System Requirements Specification

Index

For

ToDo List Application

Version 1.0

TABLE OF CONTENTS

TODO LIST APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **ToDo List Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive view/summary of user's todo list with adding details like description, setting up priority with their status.

Following is the requirement specifications:

	ToDo List Application
Modules	
1	ToDo
ToDo Module Functionalities	
1	Create a todo
2	Get a todo by id
3	Update a todo by id
4	Delete a todo by id
5	Get all todos
6	Search a todo by name
7	Get all completed todos **(should be a custom query)**
8	Get all todos which have higher priority **(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 ToDoController 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 Todo Constraints

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

- Title should not be null.
- Description should not be 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 TODO CONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/todos
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter</td><td>The todo data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Return</td><td>ToDoDTO</td></tr></table></td><td>Creates a new todo</td></tr>
<tr><td>2. /api/todos/{id}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>ToDoDTO</td></tr></table></td><td>Gets a todo by id</td></tr>
<tr><td>3. /api/todos/{id}
<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter</td><td>Long (id)

The todo data to be updated must be received in the controller using @RequestBody.</td></tr><tr><td>Return</td><td>ToDoDTO</td></tr></table></td><td>Updates a todo by id</td></tr>
<tr><td>4. /api/todos/{id}
<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Deletes a todo by id</td></tr>
<tr><td>5. /api/todos
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>Page<ToDoDTO></td></tr></table></td><td>Fetches all todos</td></tr>
<tr><td>6. /api/todos/search
<table><tr><td>Http Method</td><td>GET</td></tr></table></td><td>Searches all todos by name</td></tr>
</table>

Request Param	String (name)	
Return	List<ToDoDTO>	

7. /api/todos/completed		
Http Method	GET	Fetches list of completed todos
Parameter 1	-	
Return	List<ToDoDTO>	

8. /api/todos/priorityGreaterThan/{priority}		
Http Method	GET	Fetches all todos having greater than passed priority value
Parameter 1	int (priority)	
Return	List<ToDoDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.TODOAPPLICATION

Resources

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

5.2 PACKAGE: COM.TODOAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status
ToDoRepository (interface)	• Repository interface exposing CRUD functionality for ToDo Entity. • You can go ahead and add any custom methods as per requirements. • You need to write a function to find all todos by title. • You need to write a function to find all completed todos. • You need to write a custom query	Partially implemented.

	to find all todos having higher priority than passed one.	

5.3 Package: com.todoapplication.service

Resources

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

5.4 Package: com.todoapplication.service.impl

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

5.5 Package: com.todoapplication.controller

Resources

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

5.6 PACKAGE: COM.TODOAPPLICATION.DTO

Resources

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

5.7 PACKAGE: COM.TODOAPPLICATION.ENTITY

Resources

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

5.8 PACKAGE: COM.TODOAPPLICATION.EXCEPTION

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the todo 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**
    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.**