

# HOSPITAL MANAGEMENT

## CHAPTER - 1

### INTRODUCTION

Doctor Registration System is an online application designed to facilitate the registration of doctors. This system allows doctors to register by providing their personal and professional details. Once registered, doctors can log in to the system using their email address and password. The system provides doctors with a platform to showcase their expertise, experience, and skills to potential patients.

The Hospital Management System (HMS) is a comprehensive software solution designed to manage and automate the clinical, administrative, and financial operations of healthcare facilities. The primary objective of this system is to streamline the day-to-day operations of a hospital or clinic, improve patient care, reduce workload on staff, and enhance efficiency.

#### **Features:**

- 1. Patient management:** This module helps in managing patients' information like registration, appointment booking, consultation history, lab reports, medication details, etc.
- 2. Staff management:** This module manages doctors, nurses, and other staff members' schedules, duty hours, salaries, and performance records.
- 3. Electronic medical records (EMR):** This module manages the electronic medical records of patients, including medical histories, diagnosis reports, and laboratory test results.
- 4. Appointment scheduling:** This module streamlines the process of scheduling appointments for patients, reducing wait times and improving patient satisfaction.
- 5. Reports and analytics:** This module generates reports on various hospital operations and provides insights into key performance indicators.

## **1.1 Purpose**

This software will help the company to be more efficient in registration of their patients and manage appointments, records of patients. It enables doctors and admin to view and modify appointments schedules if required. The purpose of this project is to computerize all details regarding patient details and hospital details.

## **1.2 Scope**

The system will be used as the application that serves hospitals, clinic, dispensaries or other health institutions. The intention of the system is to increase the number of patients that can be treated and managed properly.

If the hospital management system is file based, management of the hospital has to put much effort on securing the files. They can be easily damaged by fire, insects and natural disasters. Also could be misplaced by losing data and information.

## **CHAPTER - 2**

### **SOFTWARE REQUIREMENT SPECIFICATION**

#### **2.1 Product Perspective**

This Hospital Patient Info Management System is a self-contained system that manages activities of the hospital. Due to improperly managed details medical center faces quite a lot of difficulties in accessing past data as well as managing present data. The fully functional automated hospital management system which will be developed through this project will eliminate the disadvantages caused by the manual system by improving the reliability, efficiency and performance. The usage of a database to store patient, employee, stock details etc. will accommodate easy access, retrieval, and search and manipulation of data. The access limitations provided through access privilege levels will enhance the security of the system. The system will facilitate concurrent access and convenient management of activities of the medical center.

#### **2.2 System Interfaces**

##### **2.2.1 User Interfaces**

This section provides a detailed description of all inputs into and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface.

- The protocol used shall be HTTP.
- The Port number used will be 8080.
- There shall be a logical address of the system in IPv4 format.

##### **2.2.2 Hardware Interfaces**

###### **Laptop/Desktop**

- PC-Purpose of this is to give information when Patients ask information about doctors, medicine available lab tests etc.

- To perform such Action it need very efficient computer otherwise due to that reason patients have to wait for a long time to get what they ask for.

### **2.2.3 Software Interfaces**

- JDK 1.8 - Java is fast, secure, and reliable. From laptops to data centers, game consoles to scientific supercomputers, cell phones to the Internet.
- Mysql server - Database connectivity and management.
- OS linux , Windows 7/8/8.1- Very user friendly and common OS.
- JRE 17 - JAVA Runtime Environment for run Java Application and System.

## **2.3 System Specifications**

### **2.3.1 H/W Requirement**

- Core i5 processor
- 2GB Ram.
- 20GB of hard disk space in terminal machines
- 500 TB hard disk space in Server Machine

### **2.3.2 S/W Requirement**

- Linux, Windows 7 or above operating system
- JRE 1.8 or above
- Mysql server

## CHAPTER -3

### SYSTEM IMPLEMENTATION

#### 3.1 FrontEnd Implementation:

FrontEnd Implementation is done by below languages and softwares:

1. **HTML (Hypertext Markup Language):** HTML is the backbone of any web page. It provides the basic structure of the web page by defining the elements within it. These elements include headings, paragraphs, images, links, and other content. HTML is used to create the foundation for the website and its components.
2. **CSS (Cascading Style Sheets):** CSS is used to style the website and make it visually appealing. It is used to define the color, layout, font, and other visual aspects of a web page. CSS allows developers to separate the presentation from the content, making it easier to update the design without affecting the content.
3. **TypeScript:** TypeScript is a superset of JavaScript that adds static typing to the language. This makes it easier to catch errors during development and maintain large codebases. TypeScript also includes features such as interfaces, classes, and modules that are not available in standard JavaScript.
4. **PrimeNG:** PrimeNG is a UI component library for Angular-based applications. It provides a set of reusable UI components such as buttons, forms, menus, and tables that can be easily integrated into an Angular project. PrimeNG saves time and effort by providing pre-built components that can be customized according to the project requirements.

In summary, HTML defines the structure of the web page, CSS styles the web page, TypeScript adds type safety to JavaScript, and PrimeNG provides pre-built UI components for Angular projects. Together, these technologies form the foundation of a modern front-end web development project.

### 3.2 BackEnd Implementation:

BackEnd Implementation done by using below language and softwares:

1. **Java:** Java is a popular programming language that is widely used for developing various applications including web applications. In this project, we have used Java as our primary programming language for developing the backend logic.
2. **Spring Boot API:** Spring Boot is an open-source framework that provides a way to create stand-alone Spring-based applications quickly and easily. It simplifies the development process by providing prebuilt configurations, components, and modules that can be easily configured to build a full-fledged production-ready application. In this project, we have used the Spring Boot API to develop the RESTful endpoints that are used to interact with the MySQL database.
3. **MySQL server:** MySQL is a widely used open-source relational database management system. It is known for its performance, reliability, and scalability. In this project, we have used MySQL server as our database management system to store and manage the data required by the application.

By using these technologies together, we were able to develop a robust and scalable backend system that provides high-performance APIs for interacting with the MySQL database. The use of Spring Boot ensured that the development process was streamlined and efficient while also ensuring that the code was maintainable and extendable. Overall, the use of these technologies has enabled us to deliver a high-quality software product that meets the requirements of our stakeholders.

**Database Schema:** The database schema consists of a single table called doctorRegistration. This table stores the personal and professional details of registered doctors.

### 3.2.1 Creating Tables in MySql:

#### Table Structure:

Column Name	Data Type	Constraints	Description
firstName	varchar(45)	NOT NULL	First name of the doctor
lastName	varchar(45)	NOT NULL	Last name of the doctor
gender	varchar(45)	NOT NULL	Gender of the doctor
specialization	varchar(45)	NOT NULL	Specialization of the doctor
area	varchar(45)	NOT NULL	Area of practice of the doctor
email	varchar(45)	NOT NULL	Email address of the doctor (UNIQUE)
dPassword	varchar(45)	NOT NULL	Password of the doctor
experience	int	NOT NULL	Years of experience of the doctor
PhoneNo	bigint	DEFAULT NULL	Phone number of the doctor

**Table - 1 Name:** Registration

**Description:** This table is used to store registrations of doctor's and patient's who register on the platform.

#### Columns:

- **firstName:** A string representing the first name of the doctor. It must not be NULL.
- **lastName:** A string representing the last name of the doctor. It must not be NULL.
- **gender:** A string representing the gender of the doctor. It must not be NULL.
- **specialization:** A string representing the specialization of the doctor. It must not be NULL.
- **area:** A string representing the area of the doctor. It must not be NULL.
- **email:** A string representing the email of the doctor. It must not be NULL.
- **dpassword:** A string representing the dpassword of the doctor. It must not be NULL.

- **experience:** A string representing the experience of the doctor. It must not be NULL.
- **PhoneNo:** An int representing the phoneno of the doctor. It must not be NULL.
- **appointmentId:** A string representing the appointmentId of the patient. It must not be NULL.

#### **Constraints:**

1. **NOT NULL:** Constraint for the columns “firstName, lastName, gender, specialization, area, email, appointment, dPassword, experience, and PhoneNo”. This means that these columns must have a value and cannot be left empty.
2. **PRIMARY KEY:** Constraint on the email and appointId column. This means that each row in the table must have a unique “email address & appointment”, and this column will be used as the primary key of the table.
3. **UNIQUE KEY:** Constraint on the email column. This is an additional constraint to ensure that the email addresses are unique in the table.
4. **int:** Data type constraint for the experience column. This means that only integer values can be stored in this column.
5. **bigint:** Data type constraint for the “PhoneNo” column. This means that only big integer values can be stored in this column.

This table can be used to store the registration information of doctors and can be integrated with other functionalities of the Doctor Registration System.

#### **Insert Statements**

SQL INSERT statements that are used to insert data into a table named "doctorRegistration." Each statement inserts information about a different doctor, including their unique ID (which begins with "D"), first name, last name, gender, specialty, city, email address, password, experience in years, and phone number.



#	firstName	lastName	gender	specialization	area	email	dPassword	experience	PhoneNo
1	Ananda Ku...	Pingali	Male	Cardiologist	Vizag	anandakumarpingali	AnPi@119	16	9855545068
2	Aprajita	Pandey	Female	Paediatrician	Vizianagaram	aprajitapandey@gmail.com	ApPa@112	8	8555768198
3	Arun Kumar	Agrawal	Male	General Surgen	Vizag	arunkumaragrawal@gmail.com	ArAg@105	5	8555674412
4	Basant Kumar	Agnihotri	Male	General Surgen	Vizag	basantkumaragnihotri	BaAg@103	4	8555357187
5	Bharathi Ku...	Reddy	Female	Paediatrician	Vizag	bharathikumarireddy@gmail....	BaRe@124	15	8555525038
6	Dinesh	Arya	Male	Gynaecologist	Vizianagaram	dinesharya@gmail.com	DiAr@129	11	7555407908
7	Durgesh	Pawar	Male	Child Psychiatry	Vizianagaram	durgeshpawar@gmail.com	DuPa@131	21	8555439657
8	Ghazala	Khan	Male	Child Psychiatry	Vizag	ghazalakhan@gmail.com	GhKh@135	11	7555556078
9	Hemanth K...	Kakuli	Male	Cardiologist	Vizag	hemanthkumarkakuli	HeKa@116	12	7555439248
10	Jigyasa	Khesh	Male	Dental Surgeon	Vizianagaram	jigyasakhesh@gmail.com	JiKh@110	7	9555535193
11	Jyoti	Jethani	Female	Child Psychiatry	Vizianagaram	jyothijethani@gmail.com	JyJe@134	10	7555027244
12	Kahkashan	Khan	Male	Dental Surgen	Vizag	kahkashankhan@gmail.com	KaKh@109	6	8555991645
13	Lakshmi Ku...	Jansila	Female	Cardiologist	Vizianagaram	lakshmikumarijansila@gmail....	LaJa@117	13	7555247901
14	Madhusudhan	Rao	Male	Paediatrician	Vizianagaram	madhusudhanrao@gmail.com	MaRa@125	25	9655549621
15	Mahendra Giri	Bevara	Male	Paediatrician	Vizag	mahendragiribevvari@gmail.c...	MaBe@114	8	8555724169
16	Narender	Aggarwal	Male	Gynaecologist	Vizag	narendraaggarwal	NaAg@128	15	9655549621
17	Narmadha	Sika	Female	Gynaecologist	Vizianagaram	narmadhasika@gmail.com	NaSi@126	30	9255524782

**Fig 3 Registration of doctors in this platform**

**Table - 2 Name:** patientRegistration

**Description:** This table is used to store information about patients who register on the platform.

**Columns:**

- **firstName:** A string representing the first name of the patient. It must not be NULL.
- **lastName:** A string representing the last name of the patient. It must not be NULL.
- **gender:** A string representing the gender of the patient. It must not be NULL.
- **age:** An integer representing the age of the patient. It must not be NULL.
- **area:** A string representing the area where the patient resides. It must not be NULL.
- **email:** A string representing the email address of the patient. It must not be NULL and should be unique.
- **pPassword:** A string representing the password of the patient's account. It must not be NULL.
- **pPhoneNo:** A bigint representing the phone number of the patient. It must not be NULL.
- **appointmentId:** A string representing the appointmentId of the patient. It must not be NULL.

### Constraints:

1. **NOT NULL constraint:** The columns firstName, lastName, gender, age, area, email, appointment, pPassword, and pPhoneNo are all marked as NOT NULL. This means that they cannot contain null or empty values, ensuring that each row contains complete information about a patient.
2. **PRIMARY KEY constraint:** The PRIMARY KEY constraint is set on the email & appointment column. This ensures that each email value in the table is unique and serves as the primary identifier for each row in the table.
3. **UNIQUE constraint:** An additional UNIQUE KEY constraint is also added to ensure that no two rows have the same email address. This constraint is called email\_UNIQUE.

### Table - 3 Name: appointment

**Description:** This table stores information about appointments between doctors and patients.

### Columns:

- **demail:** A non-null varchar(45) that represents the email address of the doctor.
- **pemail:** A non-null varchar(45) that represents the email address of the patient.
- **date:** A datetime column that represents the date of the appointment. This column allows null values, which means that an appointment may not have a specific date assigned yet.
- **slot:** A varchar(45) column that represents the time slot of the appointment. This column allows null values, which means that an appointment may not have a specific time slot assigned yet.
- **appointmentId:** A string representing the appointmentId of the patient. It must not be NULL.

### Constraints:

- **PRIMARY KEY:** The demail, pemail and appointment columns create a composite primary key, which ensures that each combination of doctor and patient is unique.

- **NOT NULL:** Both demail, pemail and appointment columns are marked as NOT NULL, so they must have a value for every row in the table. This constraint ensures that the appointment records always have valid doctor and patient email addresses. Additionally, there are no unique constraints defined in this table.

### 3.2.2 RegistrationForm by using Java & Spring JDBC API:-

#### Introduction

The goal of this project is to create a registration for login access to read the information of the site. This system will be implemented using Java programming language and Spring framework.

Components used to build RegistrationForm Page:

#### 1. PatientInfoPojo Class

- **package com.springapi.pojo :** This line indicates the package name for the class PatientInfoPojo. Here, it belongs to the com.springapi.pojo package.
- **public class PatientInfoPojo{ :** This line declares a public class named PatientInfoPojo.
- **private String firstName :** This line declares a private instance variable named firstName of type String.
- **private String lastName :** This line declares a private instance variable named lastName of type String.
- **private String gender :** This line declares a private instance variable named gender of type String.
- **private String area :** This line declares a private instance variable named area of type String.
- **private String email :** This line declares a private instance variable named email of type String.
- **private String pPassword :** This line declares a private instance variable named pPassword of type String.
- **private int age :** This line declares a private instance variable named age of type int.

- **private String pPhoneNo** : This line declares a private instance variable named pPhoneNo of type String.
- **private String role** : This line declares a private instance variable named role of type String.
- **public String getFirstName(){** : This line declares a public getter method for the firstName instance variable.
- **return firstName** : This line returns the value of the firstName instance variable.
- **public void setFirstName(String firstName){** : This line declares a public setter method for the firstName instance variable.
- **this.firstName = firstName** : This line sets the value of the firstName instance variable equal to the parameter passed to the method.
- **public String getLastName(){** : This line declares a public getter method for the lastName instance variable.
- **return lastName** : This line returns the value of the lastName instance variable.
- **public void setLastName(String lastName){** : This line declares a public setter method for the lastName instance variable.
- **this.lastName = lastName** : This line sets the value of the lastName instance variable equal to the parameter passed to the method.
- **public String getGender(){** : This line declares a public getter method for the gender instance variable.
- **return gender** : This line returns the value of the gender instance variable.
- **public void setGender(String gender){** : This line declares a public setter method for the gender instance variable.
- **this.gender = gender** : This line sets the value of the gender instance variable equal to the parameter passed to the method.
- **public String getArea(){** : This line declares a public getter method for the area instance variable.
- **return area** : This line returns the value of the area instance variable.
- **public void setArea(String area){** : This line declares a public setter method for the area instance variable.

- **this.area = area** : This line sets the value of the area instance variable equal to the parameter passed to the method.
- **public String getEmail(){** : This line declares a public getter method for the email instance variable.
- **return email;** This line returns the value of the email instance variable.
- **public void setEmail(String email) {** This line declares a public setter method for the email instance variable.
- **this.email = email;** This line sets the value of the email instance variable equal to the parameter passed to the method.
- **public String getpPassword(){** : This line declares a public getter method for the pPassword instance variable.
- **return pPassword** : This line returns the value of the pPassword instance variable.
- **public void setpPassword(String pPassword) {** This line declares a public setter method for the pPassword instance variable.
- **this.pPassword = pPassword** : This line sets the value of the pPassword instance variable equal to the parameter passed to the method.
- **public int getAge() {** This line declares a public getter method for the age instance variable.
- **return age** : This line returns the value of the age instance variable.
- **public void setAge(int age) {** This line declares a public setter method for the age instance variable.
- **this.age = age** : This line sets the value of the age instance variable equal to the parameter passed to the method.
- **public String getpPhoneNo(){** This line declares a public getter method for the pPhoneNo instance variable.
- **return pPhoneNo** : This line returns the value of the pPhoneNo instance variable.

## 2. Service Class

- **package com.springapi.service;** - This declares the package name for this Java class.
- **import java.util.ArrayList;** - This imports the ArrayList class from the Java Utility package, which will be used to store patient information.

- **import java.util.HashMap;** - This imports the HashMap class from the Java Utility package, which will be used to store key-value pairs of patient information.
- **import java.util.List;** - This imports the List interface from the Java Utility package, which will be used to store collections of patient information.
- **import java.util.Map;** - This imports the Map interface from the Java Utility package, which will be used to store mappings of patient information.
- **import org.springframework.beans.factory.annotation.Autowired;** - This imports the Autowired annotation, which will be used for dependency injection in Spring.
- **import org.springframework.jdbc.core.JdbcTemplate;** - This imports the JdbcTemplate class from the Spring JDBC package, which provides a simplified API for working with SQL databases.
- **import org.springframework.stereotype.Service;** - This imports the Service annotation from Spring, which indicates that this class is a service component.
- **@Service** - This annotation marks the class as a service component, which can be autowired into other components.
- **@Autowired** - This annotation injects an instance of the JdbcTemplate into the class.
- **public int insertDoctorInfo(PatientInfoPojo obj) {** - This method takes a PatientInfoPojo object as input and inserts its properties into the 'patientRegistration' table in the database.
- **String firstName = obj.getFirstName();** - This line retrieves the first name of the patient from the PatientInfoPojo object.
- **String lastName = obj.getLastName();** - This line retrieves the last name of the patient from the PatientInfoPojo object.
- **String pPassword = obj.getpPassword();** - This line retrieves the password of the patient from the PatientInfoPojo object.
- **String gender = obj.getGender();** - This line retrieves the gender of the patient from the PatientInfoPojo object.
- **String area = obj.getArea();** - This line retrieves the area of the patient from the PatientInfoPojo object.
- **String email = obj.getEmail();** - This line retrieves the email of the patient from the PatientInfoPojo object.

- **String pPhoneNo = obj.getpPhoneNo();** - This line retrieves the phone number of the patient from the PatientInfoPojo object.
- **String role=obj.getRole();** - This line retrieves the role of the patient from the PatientInfoPojo object.
- **int age = obj.getAge();** - This line retrieves the age of the patient from the PatientInfoPojo object.
- **int rows = 0;** - This initializes a variable to store the number of rows affected by the SQL insert statement.
- **String query = "insert into patientRegistration values(?,?,?,?,?,?,?,?)";** - This creates an SQL insert statement for inserting patient information into the 'patientRegistration' table.
- **List previous = this.getPatientInfoByMail(email);** - This calls the getPatientInfoByMail() method to check if a patient with the same email already exists in the database.
- **if(previous.isEmpty()) {** - This checks if the previous List is empty, which means that there are no existing patients with the same email.
- **rows = jdbc.update(query, firstName, lastName, gender, age, area, email, pPassword, pPhoneNo,role);** - This executes the SQL insert statement and returns the number of rows affected.
- **else {** - If a patient with the same email already exists in the database, this code block will be executed.
- **rows = 0;** - This sets the number of rows affected to zero, indicating that the SQL insert statement was not executed.
- **}** - This closes the else code block.
- **return rows;** - This returns the number of rows affected by the SQL insert statement.
- **public List<Map<String,Object>> getDoctorInfo() {** - This method retrieves all doctors' information from the 'patientRegistration' table in the database.
- **String query = "select \* from patientRegistration where role='doctor'";** - This creates an SQL select statement for retrieving all doctor's information from the 'patientRegistration' table.

- **return jdbc.queryForList(query);** - This executes the SQL select statement and returns a List of Maps containing each

### 3. Controller Class

- **Package Declaration:** The first line of code is a package declaration that indicates the name of the package where this class resides.
- **Import Statements:** This section includes all the import statements used in the class to include external classes and interfaces.
- **@RestController Annotation:** This annotation declares the class as a Rest Controller. It indicates that the output of the methods will be sent back to the client in the format of JSON or XML.
- **@CrossOrigin Annotation:** This annotation enables cross-origin resource sharing (CORS) requests from any domain to access your API.
- **Class Declaration:** This line of code declares the class name.
- **@Autowired Annotation:** This annotation injects the service layer bean instance into the controller layer.
- **@GetMapping Annotation:** This annotation maps HTTP GET requests onto specific handler methods. In this case, it maps the “/patientInfo/sA” URL path to the getAllData() method.
- **getAllData() Method:** This method retrieves all the patient records from the database using the PatientInfoService's getPatientInfo() method and returns the data in the form of a List of Map<String,Object>.
- **@PostMapping Annotation:** This annotation maps HTTP POST requests onto specific handler methods. In this case, it maps the "/patientInfo/iData" URL path to the insertPatientData() method.
- **insertPatientData() Method:** This method receives the request body as a PatientInfoPojo object, which contains the patient information data. It then inserts the patient information data into the database using the PatientInfoService's insertDoctorInfo() method. If the data insertion is successful, it returns a string message "inserted," else it returns "not inserted."



### 3.2.2.1 LoginForm by using Java & Spring Boot API:-

#### Introduction

The goal of this project is to login to the user profile page that allows doctors and patients to retrieve the information. This system will be implemented using Java programming language and Spring framework.

Components used to build LoginForm Page:

#### 1. PatientInfoPojo Class

Here's a step-by-step explanation of the LogininfoPojo class:

- **package com.project.pojo :** This line specifies the package name in which the LoginPojo class belongs.
- **public class LoginPojo{ :** This line defines the LoginPojo class and indicates that it is a public class that can be accessed from other packages.
- **private String email :** This line declares a private instance variable called 'email' of type String, which represents the user's email address.
- **private String password :** This line declares a private instance variable called 'password' of type String, which represents the user's login password.
- **private String role :** This line declares a private instance variable called 'role' of type String, which represents the user's role in the system (e.g., admin, regular user).
- **public String getRole(){ :** This line defines a public method called 'getRole()' that returns the value of the 'role' instance variable.
- **return role :** This line returns the value of the 'role' instance variable.
- **public void setRole(String role){ :** This line defines a public method called 'setRole()' that sets the value of the 'role' instance variable to the given argument.
- **this.role = role :** This line assigns the value of the 'role' parameter to the 'role' instance variable using the 'this' keyword, which refers to the current object.
- **public String getEmail(){ :** This line defines a public method called 'getEmail()' that returns the value of the 'email' instance variable.
- **return email :** This line returns the value of the 'email' instance variable.

- **public void setEmail(String email){** : This line defines a public method called 'setEmail()' that sets the value of the 'email' instance variable to the given argument.
- **this.email = email** : This line assigns the value of the 'email' parameter to the 'email' instance variable using the 'this' keyword.
- **public String getPassword(){** : This line defines a public method called 'getPassword()' that returns the value of the 'password' instance variable.
- **return password** : This line returns the value of the 'password' instance variable.
- **public void setPassword(String password){** : This line defines a public method called 'setPassword()' that sets the value of the 'password' instance variable to the given argument.
- **this.password = password** : This line assigns the value of the 'password' parameter to the 'password' instance variable using the 'this' keyword.

## 2. Service Class

Here's a step-by-step explanation of the LogintService class:

- **package com.project.service;** - The standard package declaration for a Java class.
- **import java.util.List;** - Importing the List interface from the java.util package to use it later in the code.
- **import java.util.Map;** - Importing the Map interface from the java.util package to use it later in the code.
- **import org.springframework.beans.factory.annotation.Autowired;** - Importing the Autowired annotation from the Spring framework to enable automatic dependency injection.
- **import org.springframework.jdbc.core.JdbcTemplate;** - Importing the JdbcTemplate class from the Spring JDBC module to use it for database operations.
- **import org.springframework.stereotype.Service;** - Importing the Service annotation from the Spring framework to indicate that this class is a service component.
- **import com.project.pojo.LoginPojo;** - Importing the LoginPojo class from the project's pojo package to use it for authentication.
- **@Service** - An annotation that marks this class as a service component that can be automatically detected and used by other components.

- **public class LoginService {** - A public class declaration for the LoginService.
- **@Autowired** - An annotation that injects the JdbcTemplate instance into this class automatically.
- **JdbcTemplate jt;** - A private instance field of type JdbcTemplate used to perform database operations.
- **public String verifyCredentials(LoginPojo lp) {** - A public method that takes a LoginPojo object and returns a string value.
- **String email = lp.getEmail();** - Extracting the email address from the LoginPojo object.
- **String password = lp.getPassword();** - Extracting the password from the LoginPojo object.
- **String role=lp.getRole();** - Extracting the role from the LoginPojo object.
- **if(role.equals("doctor"))** - A conditional statement that checks if the user is a doctor.  
String query = "SELECT \* FROM patientRegistration WHERE email = ? AND
- **pPassword = ? AND role='doctor'** - A SQL query string that selects all the rows from the patientRegistration table where the email and password match and the role is a doctor.
- **List details = jt.queryForList(query, email, password);** - Executing the query and storing the result in a list of maps.
- **if(details.isEmpty())** - A conditional statement that checks if the list of maps is empty.
- **return "Invalid User";** - Returning an error message if the authentication fails.
- **else** - An else block that executes if the authentication succeeds.
- **return "Login Successful";** - Returning a success message after successful authentication.
- **else if(role.equals("patient"))** - A conditional statement that checks if the user is a patient.
- **String query = "SELECT \* FROM patientRegistration WHERE email = ? AND pPassword = ? AND role='patient'"** - A SQL query string that selects all the rows from the patientRegistration table where the email and password match and the role is a patient.
- **List details = jt.queryForList(query, email, password);** - Executing the query and storing the result in a list of maps.
- **if(details.isEmpty())** - A conditional statement that checks if the list of maps is empty.

- **return "Invalid User";** - Returning an error message if the authentication fails.
- **else** - An else block that executes if the authentication succeeds.
- **return "Login Successful";** - Returning a success message after successful authentication.
- **return "ok";** - A default return statement that returns "ok" if no condition matches.

### 3. Controller Class

Here's a step-by-step explanation of the LoginPageController class:

- The LoginController class is annotated with `@RestController`, which indicates that it is a Spring MVC Controller and will handle HTTP requests.
- The `@Autowired` annotation is used to inject an instance of the LoginService into the controller.
- The `@PostMapping` annotation is used to map the `verifyCredentials` method to handle HTTP POST requests to the `"/login"` endpoint.
- The `verifyCredentials` method takes a LoginPojo object as a parameter, which is passed in as the request body.
- Inside the `verifyCredentials` method, the LoginService's `verifyCredentials` method is called, passing in the LoginPojo object.
- The `verifyCredentials` method then returns the result returned by the LoginService's `verifyCredentials` method as a string response from the controller.

In summary, when an HTTP POST request is sent to the `"/login"` endpoint, the LoginController receives a LoginPojo object as the request body, passes it to the LoginService's `verifyCredentials` method for validation, and returns the result as a string response.

### 3.2.2.2 ForgetPassword page by using Java & Spring Boot API:

#### Introduction

The goal of this project is to create a new password when the user forgets his password that helps the user to access the profile back. This system will be implemented using Java programming language and Spring framework.

Components used to build ForgetPassword Page:

#### 1. PojoClass:-

Here's a step-by-step explanation of the ForgetPasswordPojo class:

- **package com.project.pojo :** This line declares the package for the Java class.
- **public class ForgetPasswordPojo{ :** This line declares a public class named `ForgetPasswordPojo`. The class has default visibility meaning it can be accessed by classes in the same package.
- **private String email :** This line declares a private instance variable named `email` of type `String`.
- **private String password :** This line declares a private instance variable named `password` of type `String`.
- **private String role :** This line declares a private instance variable named `role` of type `String`.
- **public String getRole() { :** This line declares a public method named `getRole()` that returns a `String` value.
- **return role :** This line returns the value of the `role` instance variable when the `getRole()` method is called.
- **public void setRole(String role){ :** This line declares a public method named `setRole()` that accepts a `String` parameter named `role`.
- **this.role = role:** This line assigns the value of the `role` parameter to the `role` instance variable.
- **public String getPassword(){ :** This line declares a public method named `getPassword()` that returns a `String` value.

- **public String getEmail(){** : This line declares a public method named `getEmail()` that returns a `String` value.
- **public void setEmail(String email){** : This line declares a public method named `setEmail()` that accepts a `String` parameter named `email`.
- **this.email = email** : This line assigns the value of the `email` parameter to the `email` instance variable.
- **public void setPassword(String password){** : This line declares a public method named `setPassword()` that accepts a `String` parameter named `password`.
- **this.password = password** : This line assigns the value of the `password` parameter to the `password` instance variable.

## 2. Service Class :

Here's a step-by-step explanation of the ForgetPasswordService class:

- **package com.project.service** : This is the package declaration statement that specifies the namespace where the current Java file resides.
- **import statements** : These are used to import required classes, interfaces, or packages from other namespaces.
- **@Service** : This annotation is used to indicate that this class is a service component in Spring Framework. It tells the Spring container to create an instance of this class at runtime.
- **public class ForgetPasswordService {** : This is the class name, and it is declared as public, which means it can be accessed from anywhere.
- **@Autowired** : This annotation is used to enable automatic dependency injection of the JdbcTemplate object.
- **JdbcTemplate jt** : An instance variable of the JdbcTemplate class has been declared here.
- **public List verification(ForgetPasswordPojo fp)** : This is the method signature for the verification method that takes a ForgetPasswordPojo object as its input and returns a List object.

- **String email=fp.getEmail()** : This line gets the email address from the ForgetPasswordPojo object received as input.
- **String role=fp.getRole()** : This line gets the role information from the ForgetPasswordPojo object received as input.
- **if(role.equals("doctor")){** : This line checks whether the role is equal to "doctor".
- **String sql="select email from patientRegistration where email=? AND role='doctor'"** : This line creates a SQL query to select the email address from the patientRegistration table where the email is equal to the email address received as input and the role is set to "doctor".
- **List details = jt.queryForList(sql, email)** : This line executes the above SQL query and retrieves the result set from the database.
- **Map temp = (Map) details.get(0)** : This line retrieves the first row from the result set and converts it to a Map object.
- **String str=(String)temp.get("email")** : This line gets the email address from the Map object.
- **if(str.equals(email))** : This line checks whether the email address retrieved from the database is equal to the one received as input.
- **String password=fp.getPassword()** : This line gets the new password from the ForgetPasswordPojo object received as input.
- **email=fp.getEmail()** : This line gets the email address from the ForgetPasswordPojo object received as input.
- **String query="update patientRegistration set pPassword=? where email=? AND role='doctor'"** : This line creates a SQL update query to change the password of the user with the given email address and role in the patientRegistration table.
- **jt.update(query,password,email)** : This line executes the above SQL query and updates the password in the database.
- **return details** : This line returns the List object containing the email address retrieved from the database.
- **else if(role.equals("patient")){** : This line checks whether the role is equal to "patient".
- **String sql="select email from patientRegistration where email=? AND role='patient'"** : This line creates a SQL query to select the email address from the

patientRegistration table where the email is equal to the email address received as input and the role is set to "patient".

- **List details = jt.queryForList(sql, email)** : This line executes the above SQL query and retrieves the result set from the database.
- **Map temp = (Map) details.get(0)** : This line retrieves the first row from the result set and converts it to a Map object.
- **String str=(String)temp.get("email")** : This line gets the email address from the Map object.
- **if(str.equals(email))** : This line checks whether the email address retrieved from the database is equal to the one received as input.
- **String password=fp.getPassword()** : This line gets the new password from the ForgetPasswordPojo object received as input.
- **email=fp.getEmail()** : This line gets the email address from the ForgetPasswordPojo object received as input.
- **String query="update patientRegistration set pPassword=? where email=? AND role='patient'"** : This line creates a SQL update query to change the password of the user with the given email address and role in the patientRegistration table.
- **jt.update(query,password,email)** : This line executes the above SQL query and updates the password in the database.
- **return details** : This line returns the List object containing the email address retrieved from the database.
- **return null** : This line returns null if neither "doctor" nor.

### 3. Controller Class:

Here's a step-by-step explanation of the ForgetPasswordController class:

- The ForgetPasswordController class is a Spring RestController that handles HTTP PUT requests to the "/upass" endpoint.
- The @Autowired annotation is used to inject an instance of the ForgetPasswordService into the controller.



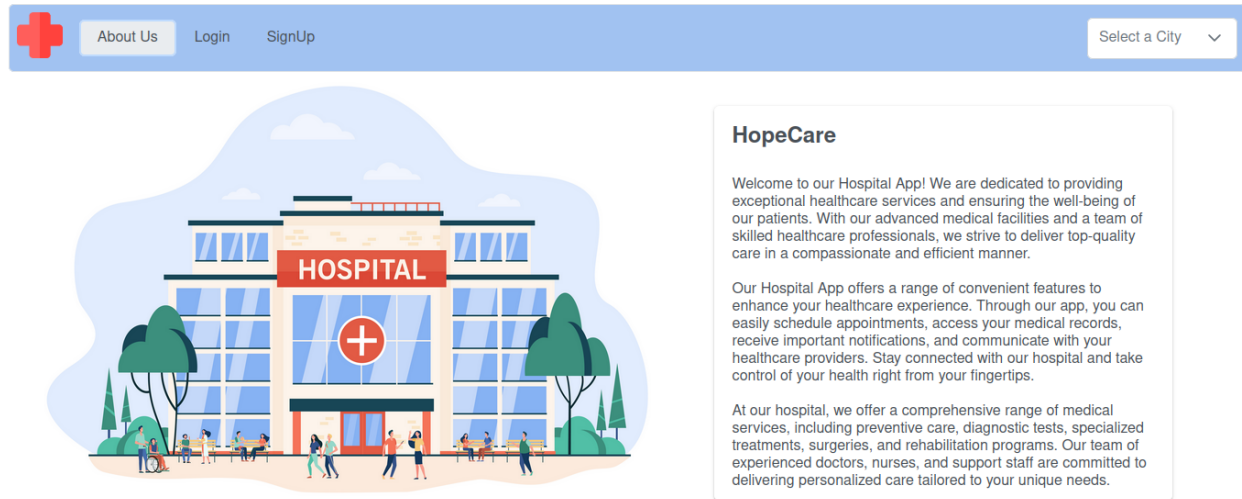
- The verification method is annotated with `@PutMapping` and takes a `ForgetPasswordPojo` object as a parameter, which is passed in as the request body.
- The verification method calls the verification method of the injected `ForgetPasswordService` instance, passing in the `ForgetPasswordPojo` object.
- The result returned by the `ForgetPasswordService`'s verification method is then returned as a `List` response from the controller.

In summary, the `ForgetPasswordController` receives a `ForgetPasswordPojo` object via a PUT request to the `"/upass"` endpoint, passes it to the `ForgetPasswordService` for verification, and returns the result as a `List` response.

## CHAPTER - 4

### SAMPLE SCREENSHOOT

Here is the Homepage of Hospital Management.

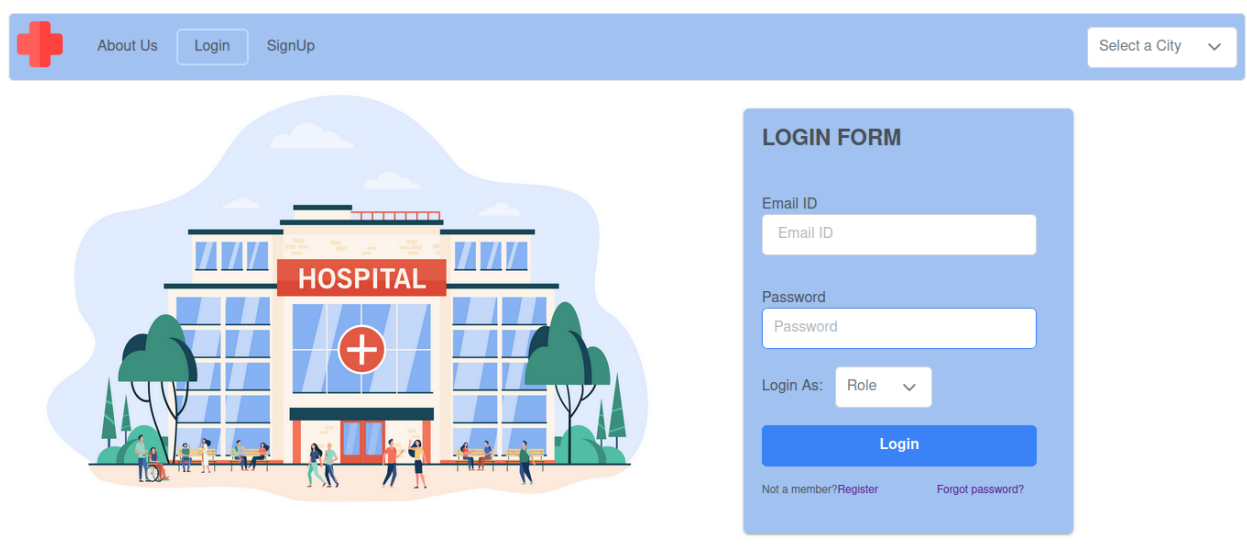


And Here is the Signup Page for new users.

The screenshot shows the signup page of the hospital management system. It has the same blue navigation bar as the homepage. The main content area features the same hospital building illustration. To the right is a blue registration form with the following fields: 'First Name:', 'Last Name:', 'Email:', 'Phone Number:', 'Age:', 'Location:', 'Password:', 'Register As:' (with a 'Role' dropdown), and 'Gender:' (with a 'Gender' dropdown). A 'Submit' button is at the bottom of the form.

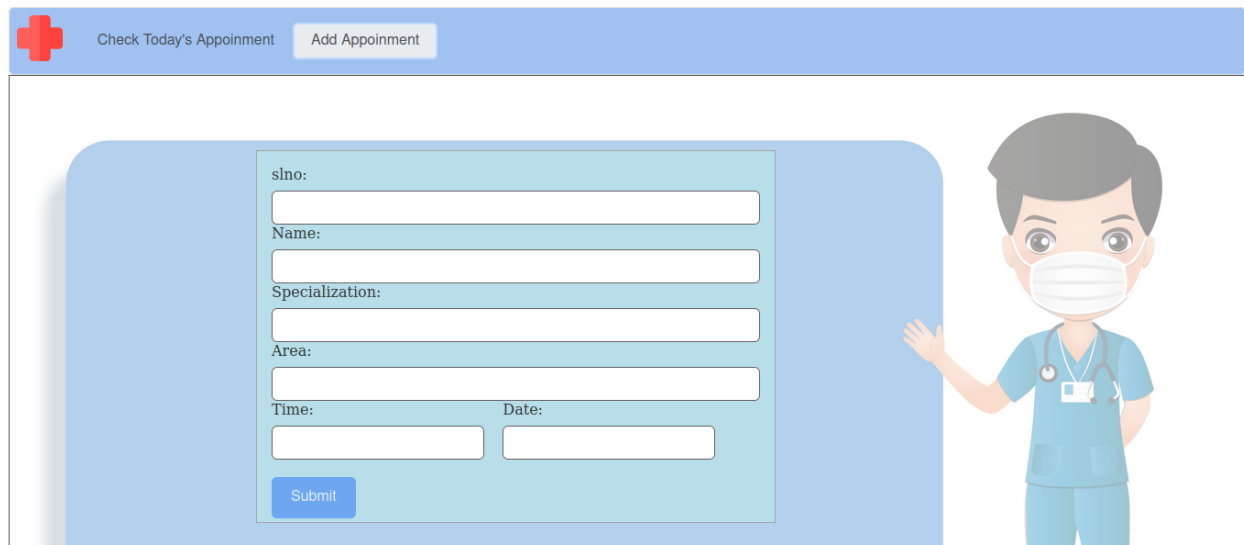
localhost:4200/main/signup

And Here is the LoginPage for user who has signed up on the website.





The screenshot shows a web application interface for a hospital. At the top, there is a blue header bar with a red cross icon on the left, followed by links for 'About Us', 'Login', and 'SignUp'. On the right side of the header is a 'Select a City' dropdown menu. The main content area is divided into two sections. On the left, there is a large, colorful illustration of a hospital building with a red cross on its facade, surrounded by trees and people. On the right, there is a 'LOGIN FORM' box. This box contains input fields for 'Email ID' and 'Password', a 'Login As:' dropdown menu with 'Role' selected, and a blue 'Login' button. Below the button are two links: 'Not a member? Register' and 'Forgot password?'.

And, Here is the appointment page for patient for taking appointment to consult doctor.



The screenshot shows a web application interface for a hospital appointment page. At the top, there is a blue header bar with a red cross icon on the left, followed by links for 'Check Today's Appointment' and 'Add Appointment'. The main content area is divided into two sections. On the left, there is a light blue box containing a form with the following fields: 'sno:', 'Name:', 'Specialization:', 'Area:', 'Time:', and 'Date:'. Each field has a corresponding input box. Below the 'Time:' and 'Date:' fields is a blue 'Submit' button. On the right side of the form box, there is a cartoon illustration of a male doctor wearing a blue scrub suit, a white face mask, and a stethoscope around his neck.

Here is the list appointment page to check list of appointment booked are booked.

 						
Sl.No	Name	Specialization	Area	Date	Time	Book