TIME
        - appointment_status ENUM(‘scheduled, cancelled, completed)

Role-Based Access Control (RBAC) is a security model that restricts system access based on a user's role. In our medical appointment system, RBAC ensures that patients, doctors, and administrators can only perform actions appropriate to their roles, preventing unauthorized access to sensitive data.

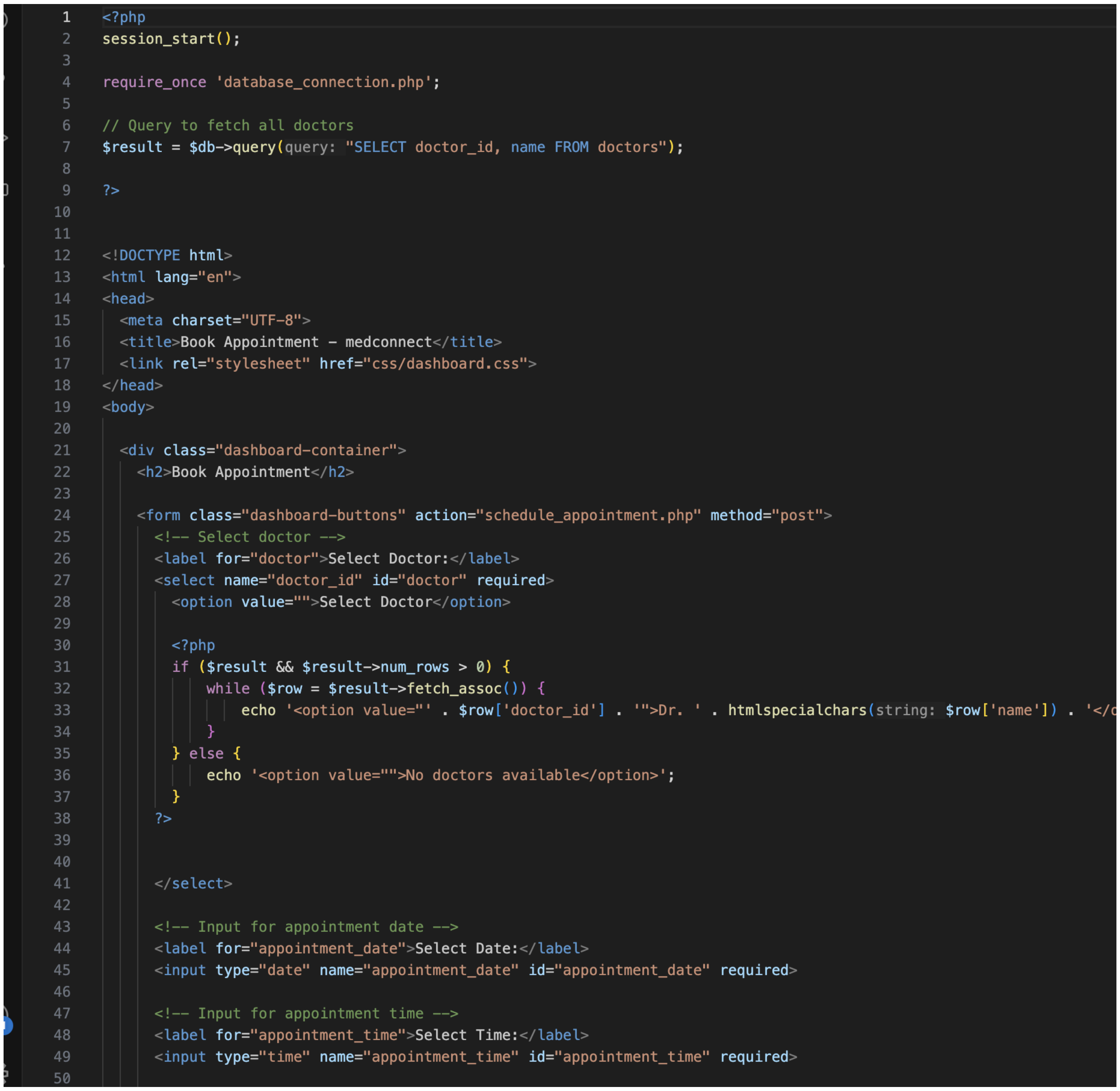
- Roles and Permissions in the System

1. **Patients**
    - Can **view available appointments**
    - Can **book, modify, and cancel their own appointments**
    - Cannot access other patients' data or modify doctor schedules
2. **Doctors**

- Can **view their own appointment schedules**
- Can **update patient visit statuses and notes**
- Cannot modify or access patient data unrelated to their own appointments



DOCTOR LOGIN



```php
<?php
session_start();

require_once 'database_connection.php';

// Query to fetch all doctors
$result = $db->query(query: "SELECT doctor_id, name FROM doctors");

?>


<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Book Appointment - medconnect</title>
  <link rel="stylesheet" href="css/dashboard.css">
</head>
<body>

  <div class="dashboard-container">
    <h2>Book Appointment</h2>

    <form class="dashboard-buttons" action="schedule_appointment.php" method="post">
      <!-- Select doctor -->
      <label for="doctor">Select Doctor:</label>
      <select name="doctor_id" id="doctor" required>
        <option value="">Select Doctor</option>

        <?php
        if ($result && $result->num_rows > 0) {
            while ($row = $result->fetch_assoc()) {
                echo '<option value="' . $row['doctor_id'] . '">Dr. ' . htmlspecialchars(string: $row['name']) . '</
            }
        } else {
            echo '<option value="">No doctors available</option>';
        }
      ?>


      </select>

      <!-- Input for appointment date -->
      <label for="appointment_date">Select Date:</label>
      <input type="date" name="appointment_date" id="appointment_date" required>

      <!-- Input for appointment time -->
      <label for="appointment_time">Select Time:</label>
      <input type="time" name="appointment_time" id="appointment_time" required>

```

- This PHP script creates a "Book Appointment" page for MedConnect, allowing patients to schedule medical appointments. It starts by initiating a session and connecting to the database, then fetches a list of available doctors from the doctors table using a SQL query. The form submits data via POST to schedule_appointment.php for processing. Basic validation ensures that a doctor, date, and time are selected before submission.

- User Guide

- The user guide is made briefly with a video that was made by our group members Jairo and Henrietta. It is a google drive that vividly shows the steps and process that the patient and/or doctor would take in order to use the web server and make/cancel apppointments. The instructions are as simple as registering an account, signing in, selecting the category to book appointments and the information upon it. As the doctor themselves would sign in and see any formulated appointments made by the patients. MedConnect_UserGuide

- Conclusion

- In conclusion, This project definitely had it's challenges and overall was a learning experience our group had to face. One of which was trying to figure out how to implement the PHP back-end into the web server but regardless we feel like we have succeeded in creating a website for the required mini-world scenario. This system enhances **patient accessibility** and **reduces scheduling conflicts.** However, future improvements could include **automated appointment reminders**, **telemedicine integration**, and **enhanced reporting capabilities** for better decision-making. Overall, this project demonstrated the importance of **team collaboration, problem-solving, and efficient database management** in building real-world web applications. The knowledge gained from this project will be valuable for future development in similar domains.