AUTOMATED CLINICAL LABORATORY REPORT GENERATION SYSTEM

Project report submitted to Alagappa University

in partial fulfillment of the requirement

for the award of the degree of

MASTER

OF

COMPUTER SCIENCE

Submitted by

N.S. SUDHAHARAN

Reg. No.: 2022551052

Under the guidance of

Prof. A. PADMAPRIYA, M.C.A., M. Phil., Ph. D.

DEPARTMENT OF COMPUTER SCIENCE

ALAGAPPA UNIVERSITY

(Accredited with 'A+' Grade by NAAC (CGPA: 3.64) in the Third Cycle and Graded as Category – I by MHRD-UGC)

KARAIKUDI 630 003

APRIL – 2024

CERTIFICATE

This is to certify that the project work titled **"AUTOMATED CLINICAL LABORATORY REPORT GENERATION SYSTEM"** is a bonafide work of **N.S. SUDHAHARAN (REG.NO:2022551052), M.Sc. Computer Science, Alagappa University** has been carried out under my supervision and guidance from December 2023 to April 2024.

PROFESSOR & HEAD **INTERNAL GUIDE**

Department of Computer Science

Alagappa University

Karaikudi

Submitted for viva-voce on …….

EXTERNAL EXAMINER **INTERNAL EXAMINER**

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Alagappa University, Karaikudi, for providing me with an opportunity to pursue my Master's degree in Computer Science. I am grateful to the Department of Computer Science for offering me a supportive and conducive environment to pursue my project.

I would like to thank the Vice-Chancellor and Registrar of Alagappa University for providing me with an opportunity to pursue my Master's degree in Computer Science and for creating a favorable academic environment to pursue project.

I extend my heartfelt thanks to my internal guide, Dr.A.Padmapriya Professor & Head, Department of Computer Science, for her constant guidance, encouragement, and valuable inputs throughout the project. Her guidance helped me to stay on track and motivated me to complete the project successfully.

Finally, I would also like to extend my gratitude to all the teaching and non-teaching staff of the Department of Computer Science for their support and cooperation throughout the project.

Thank you all for the support and encouragement.

N.S. SUDHAHARAN

(2022551052)

ABSTRACT

The "**Automated Clinical Laboratory Report Generation System**" is a transformative project designed to modernize and streamline medical laboratory operations through an innovative web-based platform. This system offers patients convenient features such as online appointment scheduling, real-time chat support with laboratory staff, and instant access to test reports for viewing, printing, or downloading. Comprehensive test instructions and information are available online, empowering patients to make informed decisions and prepare for tests. The platform enhances efficiency for administrators by automating appointment management and report retrieval processes, reducing manual intervention. By integrating cutting-edge web technologies and prioritizing user-centric design, the system aims to improve user experience, enhance data security, and position clinical laboratories for greater efficiency and adaptability.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
	ABSTRACT	
1	**INTRODUCTION**	**1**
	1.1Overview of the project	**1**
	1.2 About the department of computer science, Alagappa university	**2**
	1.3 Problem statement	**2**
	1.4. Objectives	**3**
2	**SYSTEM ENVIORNMENT**	**6**
	2.1 Hardware	**6**
	2.2 Software	**6**
	2.3 Front-end Technologies	**6**
	2.4 Back-end Technologies	**7**
	2.5 Other Tools and Libraries	**9**
	2.6. Project Setup	**9**
	2.7 Testing Environment	**9**
	2.8 Continuous Integration	**9**
3	**SYSTEM DESIGN**	**10**
	3.1 Architecture design	**10**
	3.2 Data Flow Diagram	**12**
	3.3 Module Explanation	**15**

4	**SYSTEM TESTING**	**18**
	4.1 Unit testing	**18**
	4.2 Integration testing	**18**
	4.3 System Testing	**18**
	4.4 User Acceptance Testing (Uat)	**19**
	4.5. Bug Tracking and Issue Resolution	**19**
5	**SYSTEM IMPLEMENTATION**	**20**
	5.1 Deployment Process	**20**
	5.2 Post-deployment Monitoring	**20**
	5.3 Feedback and Improvements	**20**
6	**CONCLUSION**	**22**
	6.1 Conclusion	**22**
	6.2 Future enhancement	**22**
	APPENDICES	**24**
	A. Source code	**24**
	B. Screenshots	**104**
	BIBLOGRAPHY	**115**

1. INTRODUCTION

1.1. OVERVIEW OF THE PROJECT

In recent years, the healthcare industry has experienced a significant shift toward digital transformation, particularly in the realm of medical laboratory services. As the demand for efficient and streamlined processes continues to grow, the development of modern web applications tailored to clinical laboratory management has become increasingly important. The "Automated Clinical Laboratory Report Generation System " is a comprehensive solution designed to address the challenges faced by traditional paper-based systems and to leverage the latest advancements in technology.

The traditional approach to clinical laboratory management involves manual appointment scheduling, paper-based report retrieval, and limited accessibility for users seeking information and support. These methods are often time-consuming, inefficient, and prone to human error. As a result, there is a pressing need for an automated system that enhances user experience and simplifies operations for both patients and laboratory staff.

The proposed system aims to modernize clinical laboratory management by providing an intuitive online platform for users to schedule appointments, access test reports, and communicate with administrators in real-time. By integrating cutting-edge web technologies such as Angular 17, Node.js, JavaScript, and MySQL, the system offers a seamless and efficient user experience. Additionally, the inclusion of comprehensive test instructions and online support empowers users to make informed decisions and improve their overall satisfaction.

In summary, the "Automated Clinical Laboratory Report Generation System " seeks to revolutionize the way clinical laboratory services are delivered. By automating manual processes and providing user-centric features, the system aspires to become a valuable tool for medical professionals and patients alike. This project represents a forward-thinking approach to healthcare management and sets the stage for a more efficient, secure, and accessible clinical laboratory experience.

1.2. ABOUT THE DEPARTMENT OF COMPUTER SCIENCE, ALAGAPPA UNIVERSITY

Department of Computer Science, Alagappa University is a leading academic institution in the field of computer science research and education. The department has a team of experienced faculty members who are passionate about advancing the field of computer science through innovative research and technology. The department has expertise in various areas of computer science, including machine learning, data science, artificial intelligence, and computer vision.

In this project, the Alagappa University Department of Computer Science aims to develop a Web Application to streamline and automate the process of generating reports for clinical laboratory tests. This system is designed to facilitate the efficient management of patient data and laboratory results within a clinical setting.

1.3. PROBLEM STATEMENT

The current methods of managing clinical laboratory services are largely reliant on traditional paper-based systems, which present numerous challenges in terms of efficiency, accuracy, and accessibility. Key issues with these outdated systems include:

1.3.1. Inefficiency in Scheduling:

The manual process of scheduling appointments for laboratory tests and consultations is time-consuming and prone to errors, leading to delays and potential missed appointments for patients.

1.3.2. Lack of Accessibility and Convenience:

Patients often face difficulty in obtaining test results and reports in a timely manner due to the reliance on physical paperwork and in-person visits. This creates unnecessary inconvenience and delays in medical decision-making.

1.3.3. Limited Communication Channels:

Traditional systems offer limited avenues for communication between patients and laboratory staff, resulting in inadequate support and potential misunderstandings regarding tests and procedures.

1.3.4. Increased Manual Work:

The reliance on manual processes for appointment scheduling, report retrieval, and other administrative tasks leads to an increased workload for laboratory staff, reducing their efficiency and productivity.

1.3.5. Inadequate Information for Patients:

Patients often lack easy access to comprehensive information about tests, procedures, and preparation, which can lead to confusion and poor adherence to medical instructions.

Given these challenges, there is a clear need for a modern, automated solution that leverages digital technology to streamline clinical laboratory processes, enhance user experience, and improve overall efficiency for both patients and laboratory staff. This is the goal of the "Automated Clinical Laboratory Report Generation System." By addressing these issues, the proposed system aims to revolutionize clinical laboratory management and provide a more efficient, accessible, and user-centric service.

1.4. OBJECTIVES

The "Automated Clinical Laboratory Report Generation System " aims to revolutionize the management of clinical laboratory services by streamlining processes and enhancing user experience. The specific objectives of the system include:

1.4.1. Automated Appointment Scheduling:

Provide an intuitive and efficient online platform for patients to schedule appointments for laboratory tests and consultations, thereby reducing manual scheduling and potential errors.

1.4.2. Facilitate Easy Access to Test Reports:

Enable users to retrieve, view, print, and download their test reports online, providing them with immediate access to their medical information and eliminating the need for physical visits to the laboratory.

1.4.3. Offer Real-Time Communication:

Incorporate a real-time chat support feature that allows users to communicate directly with administrators, addressing inquiries and concerns promptly to improve user satisfaction and engagement.

1.4.4. Provide Comprehensive Test Information:

Offer detailed instructions and information about tests, procedures, and preparation to patients online, empowering them to make informed decisions and comply with medical instructions more effectively.

1.4.5. Enhance User Experience:

Create a user-friendly and interactive web application that simplifies interactions with the laboratory, improves user satisfaction, and increases the efficiency of service delivery.

1.4.6. Streamline Administrative Processes:

Reduce manual intervention in laboratory management by automating administrative tasks such as appointment scheduling and report retrieval, thereby freeing up laboratory staff to focus on more complex tasks.

1.4.7. Ensure Security and Privacy:

Implement robust security measures to protect patient data and ensure compliance with relevant regulations, providing users with peace of mind when accessing their medical information online.

1.4.8. Increase Laboratory Efficiency:

Improve overall laboratory operations by integrating advanced web technologies and reducing the burden of manual tasks, enabling more efficient and effective service delivery.

By achieving these objectives, the "Automated Clinical Laboratory Report Generation System " seeks to provide an innovative and effective solution to the challenges faced by traditional clinical laboratory management methods, ultimately leading to improved healthcare outcomes and patient satisfaction.

2. SYSTEM ENVIRONMENT

2.1. HARDWARE

Processor: Intel Dual Core or higher.

RAM: 2GB or more.

Storage: 90GB or more.

Printer: HP Ink Jet (if needed for printing reports).

2.2. SOFTWARE

Operating System: Windows 10, or a later version, was used for development, although the project can be run on other platforms such as macOS or Linux.

IDE: Visual Studio Code was the preferred integrated development environment for code editing and debugging.

2.3. FRONT-END TECHNOLOGIES

2.3.1. Framework:

Angular 17: The front-end of your application was built using Angular 17, a popular JavaScript framework for creating dynamic, modern, and responsive web applications. Angular offers features such as two-way data binding, component-based architecture, observables for handling asynchronous data, and more.

2.3.2. Languages:

HTML: Used for structuring the application's layout and defining the content of the user interface.

TypeScript: The main programming language used in the Angular application, providing type safety, error checking, and modern language features.

CSS: Used for styling the application and ensuring a visually appealing and user-friendly interface.

2.3.3. Libraries and Tools:

Observables: Utilized in Angular for handling asynchronous data operations, such as fetching data from the server and updating the user interface.

Angular Material: A design library offering pre-built, customizable UI components following the Material Design guidelines, used to create a consistent and polished user experience.

Bootstrap: Used as a CSS framework to enhance the styling and responsiveness of the application.

2.3.4. Testing and Debugging:

Jasmine and Karma: Used for writing and running unit tests for Angular components and services.

2.3.5. Additional Tools:

Browser Developer Tools: Used for debugging, inspecting elements, and optimizing performance during development.

2.4. BACK-END TECHNOLOGIES

2.4.1. Runtime Environment:

Node.js: Used as the runtime environment for executing JavaScript on the server-side, enabling the development of scalable and efficient back-end systems.

2.4.2. Framework:

Express.js: A lightweight and flexible framework built on top of Node.js, used for creating RESTful APIs, handling routing, and managing server-side logic.

2.4.3. Database:

MySQL: A relational database management system (RDBMS) used to store and manage application data. MySQL was chosen for its reliability, scalability, and widespread support.

2.4.4. Languages:

JavaScript: The main language used for server-side programming and API development in Node.js.

PHP: Used as needed for specific back-end development tasks, particularly for working with MySQL.

2.4.5. Data Management:

RESTful APIs: APIs were created using Express.js to handle communication between the front-end and back-end, including requests and responses for data management.

MySQL Queries: Custom queries were written to interact with the database, perform CRUD operations, and optimize data retrieval.

2.4.6. Testing and Debugging:

Postman: Used for testing and debugging APIs to ensure they return expected data and handle different request scenarios.

2.4.7. Additional Tools:

Git: Used for version control, managing code changes, and collaborating with other developers.

Json-Server: Used for mocking data during development and testing, providing a temporary data source.

2.5. OTHER TOOLS AND LIBRARIES

Package Management: NPM was used for managing project dependencies for both front-end and back-end.

Git: Used for version control, ensuring consistent and traceable development changes.

Json-Server: Used for creating a mock server for testing and development purposes.

Angular Material: Used for incorporating material design components in the front-end.

2.6. PROJECT SETUP

Repository Setup: The project was managed using Git repositories for version control and collaboration.

Dependencies: Project dependencies were managed using npm for back-end and front-end components.

2.7. TESTING ENVIRONMENT

Testing Tools: Unit tests were conducted using testing frameworks such as Jasmine and Karma (for Angular).

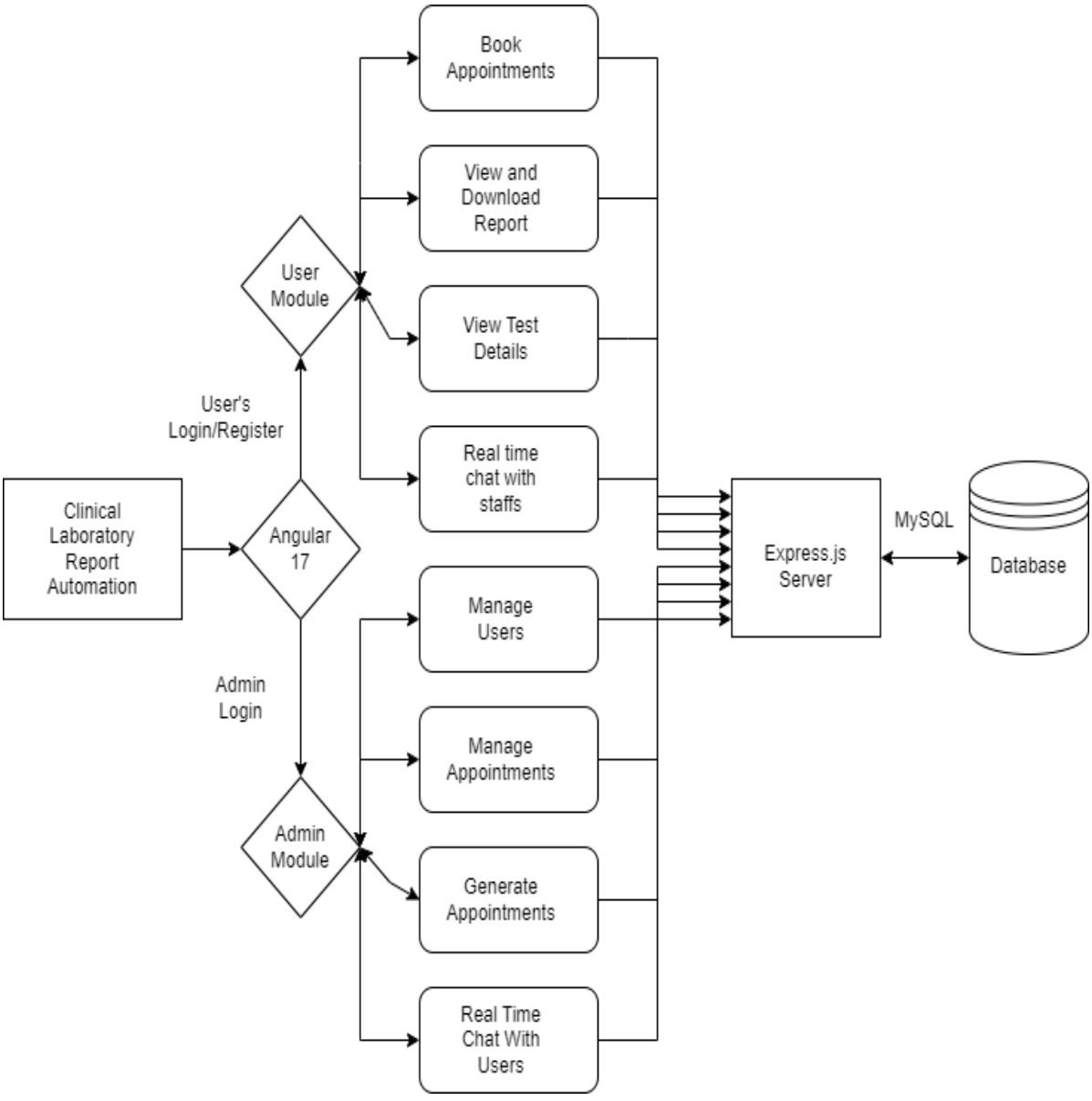
Test Data: Test data was managed and stored using mock servers and databases.

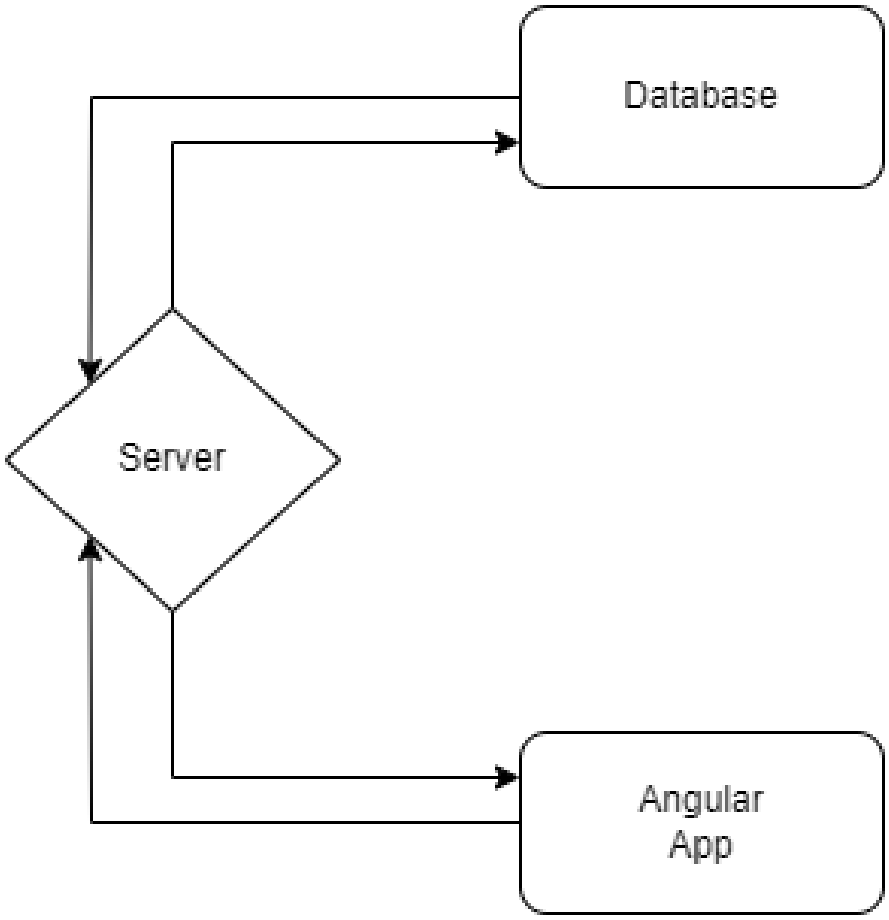
2.8. CONTINUOUS INTEGRATION

Integration Tools: Continuous integration (CI) tools such as GitHub Actions or Jenkins can be set up for automated testing and deployment if needed.

3. SYSTEM DESIGN

3.1 ARCHITECTURAL DESIGN



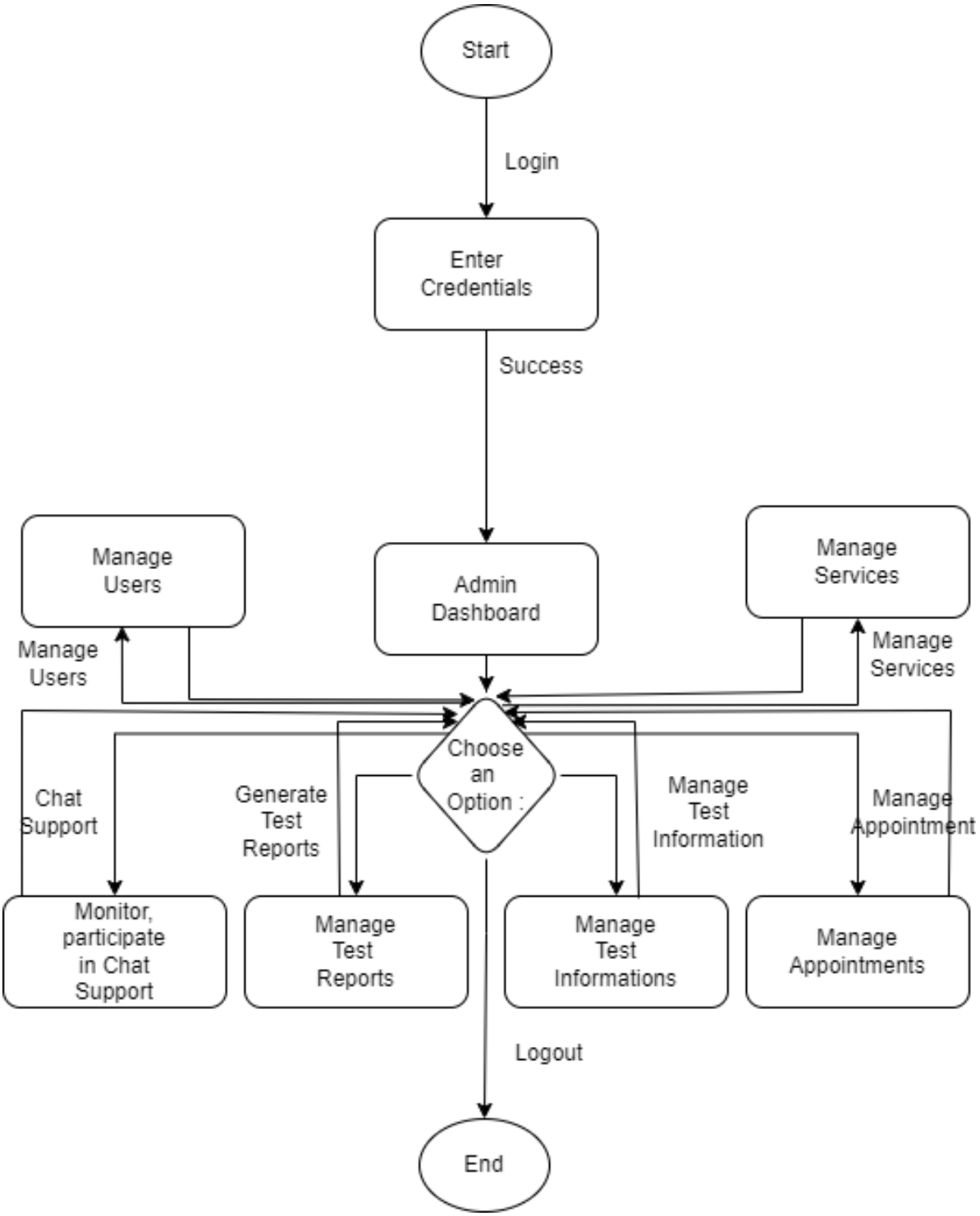

Database
Server
Angular
App

3.2. DATA FLOW DIAGRAM

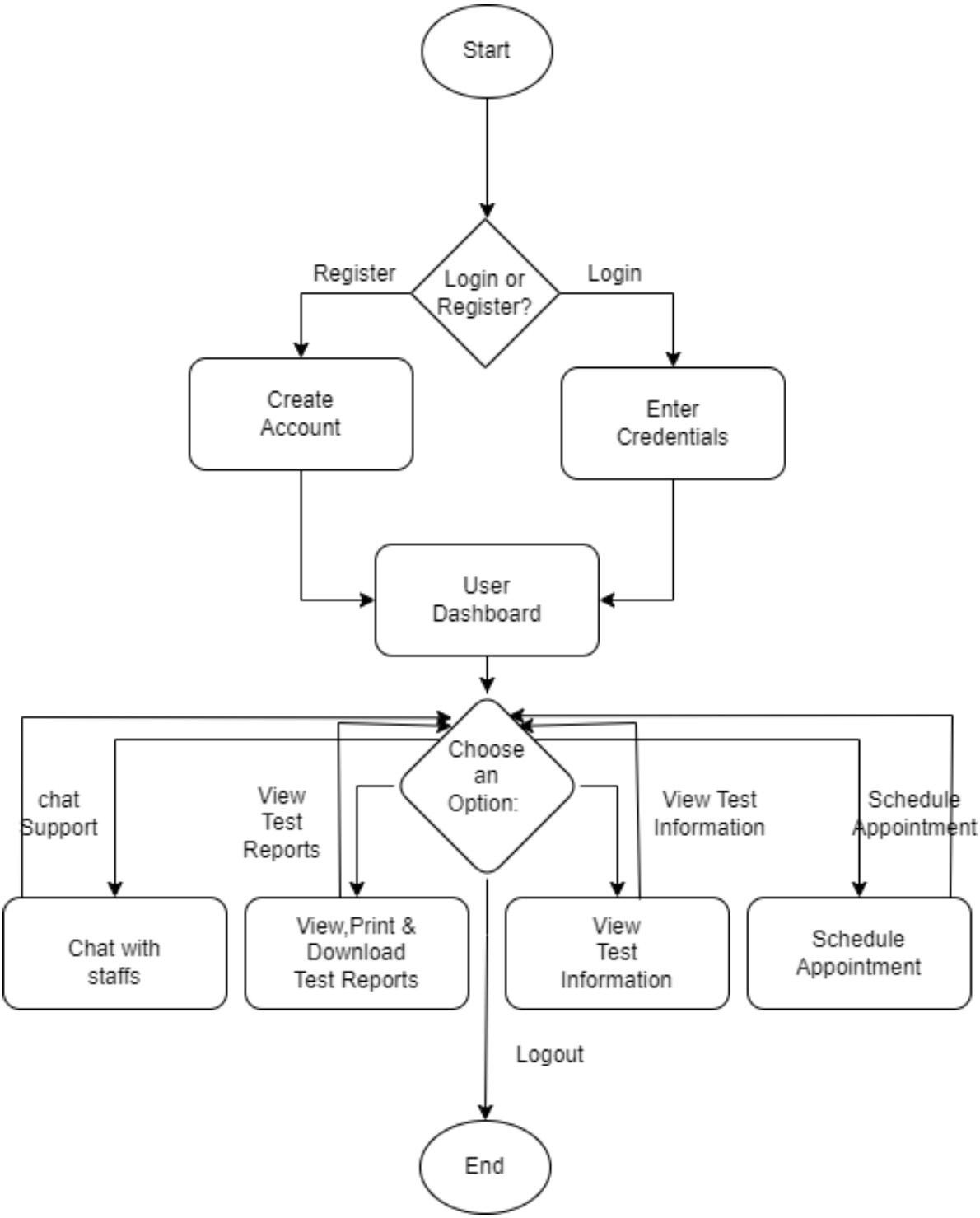
A data flow diagram (DFD) is a visual representation that illustrates how data moves through a system, showing the inputs, outputs, data storage, and processes involved. For the "Automated Clinical Laboratory Report Generation System," a DFD for the admin module will outline how data flows within the system as the administrator performs various tasks.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

3.2.1. Data Flow Diagram for Admin Module



3.2.2. Data Flow Diagram for User Module



3.3. MODULE EXPLANATION

3.3.1. User Module

The user module is responsible for managing user interactions with the application. It provides various features that enable users to access services, book appointments, and manage their accounts. Below are the key functionalities of the user module:

Login:

Users can log in using their credentials to access their account and use the services offered by the application.

Register:

New users can create an account by providing necessary details such as name, email, password, and other required information.

Book Appointments:

Users can schedule appointments for medical tests and consultations through an online booking system.

View Booked Appointments:

Users can view the details of their booked appointments, including date, time, and location.

View Services Offered:

Users can browse the list of services offered by the clinical laboratory, including various tests and medical procedures.

View and Download Reports:

Users can access their test reports online, view the results, and download them for reference.

View Test Details:

Users can view detailed information about the tests they have booked, including preparation instructions and estimated timelines.

Real-Time Chat with Admin:

Users can communicate with an administrator through a real-time chat feature to ask questions or get assistance.

3.3.2. Admin Module

The admin module is responsible for managing and overseeing the operations of the application from the administrator's perspective. This module offers various features to monitor and control the application, including:

Manage User Accounts:

Admins can manage user accounts, including creating, updating, or deleting accounts as needed.

Manage Appointments:

Admins can oversee scheduled appointments, including confirming or modifying appointments based on availability.

Manage Test Reports:

Admins can oversee the creation, storage, and delivery of test reports to users.

Manage Real-Time Chat:

Admins can engage in real-time chat with users, addressing their inquiries and providing support.

Manage Services Offered:

Admins can add, modify, or remove services offered by the clinical laboratory, such as tests and consultations.

Monitor User Activity:

Admins can monitor user activities within the application, ensuring compliance with policies and procedures.

4. SYSTEM TESTING

4.1. UNIT TESTING

Objective: Verify individual units (components, services, and functions) for correctness and reliability.

Tools: Jasmine and Karma were used for Angular unit testing.

Coverage: All critical parts of the application, including front-end and back-end, were covered by unit tests.

4.2. INTEGRATION TESTING

Objective: Ensure that different components and modules work together seamlessly.

Method: Integration tests were conducted to validate the interaction between the front-end and back-end.

Approach: API testing was performed using tools like Postman to verify data flow between Angular and Express.js.

4.3. SYSTEM TESTING

Objective: Validate the complete system to ensure it meets the specified requirements.

Scope: The entire application was tested for performance, functionality, security, and compatibility.

Approach: Comprehensive end-to-end tests were performed using manual testing methods and automated test scripts.

4.4. USER ACCEPTANCE TESTING (UAT)

Objective: Obtain feedback from end users to verify that the system meets their expectations.

Process: Test scenarios were provided to users, and their feedback was collected and analyzed.

Results: Users verified key functionalities such as appointment booking, viewing reports, and chatting with the admin.

4.5. BUG TRACKING AND ISSUE RESOLUTION

Bug Tracking Tool: A bug tracking tool such as GitHub Issues or JIRA was used to track and manage issues and bugs found during testing.

Resolution: Identified issues were promptly addressed, with fixes deployed for further testing.

5. SYSTEM IMPLEMENTATION

5.1. DEPLOYMENT PROCESS

Preparation:

Environment Setup: The production environment was set up with all necessary software and hardware requirements, such as Node.js, MySQL, and the web server.

Configuration: Configuration files were updated with production settings (e.g., database connection strings).

Build and Deployment

Build: The application was built using the Angular CLI, creating an optimized version of the front-end code.

Deployment: The back end (Node.js and Express.js) and front-end (Angular) were deployed to the production server.

Database Initialization

Schema Setup: The MySQL database schema was set up, including tables and indexes.

Data Migration: Necessary data was migrated from development/testing environments to production.

5.2. POST-DEPLOYMENT MONITORING

Monitoring Tools: Tools such as New Relic or Google Analytics were used to monitor application performance and user behavior.

Error Handling: Log files were monitored for any errors, and alerts were set up for immediate issue identification.

5.3. FEEDBACK AND IMPROVEMENTS

Feedback Collection: User feedback was collected and analyzed to identify areas for improvement.

Continuous Integration and Continuous Deployment (CI/CD): CI/CD practices were followed for regular updates and maintenance.

6. CONCLUSION

6.1. CONCLUSION

The "Automated Clinical Laboratory Report Generation System " has successfully achieved its goal of modernizing and streamlining medical laboratory processes through an innovative web application. The system allows users to schedule appointments, access reports, chat with administrators, and manage their profiles seamlessly. Meanwhile, administrators can efficiently oversee user activities, manage appointments, and maintain the application.

The use of Angular 17 for front-end development, paired with Node.js, Express.js, and MySQL for back-end operations, provided a robust foundation for the system. The observables in Angular facilitated smooth handling of asynchronous data, while the use of PHP and JSON server supported various backend functionalities.

Overall, the project has achieved its objectives, offering a user-friendly interface and efficient back-end systems that improve the clinical laboratory's operations. The system's implementation demonstrates how web technologies can be used to create an advanced, interactive, and efficient laboratory management system.

6.2. FUTURE ENHANCEMENTS

While the "Automated Clinical Laboratory Report Generation System " has been a successful project, there are several opportunities for future enhancements and improvements:

Enhanced Security:

Strengthen authentication mechanisms using multi-factor authentication for added security.

Implement encryption for sensitive data such as test reports and personal information.

Mobile Application:

Develop a dedicated mobile application for Android and iOS platforms to enhance accessibility and user convenience.

Advanced Analytics:

Integrate data analytics tools to provide insights into laboratory operations, such as test trends, appointment scheduling patterns, and user behavior.

AI and Machine Learning:

Utilize AI and machine learning algorithms to provide personalized test recommendations and improve report analysis.

Automated Notifications:

Enhance user experience with automated notifications for appointment reminders, report availability, and chat responses.

Integration with Third-Party Services:

Integrate with other healthcare systems and third-party services such as telemedicine platforms for a more comprehensive healthcare solution.

Enhanced Reporting Features:

Improve reporting features with additional visualizations, trend analysis, and health metrics tracking.

Feedback Mechanism:

Add a user feedback system to gather suggestions and complaints, aiding continuous improvement.

Internationalization and Localization:

Expand the system to support multiple languages and regional variations for broader reach and accessibility.

APPENDICES

A. SOURCE CODE

App.component.html

```
<ng-container *ngIf="userService.isAdmin(); else regularHeader">
  <app-adminheader></app-adminheader>  </ng-container>
    <ng-template #regularHeader>
    <app-header></app-header>
  </ng-template>
   <router-outlet></router-outlet>
```

App.routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './dashboard/home/home.component';
import { ServicesComponent } from './dashboard/services/services.component';
import { ReportComponent } from './dashboard/report/report.component';
import { AboutComponent } from './dashboard/about/about.component';
import { AppointmentComponent } from
'./dashboard/appointment/appointment.component';
import { ContactComponent } from './dashboard/contact/contact.component';
import { LoginComponent } from './components/pages/login/login.component';
import { RegisterComponent } from './components/pages/register/register.component';
import { AdminusersComponent } from './admin/adminusers/adminusers.component';
```

```
import { AdminAppointmentComponent } from './admin/admin-appointment/admin-appointment.component';
import { TestInfoComponent } from './admin/test-info/test-info.component';
import { TestDetailsComponent } from './test-details/test-details.component';
import { LabComponent } from './dashboard/lab/lab.component';
import { AdminLabComponent } from './admin/admin-lab/admin-lab.component';
import { ReferenceComponent } from './admin/reference/reference.component';
import { UserReportComponent } from './admin/user-report/user-report.component';
import { AdminAuthGuard } from './auth/guards/admin.guard';
import { ChangepasswordComponent } from './changepassword/changepassword.component';
import { UserGuard } from './user.guard';
import { ChatComponent } from './dashboard/chat/chat.component';
import { UserChatComponent } from './dashboard/user-chat/user-chat.component';
const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'register', component: RegisterComponent },
  { path: 'login', component: LoginComponent },
  { path: 'changepassword', component: ChangepasswordComponent },
  { path: 'home', component: HomeComponent },
  { path: 'chat', component: ChatComponent,canActivate: [AdminAuthGuard] },
  { path: 'userChat', component: UserChatComponent,canActivate: [UserGuard] },
```

```
  { path: 'appointment', component: AppointmentComponent, canActivate:
[UserGuard] },

  { path: 'services', component: ServicesComponent, canActivate: [UserGuard] },

  { path: 'report', component: ReportComponent, canActivate: [UserGuard] },

  { path: 'contact', component: ContactComponent, canActivate: [UserGuard] },

  { path: 'about', component: AboutComponent, canActivate: [UserGuard] },

  { path: 'adminusers', component: AdminusersComponent,canActivate:
[AdminAuthGuard] },

  { path: 'adminappointments', component: AdminAppointmentComponent,canActivate:
[AdminAuthGuard] },

  { path: 'testInfo', component: TestInfoComponent,canActivate: [AdminAuthGuard] },

  { path: 'testdetails', component: TestDetailsComponent, canActivate: [UserGuard] },

  { path: 'lab', component: LabComponent, canActivate: [UserGuard] },

  { path: 'adminlab', component: AdminLabComponent,canActivate:
[AdminAuthGuard] },

  { path: 'reference', component: ReferenceComponent,canActivate:
[AdminAuthGuard] },

  { path: 'userReport/:userId', component: UserReportComponent,canActivate:
[AdminAuthGuard] },

  { path: '**', redirectTo: 'home', pathMatch: 'full' },

];

@NgModule({

  imports: [RouterModule.forRoot(routes,{ scrollPositionRestoration: 'enabled' })],

  exports: [RouterModule]
```

})export class AppRoutingModule { }

Register.component.html

```
<section class="dark">
  <div class="signin">
    <div class="content">
      <form [formGroup]="registerForm" (ngSubmit)="submit()">
        <h2>Register</h2>
        <div class="form">
          <div class="inputBox">
            <input type="text" formControlName="name" required> <i>Name</i>
          </div>
          <div class="inputBox">
            <input type="text" formControlName="email" required> <i>Email</i>
          </div>
          <div class="inputBox">
            <input type="password" formControlName="password" required>
<i>Password</i>
          </div>
          <div class="inputBox">
            <input type="password" formControlName="confirmPassword" required>
<i>Confirm Password</i>
          </div>
          <div class="inputBox">
```

```
  <input type="text" formControlName="address" required> <i>Address</i>
</div>
<div class="links">
  <a routerLink="/login">Already have an account? Login</a>
</div>
<div class="inputBox">
  <input type="submit" value="Register">
</div>        </div>        </form>     </div>    </div>  </section>
```

Register.component.ts

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { UserService } from '../../../services/user.service';
import { ActivatedRoute, Router } from '@angular/router';
@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent {
  registerForm!: FormGroup;
  isSubmitted = false;
  returnUrl = '';
  constructor(
```

```
  private formBuilder: FormBuilder,
  private userService: UserService,
  private activatedRoute: ActivatedRoute,
  private router: Router
) {}
ngOnInit(): void {
  this.registerForm = this.formBuilder.group({
    name: ['', Validators.required],
    email: ['', [Validators.required, Validators.email]],
    password: ['', Validators.required],
    confirmPassword: ['', Validators.required],
    address: ['', Validators.required]
  });
  // get return url from route parameters or default to '/'
  this.returnUrl = this.activatedRoute.snapshot.queryParams['returnUrl'];
}
get fc() {
  return this.registerForm.controls;
}
submit() {
  this.isSubmitted = true;
  if (this.registerForm.invalid) return;
  this.userService.register({
```

```
    name: this.fc['name'].value,
    email: this.fc['email'].value,
    password: this.fc['password'].value,
    confirmPassword: this.fc['confirmPassword'].value,
    address: this.fc['address'].value
  }).subscribe(() => {
    this.router.navigateByUrl('/login');
  });  }}
```

Login.component.html

```
<section class="dark">
  <div class="signin">
    <div class="content">
      <form [formGroup]="loginForm" (ngSubmit)="submit()">
        <h2>Sign In</h2>
        <div class="form">
          <div class="inputBox">
            <input type="text" formControlName="email" required> <i>Email</i>
          </div>
          <div class="error-list" *ngIf="fc['email'].errors && isSubmitted">
           <div *ngIf="fc['email'].hasError('required')">Email shouldn't be empty !</div>
           <div *ngIf="fc['email'].hasError('email')">Incorrect Email !</div>
          </div>
          <div class="inputBox">
```

```
          <input type="password" formControlName="password" required>
<i>Password</i>
        </div>
        <div class="error-list" *ngIf="fc['email'].errors && isSubmitted">
          <div *ngIf="fc['password'].hasError('required')">password shouldn't be
empty !</div>
        </div>
        <div class="links">
          <a routerLink="#">Forgot Password</a>
          <a routerLink="/register">Signup</a>
        </div>
        <div class="inputBox">
          <input type="submit" value="Login">
        </div>
      </div>
    </form>
  </div>
 </div>
</section>
```

Login.component.ts

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { UserService } from '../../../services/user.service';
```

```
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.css'
})export class LoginComponent {
loginForm!: FormGroup;
isSubmitted = false;
returnUrl = '';
constructor(private FormBuilder: FormBuilder,private UserService: UserService,private activateRoute:ActivatedRoute,private router:Router){}
ngOnInit(): void{
  this.loginForm = this.FormBuilder.group({
    email:['',[Validators.required,Validators.email]],
    password:['',Validators.required]
  });
  // get return url from route parameters or default to '/'
  this.returnUrl = this.activateRoute.snapshot.queryParams['returnUrl'];
}get fc(){
  return this.loginForm.controls;
}submit() {
  this.isSubmitted = true;
  if (this.loginForm.invalid) return;
```

```
  this.UserService.login({email: this.fc['email'].value, password: this.fc['password'].value}).subscribe(() => {
    // Check if the user is an admin
    const isAdmin = this.UserService.isAdmin(); // Assuming you have a method in UserService to check if the user is admin
    console.log('Is admin:', isAdmin); // Debugging statement
    // Redirect based on user role
    if (isAdmin) {
      console.log('Redirecting to admin dashboard...'); // Debugging statement
      this.router.navigateByUrl('/adminhome');
    } else {
      console.log('Redirecting to home page...'); // Debugging statement
      // Redirect to user dashboard or any other default route
      this.router.navigateByUrl(this.returnUrl);
    }
  }, error => {
    console.error('Login error:', error); // Log any login errors for debugging
  });}}
```

Home.component.html

```
<div class="banner">
    <div class="container">
        <div class="banner-content">
            <div class="banner-left">
```

```
            <div class="content-wrapper">
                <h1 class="banner-title">Virtual healthcare <br> for you</h1>
                <p class="text text-white">Medimoor provides progressive, and affordable healthcare, accessible on mobile and online for everyone</p>
                <a href="/appointment" class="btn btn-secondary">Consult today</a>
            </div>          </div>       </div>    </div>    <div class="banner-image">
        <img [src]="imageUrl" alt="Example Image">
    </div>
</div>
<app-services></app-services>
<app-about></app-about>
```

Header.component.html

```
<header>
    <nav>
        <div class="wrapper">
            <div class="logo"><a routerLink="/home">CLINIC LAB</a></div>
            <ul class="nav-links">
                <label for="close-btn" class="btn close-btn"><i class="fas fa-times"></i></label>
                <li><a routerLink="/home">Home</a></li>
                <li><a routerLink="/appointment">Appointment</a></li>
                <li><a routerLink="/lab">LabAppointment</a></li>
                <li><a routerLink="/report">Report</a></li>
```

```
<li><a routerLink="/testdetails">Test</a></li>
<li><a routerLink="/services">services</a></li>
<li><a routerLink="/userChat">Chat</a></li>
<li><a routerLink="/about">About</a></li>
<li *ngIf="!isAuth"><a routerLink="/login" class="desktop-item">Login</a></li>
<li *ngIf="isAuth">
  <a href="javascript:void(0);" class="desktop-item">Profile</a>
  <ul class="drop-menu">
    <li><a>{{user.name}}</a></li>
    <li>
      <a routerLink="changepassword">change Password</a>
    </li>
    <li><a (click)="logout()">Logout</a></li>
  </ul>
</li>
</ul>
</div>
</nav></header>
```

Header.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../../../services/user.service';
import { User } from '../../../shared/model/user.model';
import { Router } from '@angular/router';
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
```

```
  styleUrls: ['./header.component.css'],
})export class HeaderComponent implements OnInit {
  user: User = new User(); // Initialize with default values
  constructor(private userService: UserService,private Router:Router) {}
  ngOnInit(): void {
    // Subscribe to userObservable to update user data
    this.userService.userObservable.subscribe((newUser) => {
      this.user = newUser;
    });
  }  logout() {
    this.userService.logOut();
     }
  get isAuth() {
    return !!this.user.token; // Check if token exists
  }}
```

Services.componnent.html

```
   <div class="container">
    <h2>Services</h2>
        <!-- Form to add a new service -->
    <form (submit)="addService()" *ngIf="userService.isAdmin()">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" [(ngModel)]="newService.name"
required><br>
```

```
      <label for="description">Description:</label>
      <textarea id="description" name="description"
[(ngModel)]="newService.description" required></textarea><br>
      <label for="price">Price:</label>
      <input type="number" id="price" name="price" [(ngModel)]="newService.price"
required><br>
      <button type="submit">Add Service</button>
    </form>       <!-- List of services -->
    <div *ngFor="let service of services">
      <div class="service">
        <h3>{{ service.name }}</h3>
        <p><strong>Description:</strong> {{ service.description }}</p>
        <p><strong>Price:</strong> {{ service.price }}</p>
          <!-- Update button -->
        <button (click)="showUpdateForm(service)"
*ngIf="userService.isAdmin()">Update</button>
           <!-- Delete button -->
        <button (click)="deleteService(service.id)"
*ngIf="userService.isAdmin()">Delete</button>
        <div *ngIf="service === selectedService">
          <h3>Update Service</h3>
          <form (submit)="updateService(selectedService)">
            <label for="updateName">Name:</label>
```

```
        <input type="text" id="updateName" name="updateName"
[(ngModel)]="selectedService.name" required><br>

        <label for="updateDescription">Description:</label>

        <textarea id="updateDescription" name="updateDescription"
[(ngModel)]="selectedService.description" required></textarea><br>

        <label for="updatePrice">Price:</label>

        <input type="number" id="updatePrice" name="updatePrice"
[(ngModel)]="selectedService.price" required><br>

        <button type="submit">Update Service</button>

      </form>       </div>      </div>   </div>  </div>
```

Services.component.ts

```
import { Component, OnInit } from '@angular/core';

import { Service } from '../../shared/model/service.model';

import { AdminService } from '../../services/admin.service';

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

import { UserService } from '../../services/user.service';

@Component({

  selector: 'app-services',

  templateUrl: './services.component.html',

  styleUrls: ['./services.component.css']

})

export class ServicesComponent implements OnInit {
```

```
services: Service[] = [];
newService: Service = new Service(); // Define newService property
selectedService: Service | null = null;
userService: UserService;
constructor(private adminService: AdminService,private _userService: UserService) {
  this.userService = _userService;
}
ngOnInit(): void {
  this.loadServices();
}  loadServices() {
  this.adminService.getServices().subscribe(
    (services: Service[]) => {
      this.services = services;
    },
    (error) => {
      console.error('Error fetching services:', error);
    }    );  }
addService() {
  this.adminService.addService(this.newService).subscribe(
    (response) => {
      console.log('Service added successfully:', response);
      this.loadServices(); // Refresh services after adding a new one
      this.newService = new Service(); // Reset newService for next addition
```

```
    },      (error) => {
      console.error('Error adding service:', error);
    }    );  }
   showUpdateForm(service: Service) {
  // Assign selected service to show its update form
  this.selectedService = service;
 }
updateService(service: Service) {
 this.adminService.updateService(service).pipe(
  catchError(error => {
    console.error('Error updating service:', error);
    // Handle error gracefully, e.g., display error message to user
    return throwError('Error updating service. Please try again later.');
  })  ).subscribe(
  () => {
    this.loadServices(); // Refresh services after updating
    this.selectedService = null; // Hide update form after successful update
  }  );}
 deleteService(serviceId: number) {
  this.adminService.deleteService(serviceId).subscribe(
    () => {
      this.loadServices(); // Refresh services after deletion
    },      (error) => {        console.error('Error deleting service:', error);      }
```

```
  ); }}
```

Lab.component.html

```
<button (click)="toggleForm()">Book Lab Appointment</button><hr>
<form (ngSubmit)="sendTestToServer()" *ngIf="showForm" #testForm="ngForm">
 <div>
  <label>Name:</label>
  <input type="text" [(ngModel)]="patientData.name" name="name" required>
 </div>
 <div>
  <label>Gender:</label>
  <select [(ngModel)]="patientData.gender" name="gender" required>
   <option value="" disabled selected>Select gender</option>
   <option value="Male">Male</option>
   <option value="Female">Female</option>
  </select>
 </div>
 <div>   <label>Age:</label>
  <input type="number" [(ngModel)]="patientData.age" name="age" required>
 </div>  <div>
  <label>Test Name:</label>
  <select [(ngModel)]="patientData.testName" name="testName" required>
   <option value="" disabled>Select a test</option>
```

```
      <option *ngFor="let testName of testNames"
[value]="testName">{{ testName }}</option>
    </select>  </div>  <div>
    <label>Date:</label>
    <input type="date" [(ngModel)]="patientData.date" name="date" required>
  </div>
  <button type="submit" [disabled]="!isFormValid(testForm)">Send
Test</button></form>
<div *ngIf="appointments.length > 0">
  <h2 class="heading">Lab Appointments</h2>
  <table>    <thead>      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Age</th>
        <th>Gender</th>
        <th>Test Name</th>
        <th>Date</th>
        <th>Status</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let appointment of appointments">
        <td>{{ appointment.id }}</td>
```

```
      <td>{{ appointment.name }}</td>
      <td>{{ appointment.age }}</td>
      <td>{{ appointment.gender }}</td>
      <td>{{ appointment.testName }}</td>
      <td>{{ appointment.date | date: 'medium' }}</td>
      <td [ngClass]="getStatusColor(appointment.status)"
class="status">{{ appointment.status }}</td>
    </tr>   </tbody>
  </table></div>
<div *ngIf="appointments.length === 0">
  <p>No lab appointments found.</p>
</div>
```

Lab.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { LabTestService } from '../../services/lab-test.service';
import { NgForm } from '@angular/forms';
import { UserService } from '../../services/user.service';
import { LabAppointment } from '../../shared/model/labAppointment.model';
import { ToastrService } from 'ngx-toastr';
@Component({
  selector: 'app-lab',
  templateUrl: './lab.component.html',
  styleUrls: ['./lab.component.css']
```

```
})export class LabComponent implements OnInit {
  showForm = false;
  userId: number | null = null;
  patientData: any = {
    userId: null, // Initialize as null
    name: '',
    gender: '',
    age: null,
    testName: '',
    date: ''
  };
  testNames: string[] = [];
 appointments: LabAppointment[] = [];
  constructor(private adminService: AdminService, private labTestService:
LabTestService, private userService: UserService,
    private toastrService: ToastrService) { }
  ngOnInit(): void {
    this.getTestNames();
    this.userId = this.userService.userId; // Set userId directly
    this.patientData.userId = this.userId; // Set userId inside patientData
    if (this.userId !== null) {
      this.fetchUserAppointments(this.userId);
    } else {
```

```
    console.error('User ID is null. Unable to fetch appointments.');
  }  }  getTestNames() {
  this.adminService.getTestNames().subscribe(
    (response: any) => {
      this.testNames = response.testNames;
    },
    (error) => {
      console.error('Error fetching test names:', error);
    }    );  }
sendTestToServer() {
  this.labTestService.sendTestToServer(this.patientData).subscribe(
    (response: any) => {
      this.toastrService.success('Test sent to server successfully:');
      // Optionally, handle success response
    },      (error: any) => {
      if (error.error && error.error.error) {
        this.toastrService.error(error.error.error);
      } else {
        this.toastrService.error('Error booking appointment');
      }      }    );  }
fetchUserAppointments(userId: number): void {
  this.labTestService.getUserAppointments(userId).subscribe(
    (appointments: LabAppointment[]) => {
```

```
      this.appointments = appointments;
    },      (error) => {
      console.error('Error fetching user appointments:', error);
    }    );  }
toggleForm() {
  this.showForm = !this.showForm; // Toggle the value of showForm
}
isFormValid(form: NgForm): any {
  return form && form.valid;
}
getStatusColor(status: string): any {
  switch (status) {
    case 'scheduled':
      return { 'scheduled-status': true };
    case 'pending':
      return { 'pending-status': true };
    case 'missed':
      return { 'missed-status': true };
    case 'completed':
      return { 'completed-status': true };
    default:
      return {};
  } }}
```

Report.component.html

```
<div id="contentToConvert">
  <h2>Lab Report</h2>
  <div *ngIf="appointment" class="header">
    <!-- Display appointment details -->
    <div class="left">
      <p class="appointment-details">Name: {{ appointment.name }}</p>
      <p class="appointment-details">Age: {{ appointment.age }}</p>
      <p class="appointment-details">Gender: {{ appointment.gender }}</p>
      <p class="appointment-details">Test Name: {{ appointment.testName }}</p>
    </div>
    <div class="right">
      <p class="appointment-details">Date: {{ appointment.date | date: 'longDate' }}</p>
    </div>
  </div>
    <!-- Display blood test report details if available -->
    <div *ngIf="bloodTestReport">
      <h2>Blood Test Report</h2>
      <table>
        <thead>
          <tr>
            <th>Test Name</th>
            <th>Results</th>
```

```
    <th>Reference</th>
    <th>Units</th>
  </tr>
</thead>
<tbody>
  <!-- Blood test results -->
  <tr>
    <td>Hemoglobin</td>
    <td>{{ bloodTestReport.hemoglobin }}</td>
    <td>{{ getReference('Hemoglobin') }}</td>
    <td>{{ getUnit('Hemoglobin') }}</td>
  </tr>
  <tr>
    <td>White Blood Cell Count</td>
    <td>{{ bloodTestReport.white_blood_cell_count }}</td>
    <td>{{ getReference('White Blood Cell Count') }}</td>
    <td>{{ getUnit('White Blood Cell Count') }}</td>
  </tr>
  <tr>
    <td>Red Blood Cell Count</td>
    <td>{{ bloodTestReport.red_blood_cell_count }}</td>
    <td>{{ getReference('Red Blood Cell Count') }}</td>
    <td>{{ getUnit('Red Blood Cell Count') }}</td>
```

```
      </tr>
      <tr>
        <td>Platelet Count</td>
        <td>{{ bloodTestReport.platelet_count }}</td>
        <td>{{ getReference('Platelet Count') }}</td>
        <td>{{ getUnit('Platelet Count') }}</td>
      </tr>
      <tr>
        <td>Cholesterol</td>
        <td>{{ bloodTestReport.cholesterol }}</td>
        <td>{{ getReference('Cholesterol') }}</td>
        <td>{{ getUnit('Cholesterol') }}</td>
      </tr>
      <tr>
        <td>Glucose</td>
        <td>{{ bloodTestReport.glucose }}</td>
        <td>{{ getReference('Glucose') }}</td>
        <td>{{ getUnit('Glucose') }}</td>
      </tr>
      <!-- Add more rows for other blood test results if needed -->
    </tbody>
  </table>
</div>
```

```
  <!-- Display lipid test report details if available -->
<div *ngIf="lipidTestReport">
 <h2>Lipid Test Report</h2>
 <table>
  <thead>
   <tr>
    <th>Test Name</th>
    <th>Results</th>
    <th>Reference</th>
    <th>Units</th>
   </tr>
  </thead>
  <tbody>
   <tr>
    <td>Total Cholesterol</td>
    <td>{{ lipidTestReport.total_cholesterol }}</td>
    <td>{{ getReference('Total Cholesterol') }}</td>
    <td>{{ getUnit('Total Cholesterol') }}</td>
   </tr>        <tr>
    <td>Triglycerides</td>
    <td>{{ lipidTestReport.triglycerides }}</td>
    <td>{{ getReference('Triglycerides') }}</td>
    <td>{{ getUnit('Triglycerides') }}</td>
```

```
        </tr>         <tr>           <td>HDL Cholesterol</td>
          <td>{{ lipidTestReport.hdl_cholesterol }}</td>
          <td>{{ getReference('HDL Cholesterol') }}</td>
          <td>{{ getUnit('HDL Cholesterol') }}</td>
        </tr>
        <tr>
          <td>LDL Cholesterol</td>
          <td>{{ lipidTestReport.ldl_cholesterol }}</td>
          <td>{{ getReference('LDL Cholesterol') }}</td>
          <td>{{ getUnit('LDL Cholesterol') }}</td>
        </tr>
        <!-- Add more rows for other lipid test results if needed -->
      </tbody>
    </table>
  </div>
 </div>
 <button *ngIf="bloodTestReport || lipidTestReport"
(click)="downloadHTMLAsPDF()">Download as PDF</button>
 <div *ngIf="!(bloodTestReport || lipidTestReport)">
  <p>No reports available to download.</p>
 </div>
```

Report.component.ts

```
import { Component, OnInit} from '@angular/core';
```

```
import { ActivatedRoute } from '@angular/router';
import { AdminLabService } from '../../services/admin-lab.service';
import jsPDF from 'jspdf';
import html2canvas from 'html2canvas';
import { UserService } from '../../services/user.service';
@Component({
  selector: 'app-report',
  templateUrl: './report.component.html',
  styleUrl: './report.component.css'
})export class ReportComponent implements OnInit {
  userId: number | null = null;
  bloodTestReport: any | null = null;
  lipidTestReport: any | null = null;
  appointment: any | null = null;
  formData: any[] = [];
  constructor(
    private route: ActivatedRoute,
    private adminlabService: AdminLabService,
    private userService: UserService
  ) { }
  ngOnInit(): void {
      this.userId = this.userService.userId;
    if (!this.userId) {
```

```
    console.error('User ID not found in UserService');
    // Handle the case when user ID is not found
  } else {
    this.fetchUserReport(this.userId);
  }  }
 fetchUserReport(userId: number) {
 this.adminlabService.getFormDataForUser().subscribe(formData => {
  this.formData = formData;
 });
 this.adminlabService.getAppointmentsForUser(userId).subscribe(appointments => {
  if (appointments.length > 0) {
    this.appointment = appointments[0];
    switch (this.appointment.testName) {
      case 'Complete Blood Count (CBC) Test':

this.adminlabService.getBloodTestReportsForUser(this.appointment.userId).subscribe(bl
oodTestReport => {
          this.bloodTestReport = bloodTestReport[0];
                });
        break;       case 'Lipid Profile Test':

this.adminlabService.getLipidTestReportsForUser(this.appointment.userId).subscribe(lipi
dTestReport => {
          this.lipidTestReport = lipidTestReport[0];
```

```
        });
        break;
      // Add cases for other test types if needed
      default:
        // Handle other types of tests or no tests
        break;
    }   }  });  }
  getReference(fieldName: string): string {
    const formDataEntry = this.formData.find(entry => entry.fieldName === fieldName);
    return formDataEntry ? formDataEntry.value : ''; // Return the value from form data entry or an empty string if not found
  }
  getUnit(fieldName: string): string {
    const formDataEntry = this.formData.find(entry => entry.fieldName === fieldName);
    return formDataEntry ? formDataEntry.unit : ''; // Return the unit from form data entry or an empty string if not found
  }
  downloadHTMLAsPDF() {
    const contentToConvert = document.getElementById('contentToConvert');
      if (contentToConvert) {
     html2canvas(contentToConvert).then(canvas => {
      const imgData = canvas.toDataURL('image/png');
      const pdf = new jsPDF('p', 'mm', 'a4');
```

```
      const imgWidth = pdf.internal.pageSize.getWidth();
      const imgHeight = canvas.height * imgWidth / canvas.width;
      pdf.addImage(imgData, 'PNG', 0, 0, imgWidth, imgHeight);
      pdf.save('html_to_pdf.pdf');
    });
  } else {
    console.error('HTML content not found');
    // Handle missing HTML content (e.g., display error message)
  }  }}
```

Test-details.component.html

```
<div class="test-names-container">
  <h2>Test Names</h2>
  <ul class="test-names-list">
    <li *ngFor="let testName of testNames" (click)="selectTest(testName)" class="test-name">{{ testName }}</li>
  </ul>
</div>
<<div class="test-details-container">
  <div class="details" *ngIf="selectedTest">
    <div class="modal-content">
      <button class="close" (click)="closeDetails()">&times;</button>
      <p><strong>Test Name:</strong> {{ selectedTest.testName }}</p>
      <p><strong>Description:</strong> {{ selectedTest.description }}</p>
```

```
<p><strong>Purpose:</strong> {{ selectedTest.purpose }}</p>
<p><strong>Procedure:</strong> {{ selectedTest.procedure }}</p>
<p><strong>Parameters:</strong> {{ selectedTest.parameters }}</p>
<p><strong>Interpretation:</strong> {{ selectedTest.interpretation }}</p>
<p><strong>Importance:</strong> {{ selectedTest.importance }}</p>
<p><strong>Preparation:</strong> {{ selectedTest.preparation }}</p>
<p><strong>Risks:</strong> {{ selectedTest.risks }}</p>
<p><strong>Follow-Up:</strong> {{ selectedTest.followUp }}</p>
<div class="admin-controls" *ngIf="userService.isAdmin()">
   <button (click)="deleteTest()">Delete</button>
</div>       </div>    </div></div>
```

Test-details.component.ts

```
import { Component, OnInit } from '@angular/core';
import { AdminService } from '../services/admin.service';
import { UserService } from '../services/user.service';
@Component({
  selector: 'app-test-details',
  templateUrl: './test-details.component.html',
  styleUrls: ['./test-details.component.css']
})
export class TestDetailsComponent implements OnInit {
  userService: UserService;
  testNames: string[] = [];
```

```
selectedTest: any;
constructor(private adminService: AdminService, _userService: UserService) {
  this.userService = _userService;
}
ngOnInit(): void {
  this.getTestNames();
}
getTestNames() {
  this.adminService.getTestNames().subscribe(
    (response: any) => {
      this.testNames = response.testNames;
      console.log('Test names:', this.testNames);
    },
    (error) => {
      console.error('Error fetching test names:', error);
    }
  );
}
selectTest(testName: string) {
  this.adminService.getTestDetails(testName).subscribe(
    (response: any) => {
      this.selectedTest = response.testDetails;
    },
```

```
    (error) => {
      console.error('Error fetching test details:', error);
    }
  );
}
closeDetails() {
  this.selectedTest = null;
}
deleteTest() {
  if (!this.selectedTest || !this.selectedTest.id) {
    console.error('Invalid test ID:', this.selectedTest);
    return;
  }
  const testId = this.selectedTest.id;
  this.adminService.deleteTest(testId).subscribe(
    (response: any) => {
      // Handle success
      console.log('Test deleted successfully:', response);
      // Optionally, refresh the test names list or update the UI after deletion
      this.getTestNames();
    },
    (error) => {
      console.error('Error deleting test:', error);
```

```
      // Handle error, show error message or handle accordingly
    }
  );
 }
 }
```

Chat.component.html

```
<div class="messenger-container">
  <div class="user-list">
    <div *ngFor="let user of userList" class="user-item">
      <p class="user-name">{{ user.name }}</p>
      <button (click)="selectUser(user)" class="chat-button">Chat</button>
    </div>
  </div>

  <div *ngIf="selectedUserId" class="chat-window">
    <div class="chat-header">
      <h3>Chatting with {{ getSelectedUserName() }}</h3>
    </div>
    <div class="chat-messages">
      <div *ngFor="let msg of messages">
        <p>{{ msg.senderId }}: {{ msg.message }}</p>
      </div>
```

```
    </div>
    <input type="text" [(ngModel)]="messageInput" class="message-input">
    <button (click)="sendMessage()" class="send-button">Send</button>
   </div>
  </div>
```

Chat.component.ts

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import io from 'socket.io-client';
import { ActivatedRoute } from '@angular/router';
import { AdminService } from '../../services/admin.service';
import { UserService } from '../../services/user.service';
import { ChatService } from '../../services/chat.service';

@Component({
  selector: 'app-user-chat',
  templateUrl: './user-chat.component.html',
  styleUrls: ['./user-chat.component.css'] // Note the correct property name for styleUrls
})
export class UserChatComponent implements OnInit, OnDestroy {
  socket: any;
  currentUserId: number | undefined;
  selectedUserId: string = '';
  messageInput: string = '';
```

```
messages: { senderId: number | undefined, message: string }[] = [];
userList: any[] = [];
isAdmin: boolean = false; // New property to check if the logged-in user is an admin

constructor(
  private route: ActivatedRoute,
  private adminService: AdminService,
  private userService: UserService,
  private chatService: ChatService
) { }

ngOnInit(): void {
  this.socket = io('http://localhost:3000');

  this.route.queryParams.subscribe(params => {
    this.selectedUserId = params['userId'];
  });

  this.currentUserId = this.userService.currentUser?.id;

  // Fetch all users
  this.adminService.getAllUsers().subscribe(
    (users: any[]) => {
```

```
      // Filter userList based on isAdmin field
      this.userList = users.filter(user => user.isAdmin);
      // Check if the logged-in user is an admin
      this.isAdmin = this.userList.some(user => user.id === this.currentUserId);
    },
    (error) => {
      console.error('Error fetching users:', error);
    }
  );

  this.socket.emit('authenticate', this.currentUserId);

  this.socket.on('chat message', ({ senderId, message }: { senderId: number | undefined, message: string }) => {
    if (this.selectedUserId && senderId === +this.selectedUserId) {
      this.messages.push({ senderId, message });
    }
  });

  // Fetch messages for the selected user
  if (this.selectedUserId) {
    this.fetchMessagesForUser(this.selectedUserId);
  }
```

```
}

ngOnDestroy(): void {
  this.socket.disconnect();
}

// Method to send a message
sendMessage(): void {
  if (!this.selectedUserId) {
    console.error('No user selected');
    return;
  }
  const data = {
    senderId: this.currentUserId,
    receiverId: this.selectedUserId,
    message: this.messageInput
  };
  this.socket.emit('chat message', data);
  this.messages.push({ senderId: data.senderId, message: data.message });
  this.messageInput = '';

  // Save the message to the database
  this.chatService.saveMessage(data).subscribe(
```

```
    (response) => {
      console.log('Message saved successfully:', response);
    },
    (error) => {
      console.error('Error saving message:', error);
    }
  );
}

// Method to fetch messages for a user
fetchMessagesForUser(userId: string): void {
  this.chatService.getMessagesForUser(userId).subscribe(
    (messages: any[]) => {
      this.messages = messages;
    },
    (error) => {
      console.error('Error fetching messages:', error);
    }
  );
}

// Method to get the name of the selected user
getSelectedUserName(): string {
```

```
  const selectedUser = this.userList.find(user => user.id.toString() ===
this.selectedUserId);
  return selectedUser ? selectedUser.name : '';
 }

 // Method to select a user
 selectUser(user: any): void {
  this.selectedUserId = user.id.toString();
  this.messages = []; // Clear existing messages when selecting a new user
  this.fetchMessagesForUser(this.selectedUserId); // Fetch messages for the selected
user
 }
}
```

ADMIN

Adminheader.component.html

```
<header>
  <nav>
    <div class="wrapper">
      <div class="logo"><a routerLink="/home">CLINIC ADMIN</a></div>
      <ul class="nav-links">
        <li><a routerLink="/adminusers">Users</a></li>
        <li><a routerLink="/adminappointments">Appointment</a></li>
        <li><a routerLink="/adminlab">Report</a></li>
```

```
            <li><a routerLink="/services">Services</a></li>
            <li><a routerLink="/testInfo">Test</a></li>
            <li><a routerLink="/chat">Chat</a></li>
            <li><a routerLink="/about">About</a></li>
            <li>
              <a href="javascript:void(0);" class="desktop-item">Admin</a>
              <ul class="drop-menu">
                <li><a>{{user.name}}</a></li>
                <li>
                  <a routerLink="changepassword">change Password</a>
                </li>
                <li><a (click)="logout()">Logout</a></li>
              </ul>
            </li>
          </ul>
        </div>
    </nav>
</header>
```

Adminheader.component.ts

```
import { Component } from '@angular/core';
import { UserService } from '../../services/user.service';
import { User } from '../../shared/model/user.model';
```

```
@Component({
  selector: 'app-adminheader',
  templateUrl: './adminheader.component.html',
  styleUrl: './adminheader.component.css'
})
export class AdminheaderComponent {

isAdmin: any;
user: User = new User();

ngOnInit(): void {
  // Subscribe to userObservable to update user data
  this.userService.userObservable.subscribe((newUser) => {
    this.user = newUser;
  });
}
  constructor(private userService: UserService) {}




  logout() {
    this.userService.logOut();
```

```
  }
}
```

AdminAppointment.component.html

```
<h2>Appointments</h2>


<div *ngIf="appointments.length > 0; else noAppointments">
  <table>
    <thead>
      <tr>
        <th>Appointment ID</th>
        <th>User ID</th>
        <th>Doctor ID</th>
        <th>Date</th>
        <th>Time</th>
        <th>Type</th>
        <th>Reason</th>
        <th>Created At</th>
        <th>Status</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let appointment of appointments">
```

```
        <td>{{ appointment.appointment_id }}</td>

        <td>{{ appointment.user_id }}</td>

        <td>{{ appointment.doctor_id }}</td>

        <td>{{ appointment.appointment_date }}</td>

        <td>{{ appointment.appointment_time }}</td>

        <td>{{ appointment.appointment_type }}</td>

        <td>{{ appointment.reason }}</td>

        <td>{{ appointment.created_at }}</td>

        <td > <select #statusSelect (change)="updateStatus(statusSelect.value,
appointment.appointment_id)">

            <option value="pending" [selected]="appointment.status ===
'pending'">Pending</option>

            <option value="scheduled" [selected]="appointment.status ===
'scheduled'">Scheduled</option>

            <option value="missed" [selected]="appointment.status ===
'missed'">Missed</option>

            <option value="completed" [selected]="appointment.status ===
'completed'">Completed</option>

          </select></td>

        <td>

          <button
(click)="deleteAppointment(appointment.appointment_id)">Delete</button>

        </td>

      </tr>
```

```
    </tbody>
  </table>
</div>

<!-- Template for no appointments found -->
<ng-template #noAppointments>
  <p class="no-appointments">No appointments found.</p>
</ng-template>
```

AdminAppointment.component.ts

```
// admin-appointment.component.ts
import { Component, OnInit } from '@angular/core';
import { AdminService } from '../../services/admin.service';
import { IAppointment } from '../../shared/interfaces/IAppointment ';

@Component({
  selector: 'app-admin-appointment',
  templateUrl: './admin-appointment.component.html',
  styleUrls: ['./admin-appointment.component.css']
})
export class AdminAppointmentComponent implements OnInit {
  appointments: IAppointment[] = [];

  constructor(private appointmentService: AdminService) { }
```

```
  ngOnInit(): void {
    this.getAppointments();
  }

  getAppointments(): void {
    this.appointmentService.getAllAppointments()
      .subscribe(appointments => this.appointments = appointments);
  }

  updateStatus(status: string, appointmentId: string | number): void {
    // Convert appointmentId to a number if it's a string
    const id: number = typeof appointmentId === 'string' ? parseInt(appointmentId, 10) :
appointmentId;

    this.appointmentService.updateAppointmentStatus(id, status)
      .subscribe(() => {
        // Update the status locally after successful update
        const appointment = this.appointments.find(a => a.appointment_id === id);
        if (appointment) {
          appointment.status = status;
        }
      }, error => {
```

```
        console.error('Error updating status:', error);
      });
  }
getStatusClass(status: string): string {
  switch (status) {
    case 'pending':
      return 'pending-status';
    case 'scheduled':
      return 'scheduled-status';
    case 'missed':
      return 'missed-status';
    case 'completed':
      return 'completed-status';
    default:
      return '';
  }
}
```

```
  deleteAppointment(appointmentId: number): void {
```

```
    this.appointmentService.deleteAppointment(appointmentId)
      .subscribe(() => {
        // Remove the deleted appointment from the local array
        this.appointments = this.appointments.filter(appointment =>
appointment.appointment_id !== appointmentId);
      });
  }
}
```

AdminLab.component.html

```
<div class="body">
    <!-- Add Reference Link (positioned top right) -->
    <div style="text-align: right;">
      <a routerLink="/reference" class="add-reference-link">Add Reference Value</a>
    </div>
    <br>
    <!-- Toggle Buttons for Reports -->
    <div>
      <button (click)="showBloodTestReports()">Blood Test Reports</button>
      <button (click)="showLipidTestReports()">Lipid Test Reports</button>
    </div>

    <!-- Report Form above the tables -->
    <div *ngIf="selectedAppointment">
```

```
    <h2>Appointment Details</h2>
    <p><strong>Appointment ID:</strong> {{ selectedAppointment.id }}</p>
    <p><strong>User ID:</strong> {{ selectedAppointment.user_id }}</p>
    <p><strong>Test Name:</strong> {{ selectedAppointment.testName }}</p>

    <!-- Report Form -->
    <div *ngIf="reportFormFields.length > 0">
      <form (ngSubmit)="submitReport()">
        <!-- Loop through form fields -->
        <div *ngFor="let field of reportFormFields">
          <label>{{ field.label }}</label>
          <input type="{{ field.type }}" name="{{ field.name }}"
[(ngModel)]="formData[field.name]">
        </div>
        <button type="submit">Submit Report</button>
      </form>
    </div>
  </div>

  <!-- Blood Test Reports Table -->
  <div *ngIf="showBloodReports">
    <h2>Blood Test Reports</h2>
    <table>
```

```
<thead>
  <tr>
    <th>Report ID</th>
    <th>Appointment ID</th>
    <th>User ID</th>
    <th>Hemoglobin</th>
    <th>White Blood Cell Count</th>
    <th>Red Blood Cell Count</th>
    <th>Platelet Count</th>
    <th>Cholesterol</th>
    <th>Glucose</th>
    <th>Actions</th>
  </tr>
</thead>
<tbody>
  <!-- Blood Test Reports Rows -->
  <tr *ngFor="let report of bloodTestReports">
    <td>{{ report.report_id }}</td>
    <td>{{ report.appointment_id }}</td>
    <td>{{ report.user_id }}</td>
    <td>{{ report.hemoglobin }}</td>
    <td>{{ report.white_blood_cell_count }}</td>
    <td>{{ report.red_blood_cell_count }}</td>
```

```
        <td>{{ report.platelet_count }}</td>
        <td>{{ report.cholesterol }}</td>
        <td>{{ report.glucose }}</td>
        <td>
          <button (click)="deleteBloodTestReport(report.report_id)">Delete</button>
          <button (click)="generateReport(report.user_id)">Generate</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>

<!-- Lipid Test Reports Table -->
<div *ngIf="showLipidReports">
  <h2>Lipid Test Reports</h2>
  <table>
    <thead>
      <tr>
        <th>Report ID</th>
        <th>Appointment ID</th>
        <th>User ID</th>
        <th>Total Cholesterol</th>
        <th>Triglycerides</th>
```

```
      <th>HDL Cholesterol</th>
      <th>LDL Cholesterol</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <!-- Lipid Test Reports Rows -->
    <tr *ngFor="let report of lipidTestReports">
      <td>{{ report.report_id }}</td>
      <td>{{ report.appointment_id }}</td>
      <td>{{ report.user_id }}</td>
      <td>{{ report.total_cholesterol }}</td>
      <td>{{ report.triglycerides }}</td>
      <td>{{ report.hdl_cholesterol }}</td>
      <td>{{ report.ldl_cholesterol }}</td>
      <td>
        <button (click)="deleteLipidTestReport(report.report_id)">Delete</button>
        <button (click)="generateReport(report.user_id)">Generate</button>
      </td>
    </tr>
  </tbody>
 </table>
</div>
```

```
<!-- Lab Appointments Section -->
<h2>Lab Appointments</h2>
<table>
  <thead>
    <tr>
      <th>ID</th>
      <th>User ID</th>
      <th>Name</th>
      <th>Age</th>
      <th>Gender</th>
      <th>Test Name</th>
      <th>Date</th>
      <th>Status</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <!-- Lab Appointments Rows -->
    <tr *ngFor="let appointment of appointments">
      <td>{{ appointment.id }}</td>
      <td>{{ appointment.user_id }}</td>
      <td>{{ appointment.name }}</td>
```

```
        <td>{{ appointment.age }}</td>
        <td>{{ appointment.gender }}</td>
        <td>{{ appointment.testName }}</td>
        <td>{{ appointment.date | date: 'medium' }}</td>
        <td>
          <select #statusSelect (change)="updateStatus(statusSelect.value,
appointment.id)">
            <option value="pending" [selected]="appointment.status ===
'pending'">Pending</option>
            <option value="scheduled" [selected]="appointment.status ===
'scheduled'">Scheduled</option>
            <option value="missed" [selected]="appointment.status ===
'missed'">Missed</option>
            <option value="completed" [selected]="appointment.status ===
'completed'">Completed</option>
            <option value="report" [selected]="appointment.status ===
'report'">Report</option>
          </select>
        </td>
        <td>
          <button (click)="deleteAppointment(appointment.id)">Delete</button>
          <button
(click)="openReportFormByAppointment(appointment)">Report</button>
        </td>
      </tr>
```

```
    </tbody>
  </table>
 </div>
```

AdminLab.component.ts

```
import { Component } from '@angular/core';
import { LabAppointment } from '../../shared/model/labAppointment.model';
import { AdminLabService } from '../../services/admin-lab.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-admin-lab',
  templateUrl: './admin-lab.component.html',
  styleUrls: ['./admin-lab.component.css']
})
export class AdminLabComponent {

  showBloodReports: boolean = false;
  showLipidReports: boolean = false;
  appointments: LabAppointment[] = [];
  reportFormFields: any[] = [];
  formData: any = {};
  selectedAppointment: LabAppointment | null = null;
```

```
lipidTestReports: any= [];
bloodTestReports: any= [];



constructor(private adminService: AdminLabService,private router: Router) { }

ngOnInit(): void {
  this.getLabAppointments();
  this.getBloodTestReports();
  this.getLipidTestReports();
}

getLabAppointments() {
  this.adminService.getAllLabAppointments().subscribe(
    (appointments: LabAppointment[]) => {
      this.appointments = appointments;
    },
    (error: any) => {
      console.error('Error fetching appointments:', error);
    }
  );
}
```

```
openReportFormByAppointment(appointment: LabAppointment) {
  this.selectedAppointment = appointment;
  this.formData = {
    appointment_id: appointment.id,
    user_id: appointment.user_id,
    testName: appointment.testName
  };
  this.reportFormFields = this.generateFormFields(appointment.testName);
}

submitReport() {
  this.adminService.sendReportData(this.formData).subscribe(
    (response) => {
      console.log('Report submitted successfully:', response);
      this.selectedAppointment = null;
    },
    (error) => {
      console.error('Error submitting report:', error);
    }
  );
  this.resetFormData();
}
```

```
  resetFormData() {
    this.formData = {};
  }

  generateFormFields(testName: string): any[] {
    let formFields: any[] = [];
    const lowerTestName = testName.toLowerCase();
    switch (lowerTestName) {
      case 'complete blood count (cbc) test':
        formFields = [
          { label: 'Hemoglobin', name: 'hemoglobin', type: 'number' },
          { label: 'White Blood Cell Count', name: 'white_blood_cell_count', type:
'number' },
          { label: 'Red Blood Cell Count', name: 'red_blood_cell_count', type: 'number' },
          { label: 'Platelet Count', name: 'platelet_count', type: 'number' },
          { label: 'Cholesterol', name: 'cholesterol', type: 'number' },
          { label: 'Glucose', name: 'glucose', type: 'number' }
        ];
        break;
      case 'lipid profile test':
        formFields = [
          { label: 'Total Cholesterol', name: 'total_cholesterol', type: 'number' },
```

```
        { label: 'Triglycerides', name: 'triglycerides', type: 'number' },
        { label: 'HDL Cholesterol', name: 'hdl_cholesterol', type: 'number' },
        { label: 'LDL Cholesterol', name: 'ldl_cholesterol', type: 'number' }
      ];
      break;
    default:
      // Handle other test types if needed
      break;
  }
  return formFields;
}



updateStatus(status: string, appointmentId: string | number): void {
  // Convert appointmentId to a number if it's a string
  const id: number = typeof appointmentId === 'string' ? parseInt(appointmentId, 10) :
appointmentId;

  this.adminService.updateAppointmentStatus(id, status)
    .subscribe(() => {
      // Update the status locally after successful update
      const appointment = this.appointments.find(a => a.id === id);
      if (appointment) {
```

```
        appointment.status = status;
      }
    }, error => {
      console.error('Error updating status:', error);
    });
}

  deleteAppointment(appointmentId: number): void {
    this.adminService.deleteAppointment(appointmentId)
      .subscribe(() => {
        // Remove the deleted appointment from the local array
        this.appointments = this.appointments.filter(appointment => appointment.id !==
appointmentId);
      });
    }



    getBloodTestReports(): void {
      this.adminService.getBloodTestReports().subscribe(reports => {
        this.bloodTestReports = reports;
      });
    }
```

```
getLipidTestReports(): void {
  this.adminService.getLipidTestReports().subscribe(reports => {
    this.lipidTestReports = reports;
  });
}

deleteBloodTestReport(reportId: number) {
  // Implement deletion logic for blood test reports
  // Call your service method to delete the report using reportId
}

deleteLipidTestReport(reportId: number) {
  // Implement deletion logic for lipid test reports
  // Call your service method to delete the report using reportId
}

showBloodTestReports() {
  // Show blood test reports and hide lipid test reports
  this.showBloodReports = true;
  this.showLipidReports = false;
}

showLipidTestReports() {
```

```
    // Show lipid test reports and hide blood test reports
    this.showBloodReports = false;
    this.showLipidReports = true;
  }




  generateReport(userId: number) {
    if (userId !== null) {
      this.router.navigate(['/userReport', userId]);
    } else {
      console.error("User ID is required.");
    }}}
```

SERVICES

User.services.ts

```
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { Injectable } from '@angular/core';
import { ToastrService } from 'ngx-toastr';
import { BehaviorSubject, Observable, tap } from 'rxjs';
import { User } from '../shared/model/user.model';
import { IUserLogin } from '../shared/interfaces/IUserLogin';
import { IUserRegister } from '../shared/interfaces/IUserRegister';
```

```
import { USER_LOGIN_URL, USER_REGISTER_URL } from
'../shared/constants/urls';


const USER_KEY = 'User';


@Injectable({
  providedIn: 'root'
})
export class UserService {
  private userSubject = new BehaviorSubject<User>(this.getUserFromLocalStorage());
  public userObservable: Observable<User>;


  constructor(private http: HttpClient,private router: Router, private toastrService:
ToastrService) {
    this.userObservable = this.userSubject.asObservable();
  }


  public get currentUser(): User {
    return this.userSubject.value;
  }


  public get userId(): number | null {
    return this.currentUser ? this.currentUser.id : null;
```

```
}

login(userLogin: IUserLogin): Observable<User> {
  return this.http.post<User>(USER_LOGIN_URL, userLogin).pipe(
    tap({
      next: (user) => {
        this.setUserToLocalStorage(user);
        this.userSubject.next(user);
        this.toastrService.success(
          'Login Successful', `Welcome ${user.name}! 😊`
        );
      },
      error: (errorResponse) => {
        this.toastrService.error('Your email or password is incorrect" 🧐');
      }
    })
  );
}

register(userRegiser: IUserRegister): Observable<User> {
  return this.http.post<User>(USER_REGISTER_URL, userRegiser).pipe(
    tap({
```

```
      next: (user) => {
        this.setUserToLocalStorage(user);
        this.userSubject.next(user);
        this.toastrService.success(
          'Register Successful'
        );
      },
      error: (errorResponse) => {
        this.toastrService.error(
          'Register Failed', 'UserName or Email Already Exist! 😳'
        );
      }
    })
  );
}

logOut() {
  this.userSubject.next(new User());
  localStorage.removeItem(USER_KEY);
  window.location.reload();
  this.router.navigateByUrl('/login');
}
```

```
  private setUserToLocalStorage(user: User) {
    localStorage.setItem(USER_KEY, JSON.stringify(user));
  }

  private getUserFromLocalStorage(): User {
    const userJson = localStorage.getItem(USER_KEY);
    if (userJson) return JSON.parse(userJson) as User;
    return new User();
  }

  public isAdmin(): boolean {
    // Check if the current user exists and if their isAdmin field is true
    return this.currentUser && this.currentUser.isAdmin;
  }

  changePassword(email: string, currentPassword: string, newPassword: string):
Observable<any> {
    const payload = {
      email,
      currentPassword,
      newPassword
    };
```

```
    return this.http.post(`http://localhost:3000/api/users/change-password`, payload);
  }



}
```

Admin.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { IUser } from '../shared/interfaces/IUser';
import { APPOINTMENTS_URL, TESTINFO_URL, USERS_URL } from
'../shared/constants/urls';
import { Appointment } from '../shared/model/adminappointments.model';
import { Service } from '../shared/model/service.model';
import { LabAppointment } from '../shared/model/labAppointment.model';

@Injectable({
  providedIn: 'root'
})
export class AdminService {

  constructor(private http: HttpClient) { }
```

```
getAllUsers(): Observable<IUser[]> {
  return this.http.get<IUser[]>(USERS_URL);
}

getAllAppointments(): Observable<Appointment[]> {
  return this.http.get<Appointment[]>(APPOINTMENTS_URL);
}

updateAppointmentStatus(appointmentId: number, status: string): Observable<any> {
  return this.http.put(`${APPOINTMENTS_URL}/${appointmentId}/status`, { status });
}

deleteAppointment(appointmentId: number): Observable<any> {
  return this.http.delete(`${APPOINTMENTS_URL}/${appointmentId}`);
}


//test info service

submitFormData(formData: any) {
  return this.http.post<any>(`${TESTINFO_URL}/submit-test-details`, formData);
}
```

```
  getTestNames(): Observable<any> {
    return this.http.get<any>('http://localhost:3000/api/testinfo/test-names');
  }

  getTestDetails(testName: string) {
    return this.http.get<any>(`http://localhost:3000/api/testinfo/test-details/${testName}`);
  }

  deleteTest(testId: number): Observable<any> {
    const url = `http://localhost:3000/api/testinfo/delete/${testId}`; // Assuming your API endpoint for deleting test details is /testinfo/delete/:id
    return this.http.delete(url); // Assuming your API supports DELETE method for deleting test details
  }



  //service
  private readonly SERVICE_URL = 'http://localhost:3000/api/admin/services'; // Update URL as per your backend
```

```
  getServices(): Observable<Service[]> {
    return this.http.get<Service[]>(this.SERVICE_URL);
  }

  addService(service: Service): Observable<Service> {
    return this.http.post<Service>(this.SERVICE_URL, service);
  }

  updateService(service: Service): Observable<Service> {
    const url = `${this.SERVICE_URL}/${service.id}`;
    return this.http.put<Service>(url, service);
  }

  deleteService(serviceId: number): Observable<any> {
    const url = `${this.SERVICE_URL}/${serviceId}`;
    return this.http.delete(url, { responseType: 'text' }); // Specify response type as text
  }}
```

adminLab.service.ts

```
import { Injectable } from '@angular/core';
import { LabAppointment } from '../shared/model/labAppointment.model';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';
```

```
@Injectable({
  providedIn: 'root'
})
export class AdminLabService {
  getBloodTestReport(reportId: number) {
    throw new Error('Method not implemented.');
  }

  constructor(private http: HttpClient) { }

  //lab appointment side

  getAllLabAppointments(): Observable<LabAppointment[]> {
    const url = `http://localhost:3000/api/labAppointment/appointments`;
    return this.http.get<LabAppointment[]>(url);
  }

  sendReportData(data: any): Observable<any> {
    return this.http.post<any>(`http://localhost:3000/api/reports`, data);
  }

  updateAppointmentStatus(appointmentId: number, status: string): Observable<any> {
```

```
    return
this.http.put(`http://localhost:3000/api/labAppointment/${appointmentId}/status`,
{ status });
  }

  deleteAppointment(appointmentId: number): Observable<any> {
    return this.http.delete(`http://localhost:3000/api/labAppointment/${appointmentId}`);
  }

  getBloodTestReports(): Observable<any[]> {
    return this.http.get<any[]>('http://localhost:3000/api/blood-test-reports');
  }

  getLipidTestReports(): Observable<any[]> {
    return this.http.get<any[]>('http://localhost:3000/api/lipid-test-reports');
  }


  getBloodTestReportsForUser(userId: number): Observable<any[]> {
    return this.http.get<any[]>(`http://localhost:3000/api/blood-test-
reports?user_id=${userId}`);
  }
```

```
  getLipidTestReportsForUser(userId: number): Observable<any[]> {
    return this.http.get<any[]>(`http://localhost:3000/api/lipid-test-
reports?user_id=${userId}`);
  }

  getAppointmentsForUser(userId: number): Observable<any[]> {
    return
this.http.get<any[]>(`http://localhost:3000/api/labAppointment?user_id=${userId}`);
  }

  getFormDataForUser(): Observable<any[]> {
    return this.http.get<any[]>(`http://localhost:3000/api/form-data`);
  }}
```

Lab-test.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { LabAppointment } from '../shared/model/labAppointment.model';

@Injectable({
  providedIn: 'root'
})
export class LabTestService {
```

```
  constructor(private http: HttpClient) { }


  sendTestToServer(testData: any): Observable<any> {
    return this.http.post<any>('http://localhost:3000/api/labAppointment/sendTest',
testData);
  }


  getUserAppointments(userId: number): Observable<LabAppointment[]> {
    const url = `http://localhost:3000/api/labAppointment/user-appointments/${userId}`;
    return this.http.get<LabAppointment[]>(url);
  }
}
```

BACKEND

Server.js

```
const express = require('express');
const cors = require('cors');
const socketIo = require('socket.io');
const userRouter = require('./routers/user.router');
const appointmentRouter = require('./routers/appointment.router');
const chatRouter = require('./routers/chat.router');
const doctorsRouter = require('./routers/doctors.router');
const getAllUsersRouter = require('./routers/admin/getAllUsers.router');
```

```
const getAllAppointments = require('./routers/admin/getAllAppointments.router');
const updateStatus = require('./routers/admin/updateStatus.router');
const deleteAppointment = require('./routers/admin/deleteAppointment.router');
const testinfo = require('./routers/admin/testinfo.router');
const services = require('./routers/admin/services.router');
const labAppointment = require('./routers/labAppointment.router');
const report = require('./routers/admin/report.router');
const reference = require('./routers/admin/reference.router');
const userReport = require('./routers/userReport.router');
const pdf = require('./routers/admin/pdf.router');

const app = express();
app.use(cors());
app.use(express.json());

app.use('/api/users', userRouter);
app.use('/api/user-appointment', appointmentRouter);
app.use('/api/doctors', doctorsRouter);

app.use('/api/all-users', getAllUsersRouter);
app.use('/api/appointments', getAllAppointments);
app.use('/api/appointments', updateStatus);
app.use('/api/appointments', deleteAppointment);
```

```
app.use('/api/testinfo', testinfo);

app.use('/api/testinfo',testinfo);

app.use('/api/testinfo',testinfo);

app.use('/api/testinfo/delete',testinfo);

app.use('/api/labAppointment', labAppointment);


app.use('/api/admin/services',services);

app.use('/api',report);

app.use('/api/reference',reference);

app.use('/api',userReport);


app.use('/api/pdf',pdf);


app.use('/api', chatRouter);


const server = app.listen(process.env.PORT || 3000, () => {

  console.log(`Server is listening on port ${server.address().port}`);

});



const io = socketIo(server, {

  cors: {

    origin: "*", // Allow requests from this origin
```

```
    methods: ["GET", "POST"] // Allow only these methods
  }
});

// Store user socket mappings
const userSockets = {};

io.on('connection', (socket) => {
  console.log('A user connected');

  // Handle user authentication
  socket.on('authenticate', (userId) => {
    console.log(`User authenticated: ${userId}`);
    userSockets[userId] = socket;
  });

  socket.on('disconnect', () => {
    console.log('User disconnected');
    // Remove user socket mapping on disconnect
    const userId = Object.keys(userSockets).find(key => userSockets[key] === socket);
    delete userSockets[userId];
  });
```

```
  // Handle chat messages
  socket.on('chat message', ({ senderId, receiverId, message }) => {
    console.log(`Message received from ${senderId} to ${receiverId}: ${message}`);
    // Forward the message to the receiver
    const receiverSocket = userSockets[receiverId];
    if (receiverSocket) {
      receiverSocket.emit('chat message', { senderId, message });
    } else {
      console.log(`User ${receiverId} is not online`);
      // Optionally, you can inform the sender that the receiver is not online
      socket.emit('chat error', { receiverId, message: 'User is not online' });
    }
  });
});
```

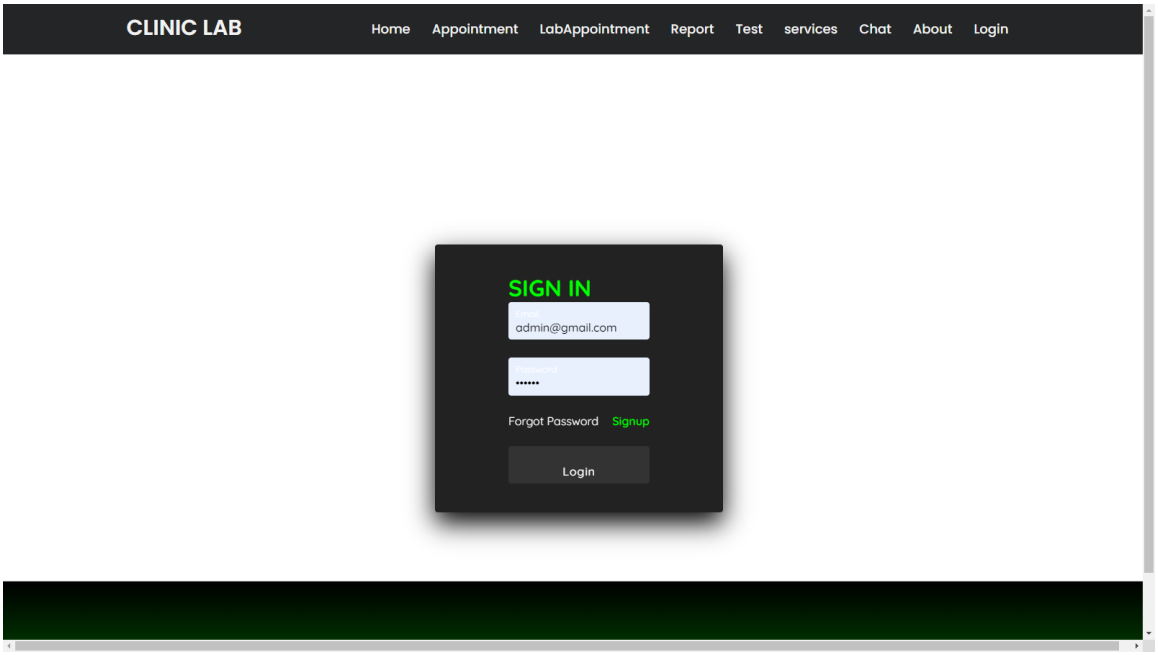
Db.js

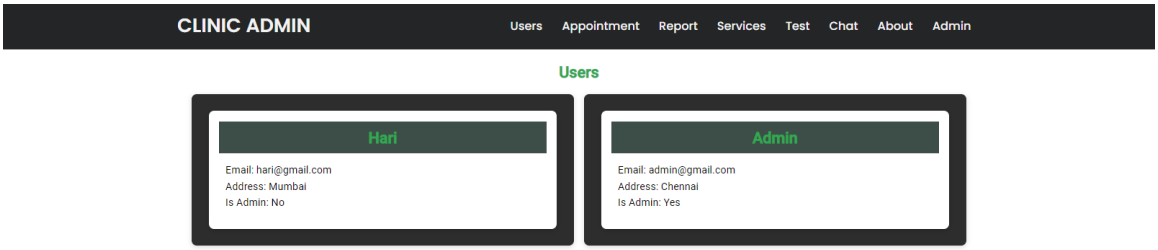
```
const mysql = require('mysql');
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'clinic'
});
connection.connect((err) => {
```

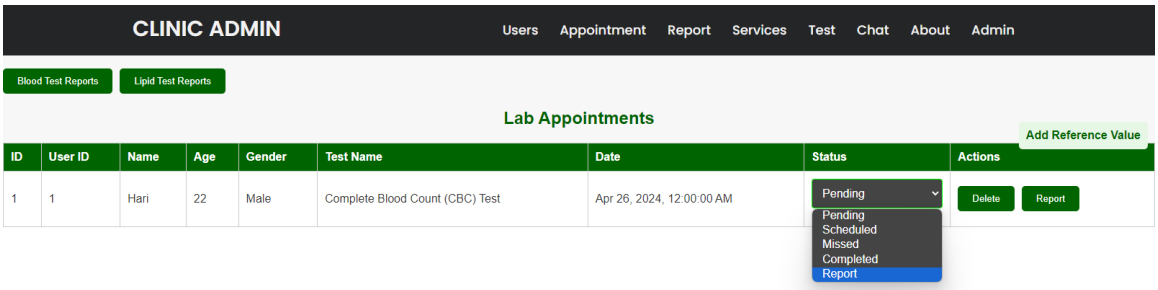
```
    if (err) {
        console.error('Error connecting to MySQL: ' + err.stack);
        return;    }
    console.log('Connected to MySQL as id ' + connection.threadId);
});
module.exports = connection;
```

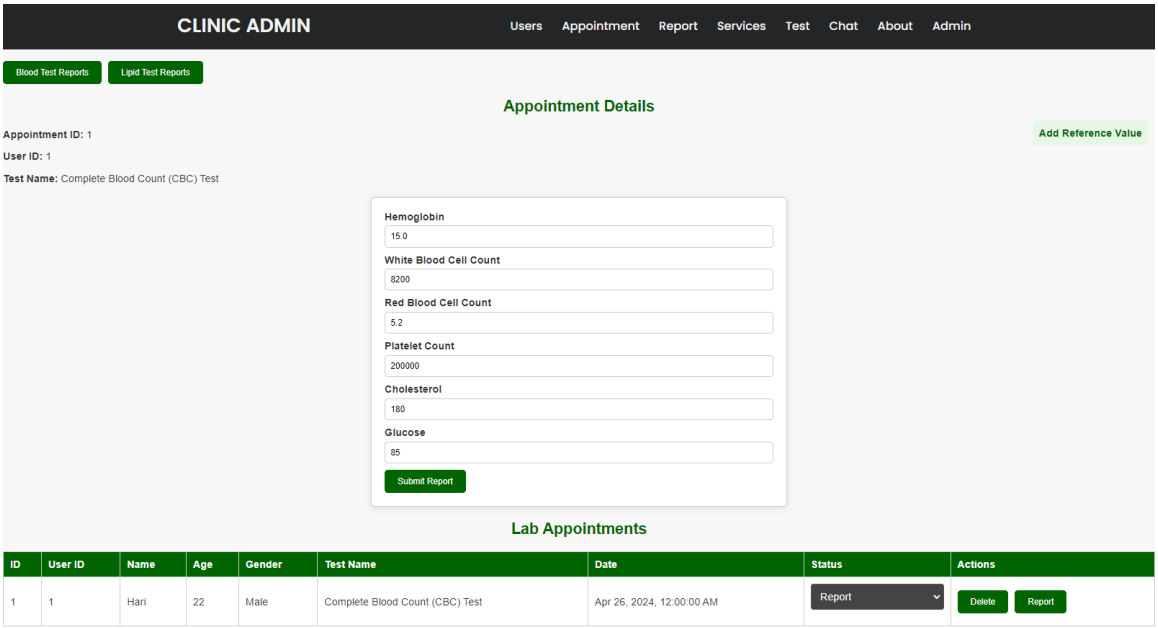
B. SCREENSHOTS

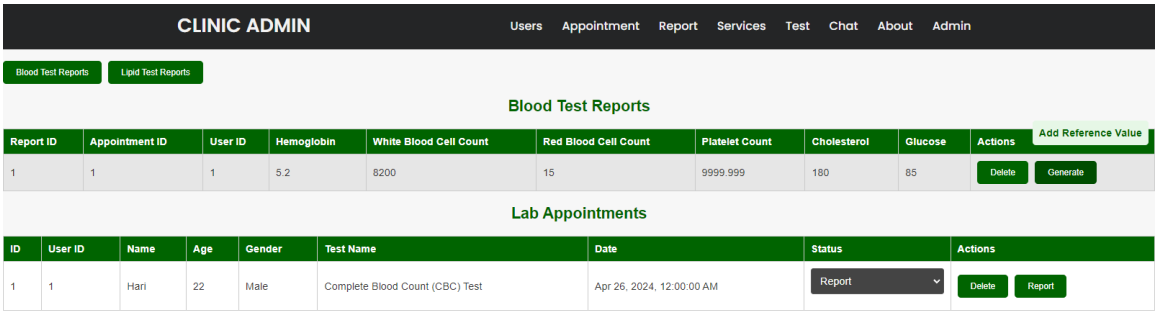
Admin

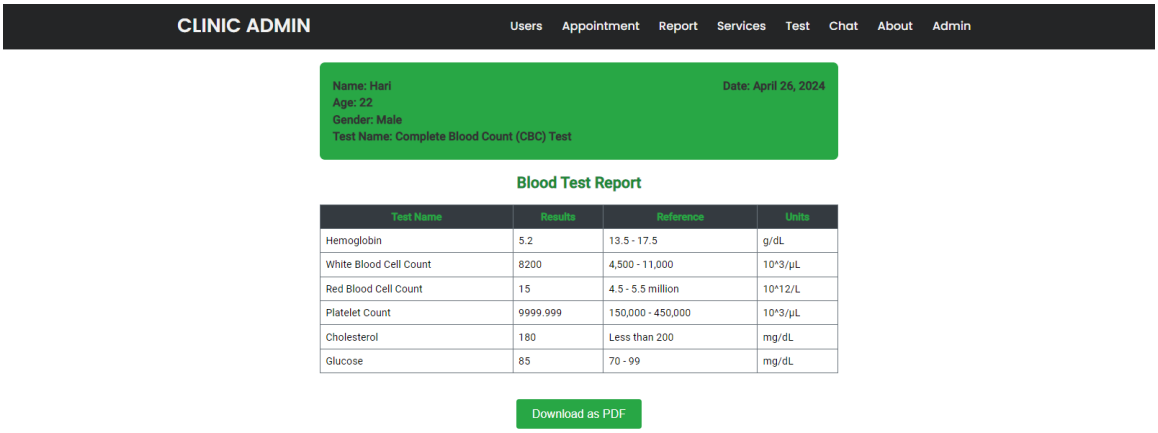


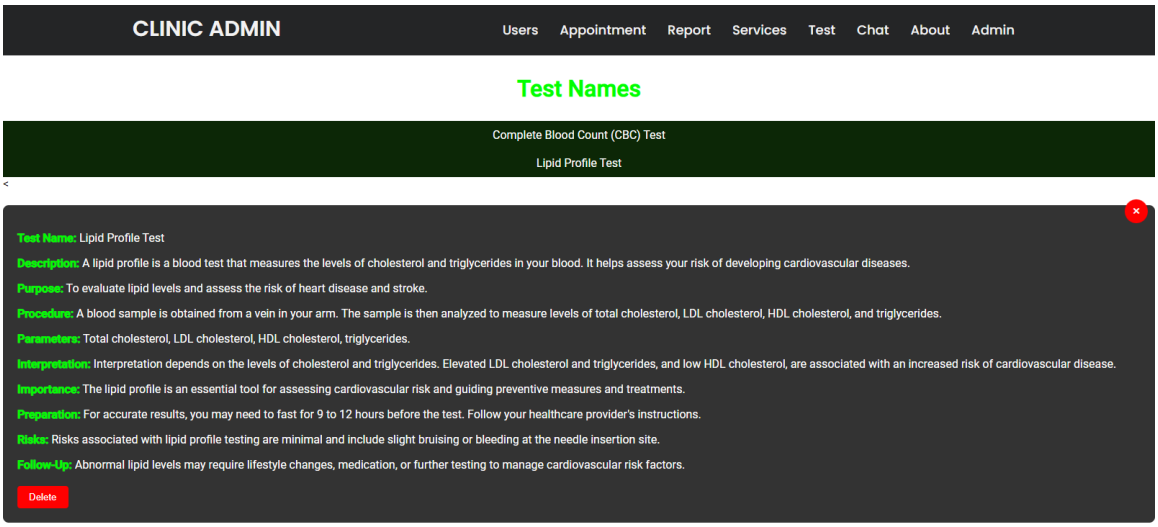

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Users
Hari
Email: hari@gmail.com
Address: Mumbai
Is Admin: No
Admin
Email: admin@gmail.com
Address: Chennai
Is Admin: Yes

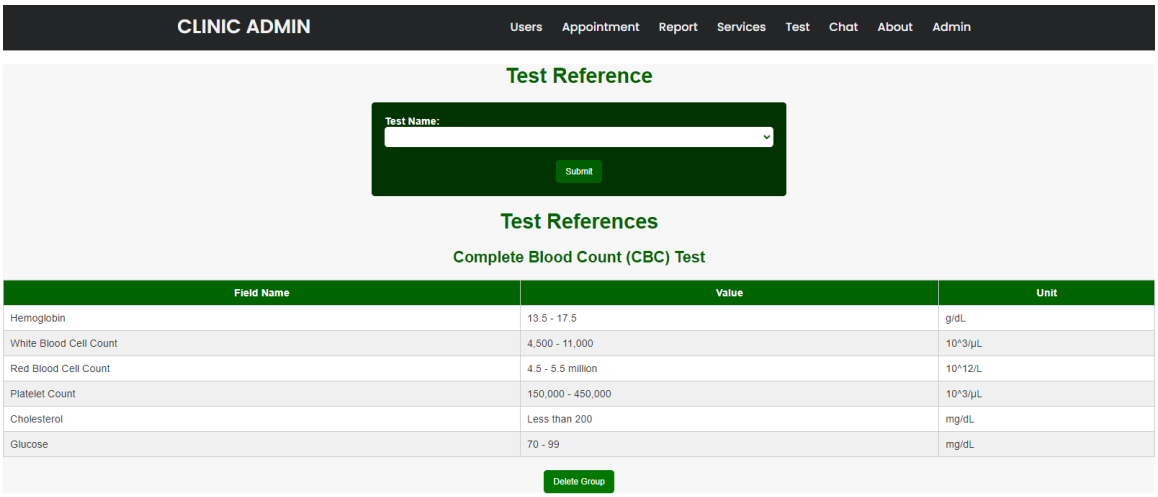

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Blood Test Reports
Lipid Test Reports
Lab Appointments
Add Reference Value
ID
User ID
Name
Age
Gender
Test Name
Date
Status
Actions
1
1
Hari
22
Male
Complete Blood Count (CBC) Test
Apr 26, 2024, 12:00:00 AM
Pending
Pending
Scheduled
Missed
Completed
Report
Delete
Report

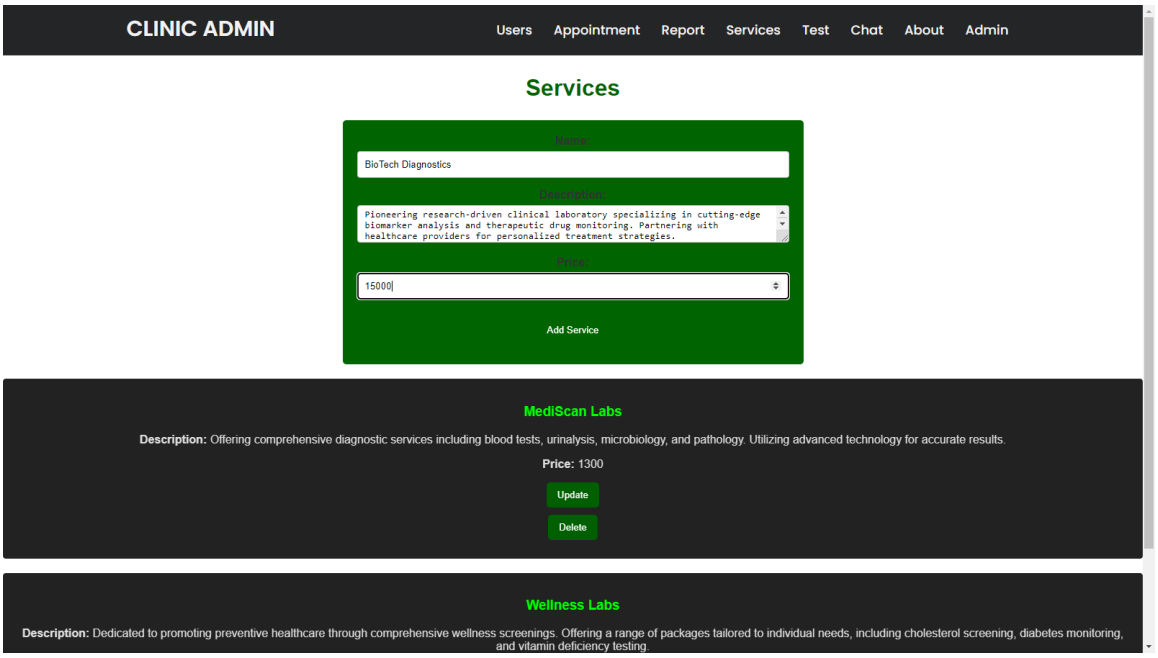

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Blood Test Reports
Lipid Test Reports
Appointment Details
Appointment ID: 1
User ID: 1
Test Name: Complete Blood Count (CBC) Test
Add Reference Value
Hemoglobin
15.0
White Blood Cell Count
8200
Red Blood Cell Count
5.2
Platelet Count
200000
Cholesterol
180
Glucose
85
Submit Report
Lab Appointments
ID
User ID
Name
Age
Gender
Test Name
Date
Status
Actions
1
1
Hari
22
Male
Complete Blood Count (CBC) Test
Apr 26, 2024, 12:00:00 AM
Report
Delete
Report


CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Blood Test Reports
Lipid Test Reports
Blood Test Reports
Add Reference Value
Report ID
Appointment ID
User ID
Hemoglobin
White Blood Cell Count
Red Blood Cell Count
Platelet Count
Cholesterol
Glucose
Actions
1
1
1
5.2
8200
15
9999.999
180
85
Delete
Generate
Lab Appointments
ID
User ID
Name
Age
Gender
Test Name
Date
Status
Actions
1
1
Hari
22
Male
Complete Blood Count (CBC) Test
Apr 26, 2024, 12:00:00 AM
Report
Delete
Report

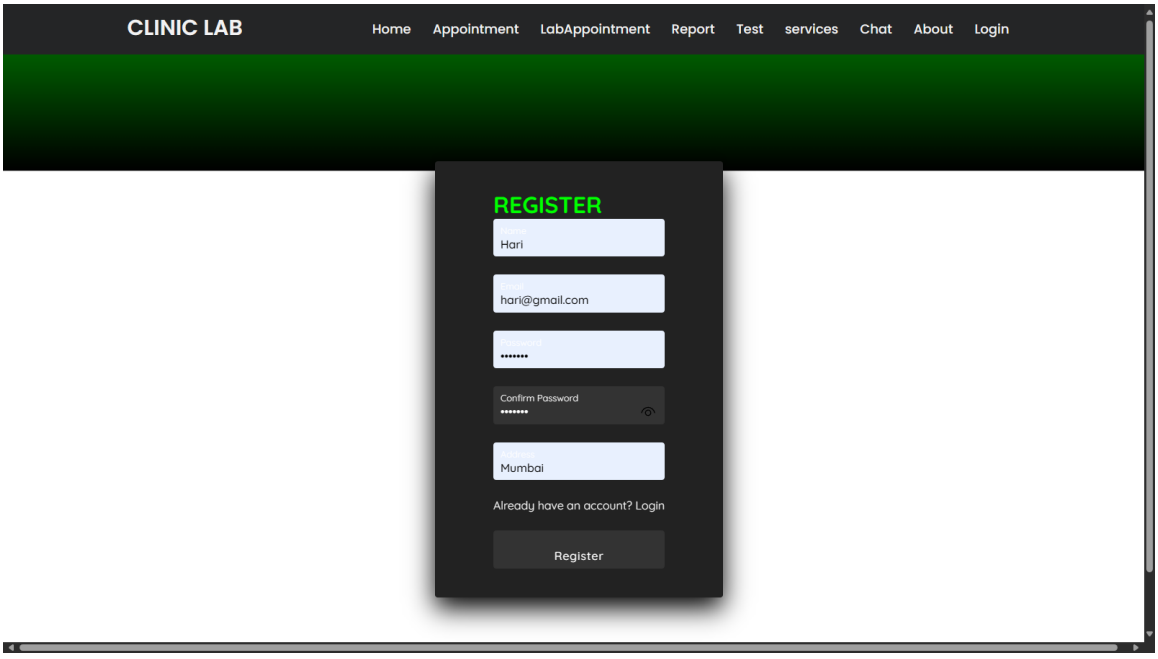

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Name: Hari
Age: 22
Gender: Male
Test Name: Complete Blood Count (CBC) Test
Date: April 26, 2024
Blood Test Report
Test Name
Results
Reference
Units
Hemoglobin
5.2
13.5 - 17.5
g/dL
White Blood Cell Count
8200
4,500 - 11,000
10^3/µL
Red Blood Cell Count
15
4.5 - 5.5 million
10^12/L
Platelet Count
9999.999
150,000 - 450,000
10^3/µL
Cholesterol
180
Less than 200
mg/dL
Glucose
85
70 - 99
mg/dL
Download as PDF

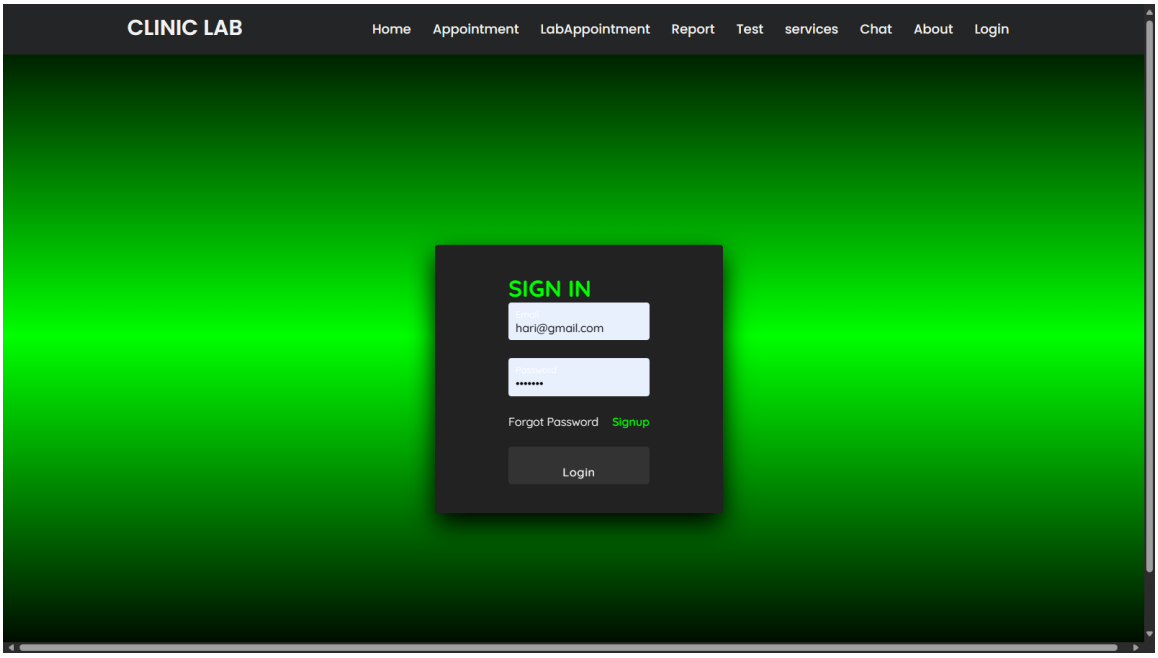

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Test Names
Complete Blood Count (CBC) Test
Lipid Profile Test
<
Test Name: Lipid Profile Test
Description: A lipid profile is a blood test that measures the levels of cholesterol and triglycerides in your blood. It helps assess your risk of developing cardiovascular diseases.
Purpose: To evaluate lipid levels and assess the risk of heart disease and stroke.
Procedure: A blood sample is obtained from a vein in your arm. The sample is then analyzed to measure levels of total cholesterol, LDL cholesterol, HDL cholesterol, and triglycerides.
Parameters: Total cholesterol, LDL cholesterol, HDL cholesterol, triglycerides.
Interpretation: Interpretation depends on the levels of cholesterol and triglycerides. Elevated LDL cholesterol and triglycerides, and low HDL cholesterol, are associated with an increased risk of cardiovascular disease.
Importance: The lipid profile is an essential tool for assessing cardiovascular risk and guiding preventive measures and treatments.
Preparation: For accurate results, you may need to fast for 9 to 12 hours before the test. Follow your healthcare provider's instructions.
Risks: Risks associated with lipid profile testing are minimal and include slight bruising or bleeding at the needle insertion site.
Follow-Up: Abnormal lipid levels may require lifestyle changes, medication, or further testing to manage cardiovascular risk factors.
Delete

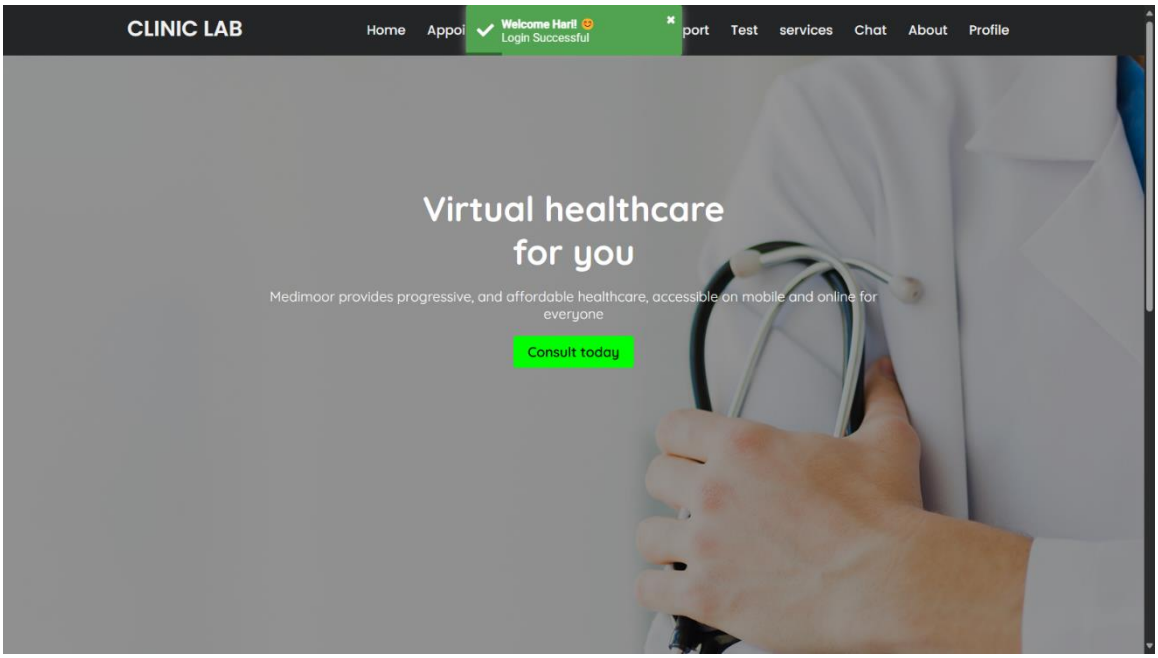

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Test Reference
Test Name:
Submit
Test References
Complete Blood Count (CBC) Test
Field Name
Value
Unit
Hemoglobin
13.5 - 17.5
g/dL
White Blood Cell Count
4,500 - 11,000
10^3/µL
Red Blood Cell Count
4.5 - 5.5 million
10^12/L
Platelet Count
150,000 - 450,000
10^3/µL
Cholesterol
Less than 200
mg/dL
Glucose
70 - 99
mg/dL
Delete Group

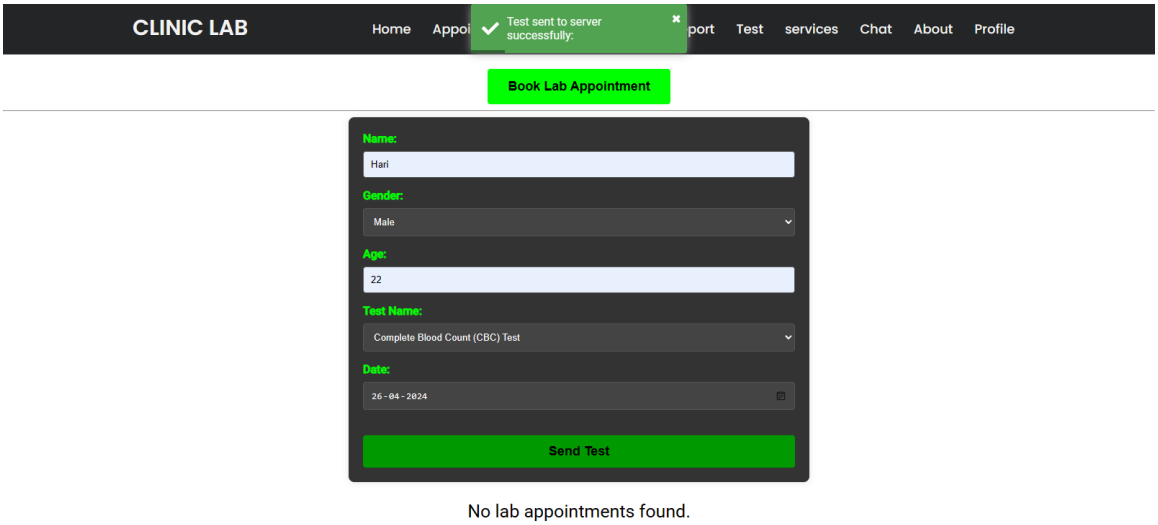

CLINIC ADMIN
Users
Appointment
Report
Services
Test
Chat
About
Admin
Services
Name
BioTech Diagnostics
Description
Pioneering research-driven clinical laboratory specializing in cutting-edge biomarker analysis and therapeutic drug monitoring. Partnering with healthcare providers for personalized treatment strategies.
Price
15000
Add Service
MediScan Labs
Description: Offering comprehensive diagnostic services including blood tests, urinalysis, microbiology, and pathology. Utilizing advanced technology for accurate results.
Price: 1300
Update
Delete
Wellness Labs
Description: Dedicated to promoting preventive healthcare through comprehensive wellness screenings. Offering a range of packages tailored to individual needs, including cholesterol screening, diabetes monitoring, and vitamin deficiency testing.

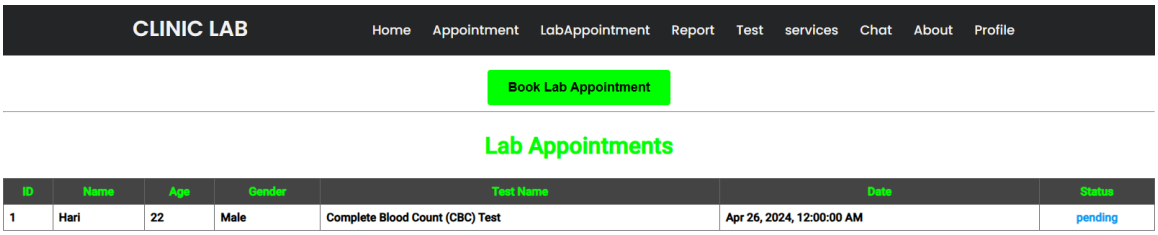
User


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Chat
About
Login
REGISTER
Name
Hari
Email
hari@gmail.com
Password
Confirm Password
Address
Mumbai
Already have an account? Login
Register


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Chat
About
Login
SIGN IN
hari@gmail.com
Forgot Password
Signup
Login


CLINIC LAB
Home
Welcome Hari!
Login Successful
Test
services
Chat
About
Profile
Virtual healthcare for you
Medimoor provides progressive, and affordable healthcare, accessible on mobile and online for everyone
Consult today


CLINIC LAB
Home
Appoi
Test sent to server successfully:
port
Test
services
Chat
About
Profile
Book Lab Appointment
Name:
Hari
Gender:
Male
Age:
22
Test Name:
Complete Blood Count (CBC) Test
Date:
26-04-2024
Send Test
No lab appointments found.


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Chat
About
Profile
Book Lab Appointment
Lab Appointments
ID
Name
Age
Gender
Test Name
Date
Status
1
Hari
22
Male
Complete Blood Count (CBC) Test
Apr 26, 2024, 12:00:00 AM
pending

Lab Report

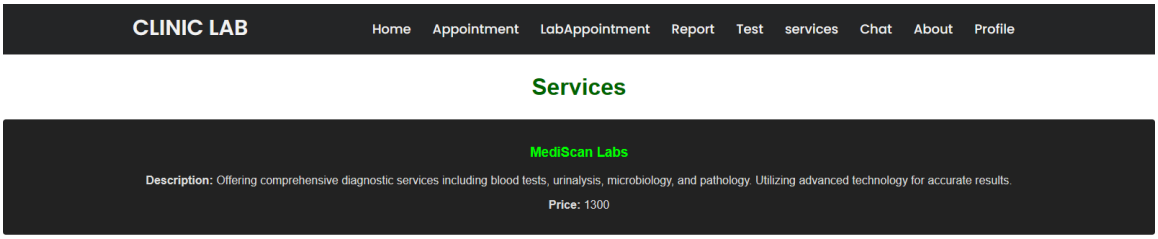
Name: Hari
Age: 22
Gender: Male
Test Name: Complete Blood Count (CBC) Test

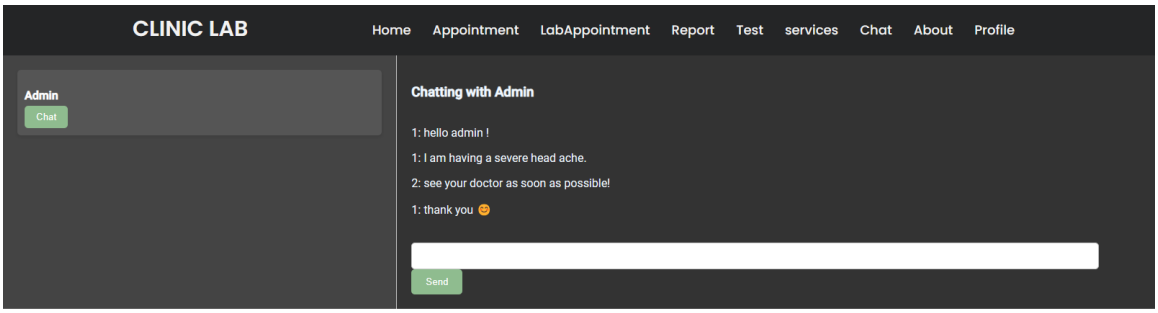
Date: April 26, 2024

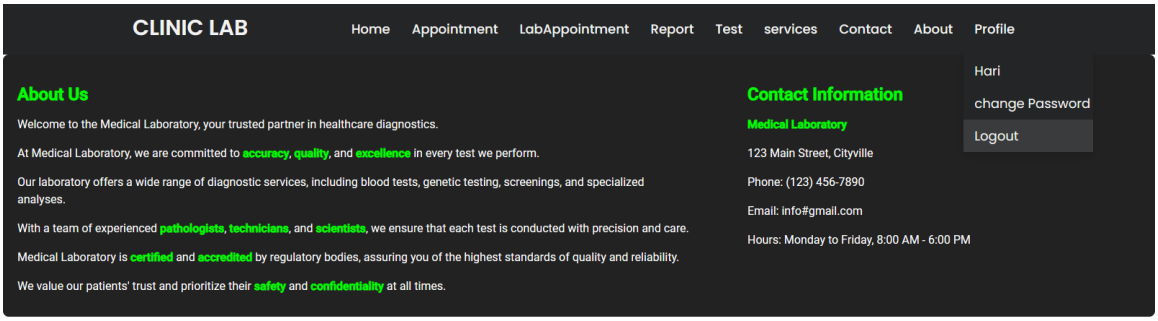
Blood Test Report

Test Name	Results	Reference	Units
Hemoglobin	5.2	13.5 - 17.5	g/dL
White Blood Cell Count	8200	4,500 - 11,000	10^3/µL
Red Blood Cell Count	15	4.5 - 5.5 million	10^12/L
Platelet Count	9999.999	150,000 - 450,000	10^3/µL
Cholesterol	180	Less than 200	mg/dL
Glucose	85	70 - 99	mg/dL

Download as PDF


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Chat
About
Profile
Services
MediScan Labs
Description: Offering comprehensive diagnostic services including blood tests, urinalysis, microbiology, and pathology. Utilizing advanced technology for accurate results.
Price: 1300
Wellness Labs
Description: Dedicated to promoting preventive healthcare through comprehensive wellness screenings. Offering a range of packages tailored to individual needs, including cholesterol screening, diabetes monitoring, and vitamin deficiency testing.
Price: 7500
BioTech Diagnostics
Description: Pioneering research-driven clinical laboratory specializing in cutting-edge biomarker analysis and therapeutic drug monitoring. Partnering with healthcare providers for personalized treatment strategies.
Price: 15000


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Chat
About
Profile
Admin
Chat
Chatting with Admin
1: hello admin !
1: I am having a severe head ache.
2: see your doctor as soon as possible!
1: thank you
Send


CLINIC LAB
Home
Appointment
LabAppointment
Report
Test
services
Contact
About
Profile
Hari
change Password
Logout
About Us
Welcome to the Medical Laboratory, your trusted partner in healthcare diagnostics.
At Medical Laboratory, we are committed to accuracy, quality, and excellence in every test we perform.
Our laboratory offers a wide range of diagnostic services, including blood tests, genetic testing, screenings, and specialized analyses.
With a team of experienced pathologists, technicians, and scientists, we ensure that each test is conducted with precision and care.
Medical Laboratory is certified and accredited by regulatory bodies, assuring you of the highest standards of quality and reliability.
We value our patients' trust and prioritize their safety and confidentiality at all times.
Contact Information
Medical Laboratory
123 Main Street, Cityville
Phone: (123) 456-7890
Email: info#gmail.com
Hours: Monday to Friday, 8:00 AM - 6:00 PM

BIBLIOGRAPHY

- Angular: Up and Running: Learning Angular, Step by Step 1st Edition
- by Shyam Seshadri
- ng-book: The Complete Guide to Angular
- by Nathan Murray, Felipe Coury , Ari Lerner , Carlos Taborda
- Node.js v21.0.0 documentation
- Head First JavaScript Programming: A Brain-Friendly Guide 1st Edition
- by Eric T. Freeman, Elisabeth Robson
- HTML, CSS, and JavaScript:
- W3Schools. HTML Tutorial. https://www.w3schools.com/html/default.asp
- W3Schools. CSS Tutorial. https://www.w3schools.com/css/default.asp
- W3Schools. JavaScript Tutorial. https://www.w3schools.com/js/default.asp
- Angular and Node.js Documentation:
- Angular. Angular Documentation. https://angular.io/docs
- Node.js. Node.js Documentation.