System Requirements Specification

Index

For

Appointment Scheduler Application

Version 1.0

TABLE OF CONTENTS

APPOINTMENT SCHEDULER APPLICATION
System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Appointment Scheduler Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for patients to book an appointment for a doctor.

Following is the requirement specifications:

		Appointment Scheduler Application
Modules		
	1	Doctor
	2	Schedule
Doctor Module Functionalities		
	1	List all doctors
	2	Get doctor by id
	3	Create doctor
	4	Update doctor by id
	5	Delete doctor by id
	6	Get doctor by speciality **(must use dynamic method)**
	7	Get schedules by doctor name and speciality **(must use custom query)**
	8	Get schedules by doctor id **(must use custom query)**
	9	Get all doctors by speciality in sorting order **(must return doctors by speciality in ascending order and that also in pages)**

Schedule Module Functionalities		
	1	Create an appointment **(must be transactional)**
	2	Update an appointment by id **(must be transactional)**
	3	Get an appointment by id
	4	Get list of all appointments for a doctor on particular day

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 DOCTOR CONSTRAINTS

- When fetching a doctor by ID, if the doctor ID does not exist, the operation should throw a not found exception.
- When updating a doctor, if the doctor ID does not exist, the operation should throw a not found exception.
- When removing a doctor, if the doctor ID does not exist, the operation should throw a not found exception.

2.2 SELL CONSTRAINTS

- When deleting a schedule by ID, if the schedule ID does not exist, the operation should throw a not found exception.
- When fetching a schedule by ID, if the schedule ID does not exist, the operation should throw a not found exception.
- When updating a schedule by ID, if the schedule ID does not exist, the operation should throw a not found 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 only
- Do 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 - Doctor

- Name is not blank.
- Hospital name should not be blank.
- Speciality names should not be blank.
- DailyTime name should not be blank.

4 BUSINESS VALIDATIONS - Schedule

- Name is not blank.
- Doctor should not be blank.
- Day should not be blank.
- Time should not be blank.
- Timings should not be blank.

5 REST ENDPOINTS

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

5.1 DOCTORCONTROLLER

URL Exposed		Purpose
1. /api/doctors		Fetches all the doctors
Http Method	GET	
Parameter	-	
Return	List<Doctor>	
2. /api/doctors/{id}		Get a doctor by id
Http Method	GET	
Parameter 1	Long (id)	
Return	Doctor	
3. /api/doctors		Create a new doctor
Http Method	POST	
Parameter	-	
Return	Doctor	
4. /api/doctors/{id}		Updates existing doctor by id
Http Method	PUT	
Parameter 1	Long (id)	
Return	Doctor	
5. /api/doctors/{id}		Deletes a doctor by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

6. /api/doctors/specialty/{specialty}		
Http Method	GET	Fetches all doctor with given specialty
Parameter 1	String (specialty)	
Return	List<Doctor>	

7. /api/doctors/schedules/byNameAndSpeciality		
Http Method	GET	Fetches all doctor's schedule by name and speciality
Parameter 1	String (doctorName and speciality)	
Return	List<Schedule>	

8. /api/doctors/schedules/{doctorId}		
Http Method	GET	Fetches a doctor's schedule by id
Parameter 1	int (doctorId)	
Return	List<Schedule>	

9. /api/doctors/sortedDoctors		
Http Method	GET	Fetches all doctor by order and in pages
Parameter 1	-	
Return	List<Doctor>	

5.2 SCHEDULCONTROLLER

URL Exposed		Purpose
1. /api/schedules/appointment		
Http Method	POST	Creates a new Schedule
Parameter	-	
Return	Schedule	
2. /api/schedules/appointment/{id}		
Http Method	PUT	Updates a schedule by id
Parameter 1	Long (id)	
Return	Schedule	
3. /api/schedules/appointment/{id}		
Http Method	GET	Fetches a schedule by id
Parameter	Long (id)	

Return	Schedule	
4. /api/schedules/doctor/{id}/{day}		Fetches the list of all schedules for a doctor by given id on given day
Http Method	GET	
Parameter	Long (id) String (day)	
Return	List<Schedule>	

6 TEMPLATE CODE STRUCTURE

6.1 PACKAGE: COM.APPOINTMENT

Resources

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

6.2 PACKAGE: COM.APPOINTMENT.REPOSITORY

Resources

Class/Interface	Description	Status
DoctorRepository (interface)	• Repository interface exposing CRUD functionality for doctor entity. • It must contain the methods for: ○ finding schedules of doctors by name & speciality ○ finding schedule by	Partially implemented.

Class/Interface	Description	Status
	○ doctor id ○ fetching all doctors in order with speciality in pages ● You can go ahead and add any custom methods as per requirements.	
ScheduleRepository (interface)	● Repository interface exposing CRUD functionality for schedule entity. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.

6.3 PACKAGE: COM.APPOINTMENT.SERVICE

Resources

Class/Interface	Description	Status
DoctorService (interface)	● Interface to expose method signatures for doctor related functionality. ● Do not modify, add or delete any method.	Already implemented.

Class/Interface	Description	Status
ScheduleService (interface)	• Interface to expose method signatures for schedule related functionality. • Do not modify, add or delete any method.	Already implemented.

6.4 PACKAGE: COM.APPOINTMENT.SERVICE.IMPL

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.
ScheduleServiceImpl (class)	• Implements ScheduleService. • Contains template method implementation. • Need to provide implementation for schedule related functionalities. • Do not modify, add or delete any method signature	To be implemented.

6.5 PACKAGE: COM.APPOINTMENT.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
ScheduleController (Class)	• Controller class to expose all rest-endpoints for schedule related activities. • May also contain local exception handler methods	To be implemented

6.6 PACKAGE: COM.APPOINTMENT.DTO

Resources

Class/Interface	Description	Status
DoctorDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.
ScheduleDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.

6.7 PACKAGE: COM.APPOINTMENT.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 **id** as primary key. - Map this class with a **doctor table**. - Generate the **id** using the IDENTITY strategy	Partially implemented.
Sell (Class)	- This class is partially implemented. - Annotate this class with proper annotation to declare it as an entity class with **id** as primary key. - Map this class with a **schedule table**. - Generate the **id** using the IDENTITY strategy	Partially implemented.

6.8 PACKAGE: COM.APPOINTMENT.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	- Custom Exception to be thrown when trying to fetch or delete the product/sell info which does not exist. - Need to create Exception	Already implemented.

	Handler for same wherever needed (local or global)	

1 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 use command:

   mvn clean package -Dmaven.test.skip

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.