System Requirements Specification

Index

For

HealthCare Information System

Version 1.0

TABLE OF CONTENTS

HealthCare Information System

System Requirements Specification

You need to consume APIs exposed by Backend application in Angular to make application work as FULLSTACK

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

HealthCare Information System is Spring boot RESTful application with MySQL, where it manages hospitals, doctors and patient's profiles.

Following is the requirement specifications:

	HealthCare Information System
Modules	
1	Hospital
2	Doctor
3	Patient
Hospital Module Functionalities	
1	Create a Hospital
2	Update the existing hospital details
3	Get hospital info by hospital id
4	Get all registered hospitals
5	Search hospital Info by hospital name
6	Delete an existing hospital
Doctor Module Functionalities	
1	Create a doctor
2	Update the existing doctor
3	Get a doctor by Id
4	Fetch all registered doctors
5	Delete an existing doctor
6	Search doctor Info by doctor name
7	Get a doctor by hospital Id
8	Search doctor by hospital name

Patient Module Functionalities	
1	Create a patient
2	Update the existing patient
3	Get a patient by Id
4	Get a patient by doctor Id
5	Fetch all registered patients
6	Delete an existing patient
7	Search patientInfo by patient name
8	Search patientInfo by doctor id

2 Assumptions, Dependencies, Risks / Constraints

2.1 HOSPITAL CONSTRAINTS:

- While updating a hospital, if hospitalId does not exist then the operation should throw a custom exception.
- While fetching the hospital details by id, if hospitalId does not exist then the operation should throw a custom exception.
- While deleting the hospital details by id, if hospitalId does not exist then the operation should throw a custom exception.

2.2 DOCTOR CONSTRAINTS

- While updating a doctor, if doctorId does not exist then the operation should throw a custom exception.
- While fetching the doctor details by id, if doctorId does not exist then the operation should throw a custom exception.

2.3 PATIENT CONSTRAINTS

- While updating a patient, if patientId does not exist then the operation should throw a custom exception.
- While fetching the patient details by id, if patientId does not exist then the operation should throw a custom exception.

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in dto classes only.
- All the database operations must be implemented on entity object onlyDo not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 BUSINESS VALIDATIONS

3.1 Hospital Entity

- Hospital name is not null, min 3 and max 100 characters.

3.2 Doctor Entity

- Doctor name is not null, min 3 and max 100 characters.
- Hospital id is not null.

3.3 Patient Entity

- Patient name is not null, min 3 and max 100 characters.
- doctor id is not null.

4 REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

4.1 HOSPITAL CONTROLLER

URL Exposed		Purpose
1. /hospitals		Create a Hospital
Http Method	POST	
Parameter 1	Hospital	
Return	Hospital	
2. /hospitals/{id}		Update the Hospital
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	Hospital	
Return	Hospital	
3. /hospitals/{id}		Fetches the details of Hospital by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	Hospital	
4. /hospitals/{id}		Delete the Hospital
Http Method	DELETE	
Parameter 1	Long (id)	
Return	Boolean	
5. /hospitals/		Fetch all registered Hospitals
Http Method	GET	
Parameter	-	
Return	List<Hospitals>	
6. /hospitals/search?name={name}		Fetches the Hospital with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	List<Hospitals>	

4.2 DOCTOR CONTROLLER

URL Exposed		Purpose
1. /doctors/		Create a Doctor
Http Method	POST	
Parameter 1	Doctor	
Return	Doctor	
2. /doctors/{id}		Update the Doctor
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	Doctor	
Return	Doctor	
3. /doctors/{id}		Fetches the details of Doctor by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	Doctor	
4. /doctors/{id}		Delete the Doctor
Http Method	DELETE	
Parameter 1	Long (id)	
Return	Boolean	
5. /doctors/		Fetch all registered Doctors
Http Method	GET	
Parameter 1	-	
Return	List<Doctor>	
6. /doctors/search?name={name}		Fetches the Doctor with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	List<Doctor>	
7. /doctors/hospital/{hospitalid}		Fetches the Doctor with the given hospital id
Http Method	GET	
Parameter 1	Long (id)	
Return	List<Doctor>	
8. /doctors/searchByHospital?name={name}		Fetches the Doctor with the given hospital name
Http Method	GET	
Parameter 1	Long (id)	
Parameter 2	String (name)	
Return	List<Doctor>	

4.3 PATIENT CONTROLLER

URL Exposed		Purpose
1. /patients/		Create a Patient
Http Method	POST	
Parameter 1	Patient	
Return	Patient	
2. /patients/{id}		Update the Patient
Http Method	PUT	
Parameter 1	Long (id)	
Parameter 2	Patient	
Return	Patient	
3. /patients/{id}		Fetches the details of Patient by Id
Http Method	GET	
Parameter 1	Long(id)	
Return	Patient	
4. /patients/{id}		Delete the Patient
Http Method	DELETE	
Parameter 1	Long (id)	
Return	Boolean	
5. /patients/		Fetch all registered Patients
Http Method	GET	
Parameter 1	-	
Return	List<Teachers>	
6. /patients/search?name={name}		Fetches the Patient with the given name
Http Method	GET	
Parameter 1	String (name)	
Return	List<Patient>	
7. /patients/doctor/{doctorid}		Fetches the patient with the given doctor id
Http Method	GET	
Parameter 1	Long (id)	
Return	List<Patient>	
8. /patients/searchByDoctor?id={id}		Search the patient with the given doctor id
Http Method	GET	
Parameter 1	Long (id)	
Return	List<Patient>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.HEALTHCARE

Resources

HealthCareApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

5.2 PACKAGE: COM.HEALTHCARE.ENTITY

Resources

Class/Interface	Description	Status
Doctor (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **doctorId** as primary key. • Map this class with a **doctortable**. • Generate the **doctorId** using the IDENTITY strategy.	Partially implemented.
Hospital(Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **hospitalId** as primary key. • Map this class with a **hospital table**. • Generate the **hospitalId** using the IDENTITY strategy.	Partially implemented.

Patient (Class)	• This class is partially implemented. • Annotate this class with proper annotation to declare it as an entity class with **patientId** as primary key. • Map this class with a **patient table**. • Generate the **patientId** using the IDENTITY strategy.	Partially implemented.

5.3 PACKAGE: COM.HEALTHCARE.DTO

Resources

Class/Interface	Description	Status
DoctorDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
HospitalDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
PatientDTO (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

5.4 PACKAGE: COM.HEALTHCARE.REPOSITORY

Resources

Class/Interface	Description	Status
DoctorDAO (interface)	1. Repository interface exposing CRUD functionality for **Doctor** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented
HospitalDAO (interface)	1. Repository interface exposing CRUD functionality for **Hospital** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented
PatienDAO (interface)	1. Repository interface exposing CRUD functionality for **Patient** Entity. 2. You can go ahead and add any custom methods as per requirements.	Partially implemented

5.5 PACKAGE: COM.HEALTHCARE.SERVICE.IMPL

Resources

Class/Interface	Description	Status
DoctorServiceImpl (class)	• Implements DoctorService. • Contains template method implementation. • Need to provide implementation for doctor related functionalities. • Do not modify, add or delete any method signature	To be implemented.
HospitalServiceImpl (class)	• Implements HospitalService. • Contains template method implementation. • Need to provide implementation for hospital related functionalities. • Do not modify, add or delete any method signature	To be implemented.
PatientServiceImpl (class)	• Implements PatientService. • Contains template method implementation. • Need to provide implementation for patient related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.6 PACKAGE: COM.HEALTHCARE.SERVICE

Resources

Class/Interface	Description	Status
DoctorService(interface)	• Need to provide implementation for Doctor related functionalities. • Add required repository dependency. • Do not modify, add or delete any method signature.	Already implemented.
HospitalService (interface)	• Need to provide implementation for Hospital related functionalities • Add required repository dependency • Do not modify, add or delete any method signature.	Already implemented.
PatientService (class)	• Need to provide implementation for Patient related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	Already implemented.

5.7 PACKAGE: COM.HEALTHCARE.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalExceptionHandler (class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class. • Use this as a reference for creating exception handler for other custom exception classes.	Partially implemented.
ResourceNotFound Exception (Class)	• Custom Exception to be thrown when trying to fetch or delete the Hospital info which does not exist. • Need to create Exception Handler for same wherever needed (local or global).	Already created.
DoctorNotFound Exception (Class)	• Custom Exception to be thrown when trying to fetch or delete a doctor info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

HospitalNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete a hospital info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
PatientNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete a patient info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
ErrorResponse (Class)	• Object of this class is supposed to be returned in case of exception through exception handlers	Already created.

5.8 PACKAGE: COM.HEALTHCARE.CONTROLLER

Resources

Class/Interface	Description	Status
DoctorController (Class)	• Controller class to expose all rest-endpoints for Doctor related activities. • May also contain local exception handler methods.	To be implemented
HospitalController (Class)	• Controller class to expose all rest-endpoints for Hospital related activities. • May also contain local exception handler methods.	To be implemented
PatientController (Class)	• Controller class to expose all rest-endpoints for Patient related activities. • May also contain local exception handler methods.	To be implemented

6 CONSIDERATIONS

A. There is no roles in this application
B. You can perform the following 3 possible actions

Hospital
Doctor
Patient

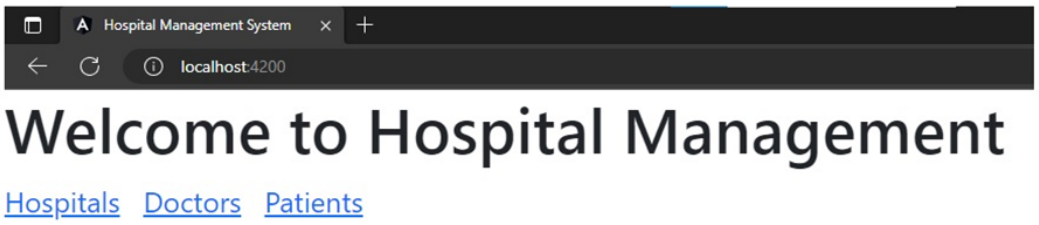
FRONTEND-ANGULAR SPA

1 Problem Statement

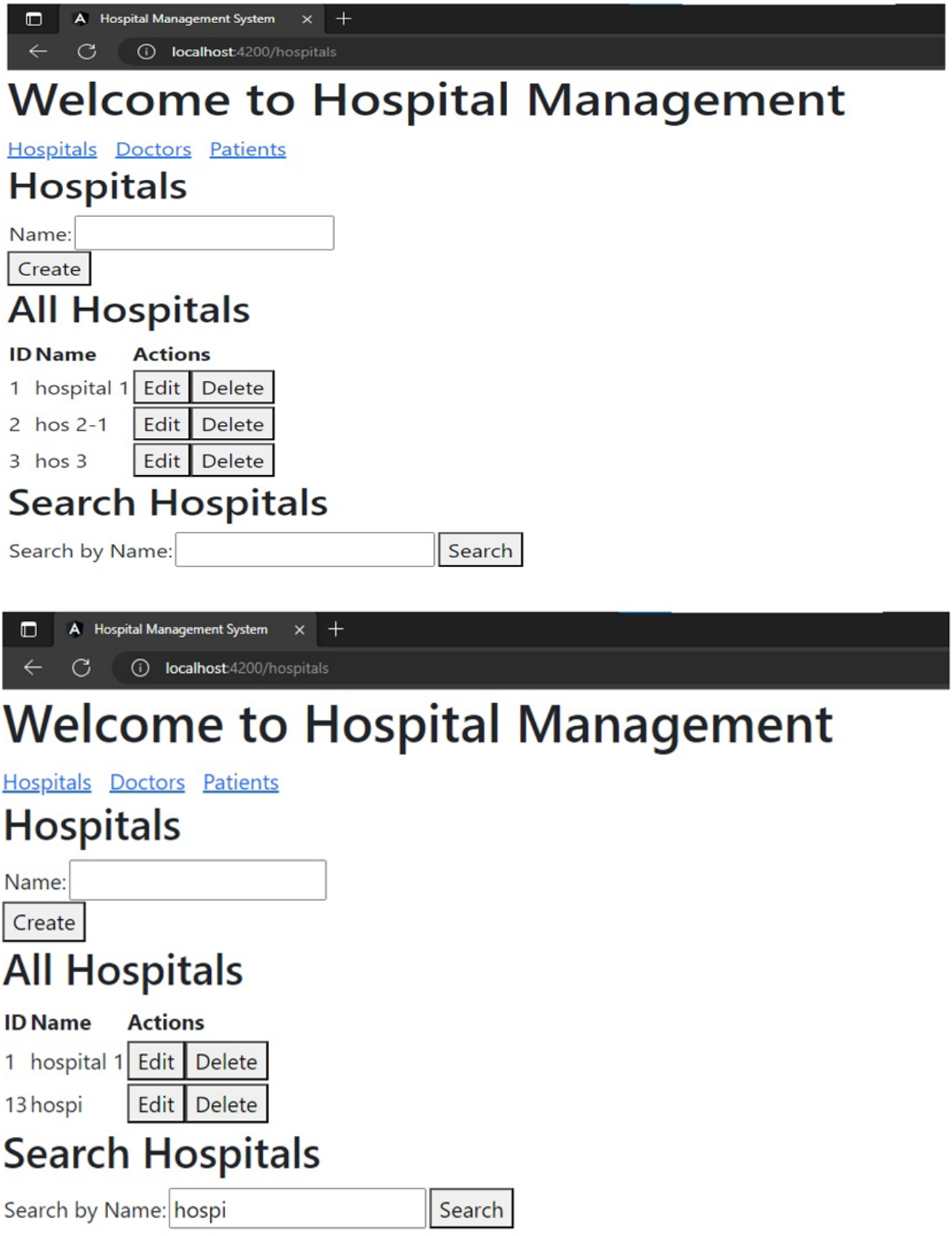
HealthCare information system is SPA (Single Page Application), it allows to manage hospitals, doctors, and patients' profiles. It helps us to view, update, delete, create and filter all modules along with the functionality to search all modules.

2 Proposed Healthcare Information System Wireframe

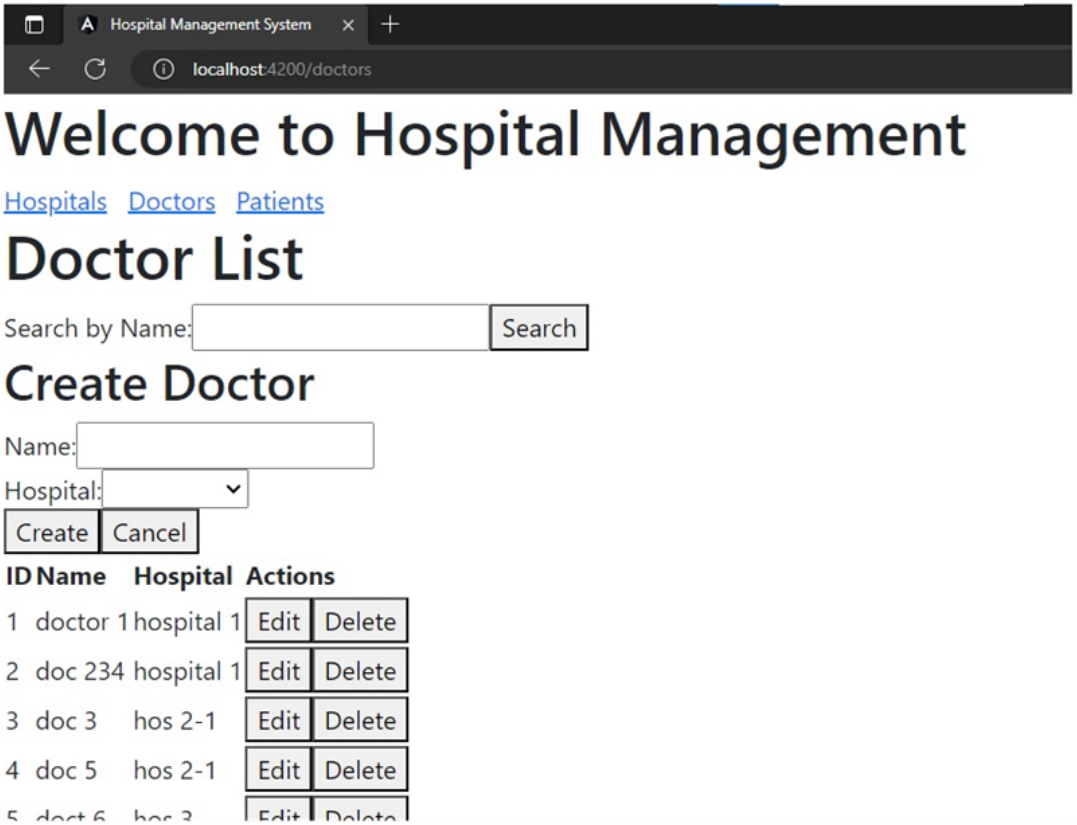
2.1 Homepage

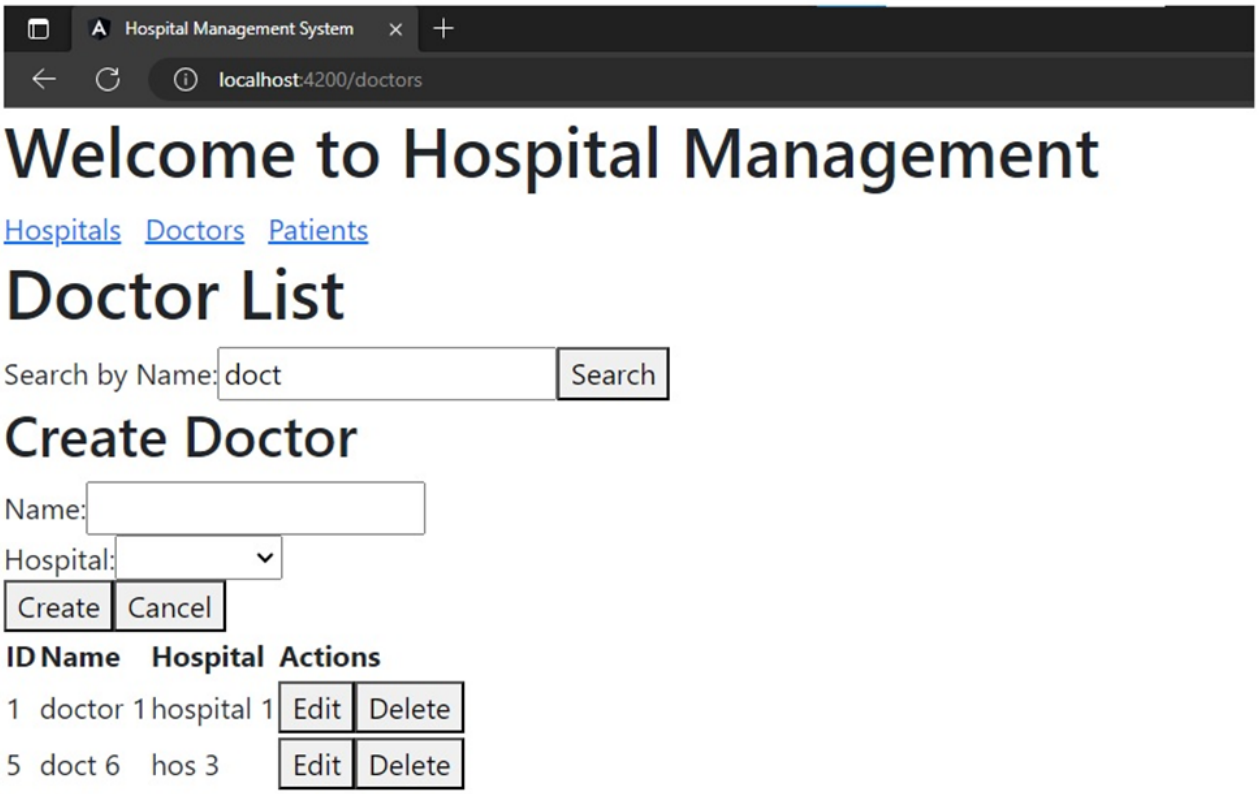


2.2 HOSPITALS PAGE

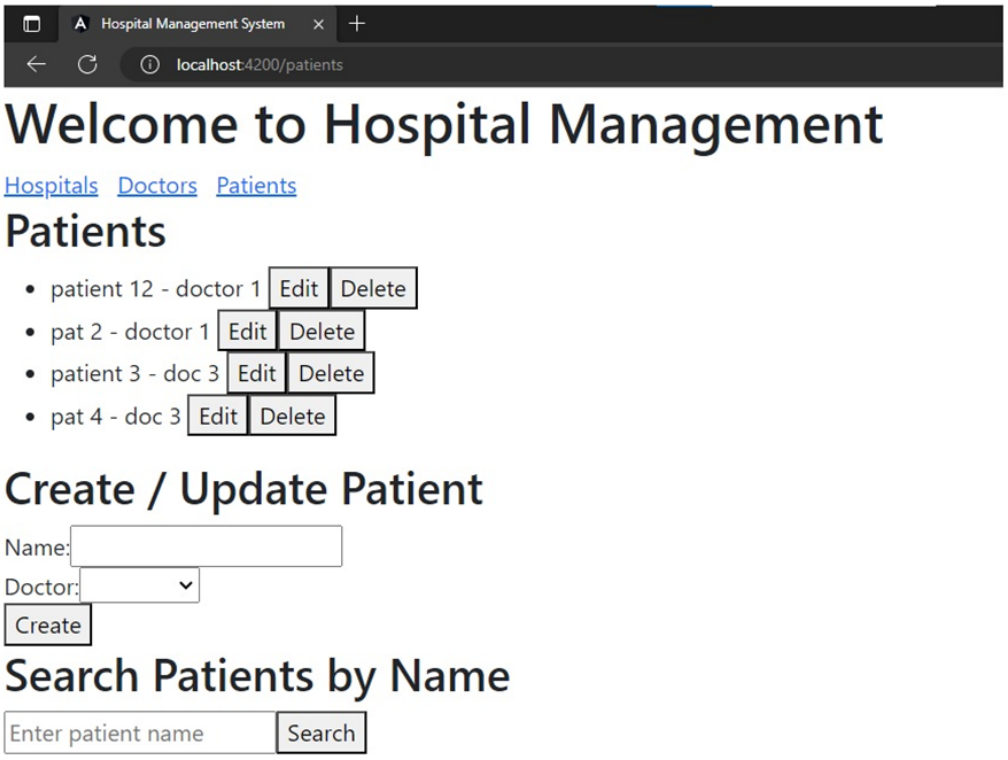


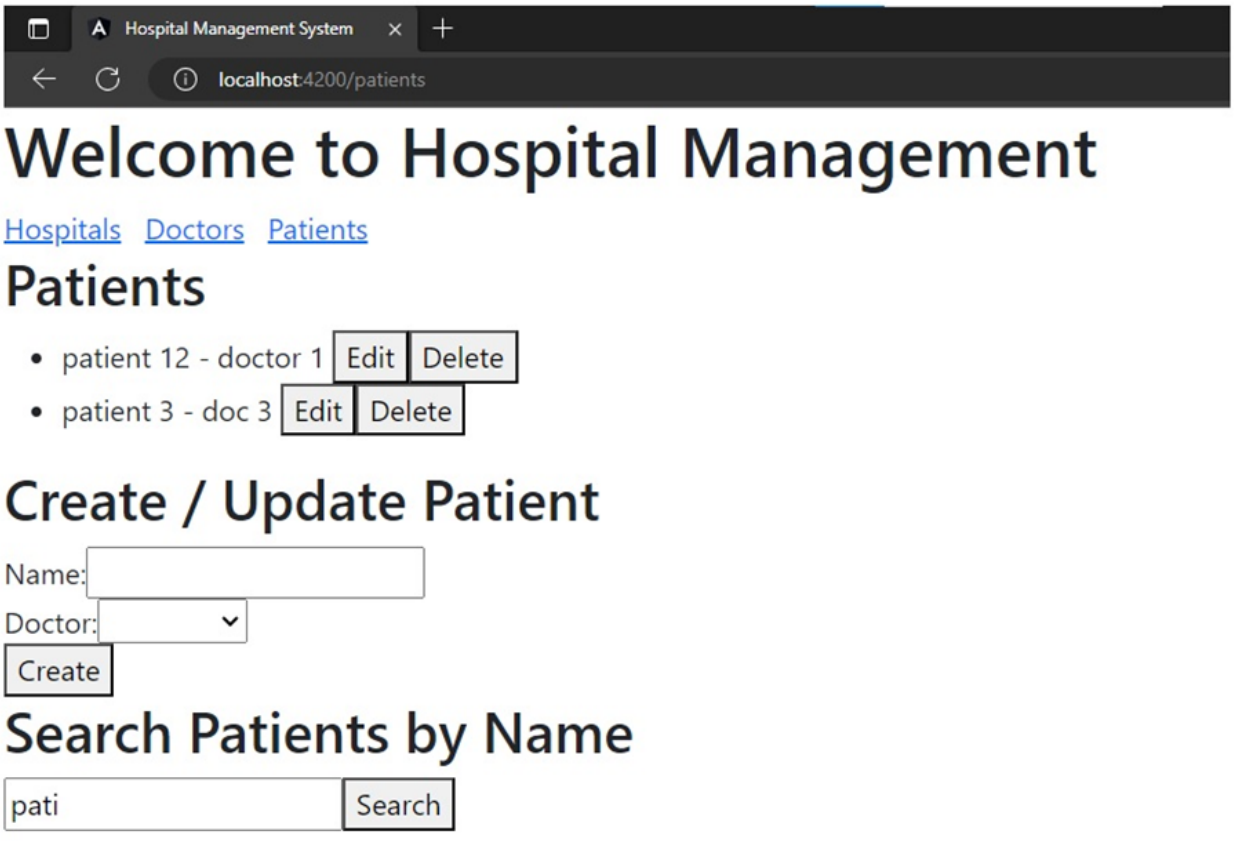
2.3 DOCTORS PAGE





2.4 PATIENTS PAGE





3 BUSINESS-REQUIREMENT:

As an application developer, develop the Healthcare Information System (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the Home page as default page. There should be heading as “Welcome to Hospital Management” and 3 hyperlinks as “Hospitals”, “Doctors” and “Patients” for routing to “/hospitals”, “/doctors” and “/patients” page respectively.
US_02	Hospitals Page	As a user I should be able to see the hospital page and perform all operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating hospital. 1.1 Name 2. Create button should be disable until name field is validated. 3. Edit and delete functionality should be there to edit and delete the hospital respectively. 4. We should be able to search the hospital on name basis and list should be updated accordingly.

US_03	Doctors Page	As a user I should be able to see the doctor page and perform all operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating doctor. 1.1 Name 1.2 Hospital 2. Create button should be disable until both name and hospital fields are selected. Note: - While creating a new doctor, a dropdown should be visible with hospital name. 3. Edit and delete functionality should be there to edit and delete the doctor respectively. 4. We should be able to search for the doctor on a name basis and the list should be updated accordingly.

US_04	Patients Page	As a user I should be able to see the patient page and perform all operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating patient. 1.1 Name 1.2 Doctor 2. Create button should be disable until both name and doctor fields are selected. Note: - While creating a new patient, a dropdown should be visible with patient name. 3. Edit and delete functionality should be there to edit and delete the patient respectively. 4. We should be able to search for the patient on a name basis and the list should be updated accordingly.

4 CONSTRAINTS

1. On the page load, input focus must come to the first name input field.
2. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

7 EXECUTION STEPS TO FOLLOW FOR BACKEND

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. cd into your backend project folder
4. To build your project and run test cases use command: mvn clean package
5. To launch your application, move into the target folder (cd target). Run the following command to run the application:
   java -jar <your application jar file name>
6. This editor Auto Saves the code.
7. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
8. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
9. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
10. To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
11. Default credentials for MySQL:
    a. Username: root
    b. Password: pass@word1
11. To login to mysql instance: Open new terminal and use following command:
    a. sudo systemctl enable mysql
    b. sudo systemctl start mysql
    c. mysql -u root -p
    The last command will ask for password which is 'pass@word1'
12. Mandatory: Before final submission run the following command:
    mvn test
13. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

8 EXECUTION STEPS TO FOLLOW FOR FRONTEND

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. **This is a web-based application, to run the application on a browser, use the internal browser in the environment.**
4. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min**
   b. **npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min**
   c. **npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**
5. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**