
System Requirements Specification

Index

For

Job Board Application

Version 1.0

TABLE OF CONTENTS

BACKEND-SPRING DATA RESTFUL APPLICATION	3
1 Project Abstract	3
2 Assumptions, Dependencies, Risks / Constraints	4
2.1 Job Constraints	
2.2 User Constraints	4
3 Business Validations	4
4 Rest Endpoints	5
4.1 JobController	
4.2 UserController	5
5 Template Code Structure	6
5.1 Package: com.jobboard	6
5.2 Package: com.jobboard.repository	6
5.3 Package: com.jobboard.service	6
5.4 Package: com.jobboard.service.impl	7
5.5 Package: com.jobboard.controller	7
5.6 Package: com.jobboard.dto	8
5.7 Package: com.jobboard.entity	8
5.8 Package: com.jobboard.exception	9
6 Execution Steps to Follow for Backend	10

JOB BOARD APPLICATION

System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Job Board Application** is implemented using Spring Data with a MySQL database. The application aims to provide a comprehensive platform for managing and organizing all jobs for job seekers.

Following is the requirement specifications:

	Job Board Application
Modules	
1	Job
2	User
Job Module Functionalities	
1	List all jobs (must return all jobs by title in ascending order and that also in pages)
2	Get job by id
3	Create job
4	Update job by id
5	Delete job by id
6	List jobs by location (must use dynamic method)
User Module Functionalities	
1	List all users
2	Get user by id
3	Create user
4	Update user by id (must be transactional)
5	Delete user by id
6	Apply for a job (must be transactional)
7	Get applied jobs by user id (must use custom query)

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 JOB CONSTRAINTS

- When fetching a job by ID, if the job ID does not exist, the service method should throw a `NotFoundException` with "Job not found." message.
- When updating a job, if the job ID does not exist, the service method should throw a `NotFoundException` with "Job not found." message.

2.2 USER CONSTRAINTS

- When fetching a user by ID, if the user ID does not exist, the service method should throw a `NotFoundException` with "User not found." message.
- When updating a user, if the user ID does not exist, the service method should throw a `NotFoundException` with "User 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

Job

- Title should not be blank.
- Salary bracket should not be blank.
- Location should not be blank.
- Required skills should not be blank.

User

- Name should not be blank.
- Phone number should not be blank.
- Email should not be blank and must be of email type.

4 REST ENDPOINTS

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

4.1 JOBCONTROLLER

URL Exposed		Purpose
1. /api/jobs		Fetches all the jobs
Http Method	GET	
Parameter	-	
Return	Page<JobDTO>	
2. /api/jobs/{id}		Get a job by id
Http Method	GET	
Parameter 1	Long (id)	
Return	JobDTO	
3. /api/jobs		Create a new job
Http Method	POST	
	The job data to be created must be received in the controller using @RequestBody.	
Parameter	-	
Return	JobDTO	
4. /api/jobs/{id}		Updates existing job by id
Http Method	PUT	
	The job data to be updated must be received in the controller using @RequestBody.	
Parameter 1	Long (id)	
Return	JobDTO	
5. /api/jobs/{id}		Deletes a job by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

5. /api/jobs/byLocation		Fetches a list of all jobs in given location
Http Method	GET	
Request Parameter 1	location	
Return	List<JobDTO>	

4.2 USERCONTROLLER

URL Exposed		Purpose
1. /api/users		Fetches all the users
Http Method	GET	
Parameter	-	
Return	List<UserDTO>	
2. /api/users/{id}		Get a user by id
Http Method	GET	
Parameter 1	Long (id)	
Return	UserDTO	
3. /api/users		Create a new user
Http Method	POST	
	The user data to be created must be received in the controller using @RequestBody.	
Parameter	-	
Return	UserDTO	
4. /api/users/{id}		Updates existing user by id
Http Method	PUT	
	The user data to be updated must be received in the controller using @RequestBody.	
Parameter 1	Long (id)	
Return	UserDTO	
5. /api/users/{id}		Deletes a user by id
Http Method	DELETE	
Parameter 1	Long (id)	

Return	-	
--------	---	--

6. /api/users/{userId}/apply/{jobId}		Applies a job by id by user
Http Method	POST	
Parameter 1	Long (userId)	
Parameter 2	Long (jobId)	
Return	UserDTO	

7. /api/users/{userId}/appliedJobs		Fetches all applied jobs by user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<JobDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.JOBBOARD

Resources

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

5.2 PACKAGE: COM.JOBBOARD.REPOSITORY

Resources

Class/Interface	Description	Status
JobRepository (interface)	<ul style="list-style-type: none"> Repository interface exposing CRUD functionality for Job Entity. It must contain the methods for: <ul style="list-style-type: none"> finding all jobs sorted by tile in ascending order. finding all jobs by location sorted by title in ascending order. You can go ahead and add any custom methods as per requirements. 	Partially implemented.

UserRepository (interface)	<ul style="list-style-type: none"> Repository interface exposing CRUD functionality for User Entity. It must contain the methods for: <ul style="list-style-type: none"> finding all applied jobs by user id. method to apply a job. You can go ahead and add any custom methods as per requirements. 	Partially implemented.
-----------------------------------	---	------------------------

5.3 PACKAGE: COM.JOBBOARD.SERVICE

Resources

Class/Interface	Description	Status
JobService (interface)	<ul style="list-style-type: none"> Interface to expose method signatures for job related functionality. Do not modify, add or delete any method. 	Already implemented.
UserService (interface)	<ul style="list-style-type: none"> Interface to expose method signatures for user related functionality. Do not modify, add or delete any method. 	Already implemented.

5.4 PACKAGE: COM.JOBBOARD.SERVICE.IMPL

Class/Interface	Description	Status
JobServiceImpl (class)	<ul style="list-style-type: none"> Implements JobService. Contains template method implementation. Need to provide implementation for job related functionalities. 	To be implemented.

	<ul style="list-style-type: none"> Do not modify, add or delete any method signature 	
UserServiceImpl (class)	<ul style="list-style-type: none"> Implements UserService. Contains template method implementation. Need to provide implementation for user related functionalities. Do not modify, add or delete any method signature 	To be implemented.

5.5 PACKAGE: COM.JOBBOARD.CONTROLLER

Resources

Class/Interface	Description	Status
JobController (Class)	<ul style="list-style-type: none"> Controller class to expose all rest-endpoints for job related activities. May also contain local exception handler methods 	To be implemented
UserController (Class)	<ul style="list-style-type: none"> Controller class to expose all rest-endpoints for user related activities. May also contain local exception handler methods 	To be implemented

5.6 PACKAGE: COM.JOBBOARD.DTO

Resources

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

5.7 PACKAGE: COM.JOBBOARD.ENTITY

Resources

Class/Interface	Description	Status
Job (Class)	<ul style="list-style-type: none">• 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 job table.• Generate the id using the IDENTITY strategy	Partially implemented.
User (Class)	<ul style="list-style-type: none">• 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 user table.• Generate the id using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM.JOBBOARD.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	<ul style="list-style-type: none">• Custom Exception to be thrown when trying to fetch or delete the Job/User info which does not exist.• Need to create Exception Handler for same wherever needed (local or global)	Already 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.
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.