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.

You are responsible for developing a system that allows patients to easily search, book, and manage their appointments with healthcare providers. The application offers functionalities to create, update, and delete doctor profiles. Manage appointments including the booking of new appointments and updating or canceling existing ones and also should allow users to retrieve detailed appointment schedules by doctor or by date.

Following is the requirement specifications:

	Appointment Scheduler Application
Modules	
1	Doctor
2	Schedule
Doctor Module Functionalities	
1	List all doctors **(must return all doctors by name and that also in list)**
2	Get doctor by id
3	Create doctor
4	Update doctor by id
5	Delete doctor by id
6	Get doctor by speciality **(should be a custom query)**

Schedule Module Functionalities	
1	Create an appointment
2	Update an appointment by id
3	Get an appointment by id
4	Get list of all appointments for a doctor on particular day **(must return schedules for a doctor on given day and that also in list) (should be a custom query)**

Overall Application	
1	Actuator support needs to be added in the properties file. Expose all actuator endpoints except beans.
2	In application.properties file expose a property “profile.validate.data” with value as “This is default profile”. Create application-qa.properties file (for QA profile) and expose a property “profile.validate.data” with value as “This is qa profile”.
3	Create an endpoint in DoctorController with following configurations: 1. Method – GET 2. Endpoint - /profile 3. Return – String **The method for this endpoint must read the “profile.validate.data” property file and return its value based on the active profile.**

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 DOCTOR CONSTRAINTS

- When fetching a doctor by ID, if the doctor ID does not exist, the service method should throw a NotFoundException with “Doctor not found” message.
- When updating a doctor, if the doctor ID does not exist, the service method should throw a NotFoundException with “Doctor not found” message.
- When removing a doctor, if the doctor ID does not exist, the service method should throw a NotFoundException with “Doctor not found” message.

2.2 SCHEDULE CONSTRAINTS

- When deleting a schedule by ID, if the schedule ID does not exist, the service method should throw a NotFoundException with “Schedule not found” message.
- When fetching a schedule by ID, if the schedule ID does not exist, the service method should throw a NotFoundException with “Schedule not found” message.
- When updating a schedule by ID, if the schedule ID does not exist, the service method should throw a NotFoundException with “Schedule not found” message.

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:

- Id must be of type id.
- Name should not be blank.
- Hospital name should not be blank.
- Speciality should not be blank.
- DailyTime should not be null.

Schedule:

- Id must be of type id.
- Name should not be blank.
- Doctor should not be null.
- Day should not be null.
- Time should not be null.
- Timings should not be blank.

4 Rest Endpoints

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

4.1 DoctorController

URL Exposed		Purpose
1. /api/doctors		Fetches all the doctors
Http Method	GET	
Parameter	-	
Return	List<DoctorDTO>	

2. /api/doctors/{id}		
Http Method	GET	Get a doctor by id
Parameter 1	Long (id)	
Return	DoctorDTO	

3. /api/doctors		
Http Method	POST **The doctor data to be created must be received in the controller using @RequestBody.**	Create a new doctor
Parameter	-	
Return	DoctorDTO	

4. /api/doctors/{id}		
Http Method	PUT **The doctor data to be updated must be received in the controller using @RequestBody.**	Updates existing doctor by id
Parameter 1	Long (id)	
Return	DoctorDTO	

5. /api/doctors/{id}		
Http Method	DELETE	Deletes a doctor by id
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<DoctorDTO>	

7. /api/doctors/profile		
Http Method	GET	Fetches the profile
Parameter 1	-	
Return	String	

4.2 ScheduleController

URL Exposed			Purpose
1. /api/schedules/appointment	Http Method	POST **The schedule data to be created must be received in the controller using @RequestBody.**	Creates a new Schedule
	Parameter	-	
	Return	ScheduleDTO	
2. /api/schedules/appointment/{id}	Http Method	PUT **The schedule data to be updated must be received in the controller using @RequestBody.**	Updates a schedule by id
	Parameter 1	Long (id)	
	Return	ScheduleDTO	
3. /api/schedules/appointment/{id}	Http Method	GET	Fetches a schedule by id
	Parameter	Long (id)	
	Return	ScheduleDTO	
4. /api/schedules/doctor/{id}/{day}	Http Method	GET	Fetches the list of all schedules for a doctor by given id on given day
	Parameter 1	Long (id)	
	Parameter 2	String (day)	
	Return	List<ScheduleDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.APPOINTMENT

Resources

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

5.2 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 a list of doctors by their speciality. ● You can go ahead and add any custom methods as per requirements.	To be implemented.
ScheduleRepository (interface)	● Repository interface exposing CRUD functionality for Schedule entity. ● It must contain the method for: ○ Finding a list of schedules for a specific doctor on a specific day. ● You can go ahead and add any custom methods as per requirements.	To be implemented.

5.3 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.
ScheduleService (interface)	● Interface to expose method signatures for schedule related functionality. ● Do not modify, add or delete any method.	Already implemented.

5.4 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.

5.5 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

5.6 PACKAGE: COM.APPOINTMENT.DTO

Resources

Class/Interface	Description	Status
DoctorDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
ScheduleDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 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.
Schedule (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.

5.8 Package: com.appointment.exception

Resources

Class/Interface	**Description**	**Status**
NotFoundException (Class)	● Custom Exception to be thrown when trying to fetch, update or delete the doctor or schedule info which does not exist.	Already implemented.

	• Need to create Exception Handler for same wherever needed (local or global)	
ErrorResponse (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	Already implemented.
RestExceptionHandler (Class)	• RestControllerAdvice Class for defining rest exception handlers. • Contains Exception Handler for **NotFoundException** class. • Use this as a reference for creating exception handler for other custom exception classes	Already implemented.

5.9 PROPERTIES FILES

Resources

Class/Interface	**Description**	**Status**
application.properties	• This file is treated as the default properties file for this application. • You need to write properties to add actuator support.	Partially implemented.

	● You need to write property to expose all endpoints. ● You need to write property to exclude /beans endpoint. ● Add “profile.validate.data” property with value as “This is default profile”.	
application-qa.properties	● This file is treated as the qa properties file for this application. ● You need to write properties to add actuator support. ● You need to write property to expose all endpoints. ● You need to write property to exclude /beans endpoint. ● Add “profile.validate.data” property with value as “This is qa profile”.	To be implemented.

6 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. Please use 127.0.0.1 instead of localhost to test rest endpoints.**
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**

12. To login to mysql instance: Open new terminal and use following command:

    a. **sudo systemctl enable mysql**

    b. **sudo systemctl start mysql**

    NOTE: After typing any of the above commands you might encounter any warnings.

    >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. **mysql -u root -p**

    The last command will ask for password which is 'pass@word1'

13. Mandatory: Before final submission run the following command:

    mvn test

14. 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.