System Requirements Specification

Index

For

TPO Hub Application

Version 1.0

TABLE OF CONTENTS

BACKEND-SPRING BOOT RESTFUL APPLICATION 3
1. Project Abstract 3
2. Assumptions, Dependencies, Risks / Constraints 5
2.1. Company Constraints 5
2.2. Common Constraints 5
3. Business Validations 6
3.1. Business Validations - Company 6
4. Rest Endpoints 7
4.1. CompanyController 7
5. Template Code Structure 9
5.1. Package: com.tpohubapplication 9
5.2. Package: com.tpohubapplication.repository 9
5.3. Package: com.tpohubapplication.service 10
5.4. Package: com.tpohubapplication.service.impl 11
5.5. Package: com.tpohubapplication.controller 11
5.6. Package: com.tpohubapplication.dto 12
5.7. Package: com.tpohubapplication.entity 12
5.8. Package: com.tpohubapplication.exception 13
5.9. Properties Files 13
6. Execution Steps to Follow for Backend 14

TPO HUB APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **TPO Hub Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for managing and registering different types of volunteers for different types of programs.

Following is the requirement specifications:

	TPO Hub Application
Modules	
1	Company
Company Module Functionalities	
1	Create a company
2	Get a company by id
3	Update a company by id
4	Delete a company by id
5	Get all companies
6	Get all companies by stream **(should be a custom query)**
7	Get all completed by minimum passing percentage **(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 CompanyController 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 Company Constraints

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

2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation checks must be done and must throw custom exceptions 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 the 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 Business Validations - Company

- Company name should not be blank.
- Stream should not be blank.
- MinimumQualification should not be blank.
- MustToHave should not be blank.
- GoodToHave 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 COMPANY CONTROLLER

URL Exposed		Purpose
1. /api/companies		
Http Method	POST	Creates a new company
Parameter	**The company data to be created must be received in the controller using @RequestBody.**	
Return	CompanyDTO	
2. /api/companies/{id}		
Http Method	GET	Gets a company by id
Parameter 1	Long (id)	
Return	CompanyDTO	
3. /api/companies/{id}		
Http Method	PUT	Updates a company by id
Parameter	Long (id) **The company data to be updated must be received in the controller using @RequestBody.**	
Return	CompanyDTO	
4. /api/companies/{id}		
Http Method	DELETE	Deletes a company by id
Parameter 1	Long (id)	
Return	-	
5. /api/companies		
Http Method	GET	Fetches all companies
Parameter 1	-	
Return	List<CompanyDTO>	
6. /api/companies/stream/{stream}		
Http Method	GET	Searches all companies by stream

Parameter 1	String (stream)	
Return	List<CompanyDTO>	

7. /api/companies/minPassingPercentage/{percentage}		Fetches list of companies having passed minimum passing percentage
Http Method	GET	
Parameter 1	String (percentage)	
Return	List<CompanyDTO>	

8. /api/companies/profile		Fetches the profile
Http Method	GET	
Parameter 1	-	
Return	String	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.TPOHUBAPPLICATION

Resources

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

5.2 PACKAGE: COM.TPOHUBAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status
CompanyRepository (interface)	• Repository interface exposing CRUD functionality for Company Entity. • You can go ahead and add any custom methods as per requirements. • You need to write a function to find all companies by stream. • You need to write a function to	Partially implemented.

	find all companies having a minimum passing percentage as shared.	

5.3 PACKAGE: COM.TPOHUBAPPLICATION.SERVICE

Resources

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

5.4 PACKAGE: COM.TPOHUBAPPLICATION.SERVICE.IMPL

Class/Interface	Description	Status
CompanyServiceImpl (class)	• Implements CompanyService. • Contains template method implementation. • Need to provide implementation for company related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.TPOHUBAPPLICATION.CONTROLLER

Resources

Class/Interface	Description	Status
CompanyController (Class)	• Controller class to expose all rest-endpoints for company related activities. • May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.TPOHUBAPPLICATION.DTO

Resources

Class/Interface	Description	Status
CompanyDTO (Class)	• Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.

5.7 PACKAGE: COM.TPOHUBAPPLICATION.ENTITY

Resources

Class/Interface	Description	Status
Company (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 company table. • Generate the id using the IDENTITY strategy	Partially implemented.

5.8 PACKAGE: COM.TPOHUBAPPLICATION.EXCEPTION

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the company info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	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. • 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".	Partially implemented.
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".	Partially 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:**
    i. **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:**
    i. **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**

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 the second sql command (sudo systemctl start mysql), you may encounter a warning message like :

    System has not been booted with systemd as init system (PID 1). Can't operate.
    Failed to connect to bus: Host is down

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

    c. **mysql -u root -p**
        i. **The last command will ask for password which is 'pass@word1'**
13. **Mandatory: Before final submission run the following command:**
        i. **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.**