System Requirements Specification

Index

For

Daily Joggers Application

Version 1.0

TABLE OF CONTENTS

DAILY JOGGERS APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Daily Joggers Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive view/summary of user's calories burnt in different types of activities across different days.

Following is the requirement specifications:

	Daily Joggers Application
Modules	
1	DailyActivity
2	User
DailyActivity Module Functionalities	
1	Create an activity for a user
2	Get activities for a user
3	Update an activity for a user
4	Delete an activity for a user
5	Get summarized daily activity for a user on particular date (**should be a custom query)**.
6	Get weekly summarized activity for a user (**should be a custom query).**

User Module Functionalities	
1	Create a user
2	Get all users
3	Get user by id
4	Update user by id
5	Delete user by id

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

2 Assumptions, Dependencies, Risks / Constraints

2.1 DAILY ACTIVITY CONSTRAINTS

- When fetching a daily activity for user by ID, if the user ID does not exist, the service method should throw ResourceNotFoundException with “No daily activity found.” message.
- When deleting a daily activity for user by ID, if the user ID does not exist, the service method should throw ResourceNotFoundException with “No daily activity found.” message.

2.2 USER CONSTRAINTS

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

- UserId must not be null.
- Date must not be null.
- Steps must not be null.
- Distance must not be null.

Business Validations - User

4 Username must not be blank.

5 Email must not be blank and of type email.

6 REST ENDPOINTS

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

6.1 DAILYACTIVITY CONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/daily-activities/{userId}
The data must be received in controller using @RequestBody.
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter</td><td>userId</td></tr><tr><td>Return</td><td>DailyActivity</td></tr></table></td><td>Creates a new daily activity for user</td></tr>
<tr><td>2. /api/ daily-activities/{userId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>userId</td></tr><tr><td>Return</td><td>DailyActivity</td></tr></table></td><td>Gets daily activity for a user</td></tr>
<tr><td>2. /api/ daily-activities/{userId}/{activityId}
The data must be received in controller using @RequestBody.
<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter</td><td>userId, activityId</td></tr><tr><td>Return</td><td>DailyActivity</td></tr></table></td><td>Updates a daily activity of user</td></tr>
<tr><td>4. /api/ daily-activities/{userId}/{activityId}
<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>userId, activityId</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Deletes an activity of a user</td></tr>
</table>

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>5. /api/ daily-activities/{userId}/summary/{date}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter</td><td>userId, date</td></tr><tr><td>Return</td><td>List<DailyActivity></td></tr></table></td><td>Fetches all daily activities for a user on particular</td></tr>
<tr><td>6. /api/ daily-activities/{userId}/summary/weekly
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>userId</td></tr><tr><td>Return</td><td>List<DailyActivity></td></tr></table></td><td>Fetches list of daily activities summarized weekly basis</td></tr>
<tr><td>7. /api/daily-activities/{userId}/summary/monthly</td><td></td></tr>
</table>

Http Method	GET	Fetches list of daily activities summarized monthly basis
Parameter	userId	
Return	List<DailyActivity>	

1.1 USER CONTROLLER

URL Exposed		Purpose
1. /api/users **The data must be received in controller using @RequestBody.**		Creates a new User
Http Method	POST	
Parameter	-	
Return	User	
2. /api/users		Fetches all users in pages
Http Method	GET	
Parameter 1	-	
Return	List<User>	
3. /api/users/{userId}		Fetches a user by id
Http Method	GET	
Parameter	userId	
Return	User	
4. /api/ users/{userId} **The data must be received in controller using @RequestBody.**		Updates a user by id
Http Method	PUT	
Parameter	userId	
Return	User	

5. /api/users/{userId}		Deletes a user by id
Http Method	DELETE	
Parameter	userId	
Return	-	

2 TEMPLATE CODE STRUCTURE

2.1 PACKAGE: COM.LAPTOPSTORE

Resources

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

2.2 PACKAGE: COM.DAILYJOGGERS.REPOSITORY

Resources

Class/Interface	Description	Status
DailyAcitivityRepository (interface)	● Repository interface exposing CRUD functionality for Daily Activity Entity. ● You can go ahead and add any custom methods as per requirements. ● You need to write dynamic query function to find all daily activities for user by id in pages. ● You need to write dynamic query function to find daily activity by user id and date. ● You need to write custom query to find daily activity summarized for a user on a particular date. ● You need to write native query to find daily activity summarized for a user on weekly basis. ● You need to write custom query to find daily activity summarized for a user on monthly basis.	Partially implemented.

UserRepository (interface)	• Repository interface exposing CRUD functionality for User Entity. • You can go ahead and add any custom methods as per requirements. • You need to write dynamic query to find all users by user name in ascending order in pages.	Partially implemented.

2.3 PACKAGE: COM. DAILYJOGGERS.SERVICE

Resources

Class/Interface	Description	Status
DailyActivityService (interface)	• Interface to expose method signatures for daily activity related functionality. • Do not modify, add or delete any method.	Already implemented.
UserService (interface)	• Interface to expose method signatures for user related functionality. • Do not modify, add or delete any method.	Already implemented.

2.4 PACKAGE: COM. DAILYJOGGERS.SERVICE.IMPL

Class/Interface	Description	Status
DailyAcitivityServiceImpl (class)	● Implements DailyActivityService. ● Contains template method implementation. ● Need to provide implementation for daily activity related functionalities. ● Do not modify, add or delete any method signature	To be implemented.
UserServiceImpl (class)	● Implements UserService. ● Contains template method implementation. ● Need to provide implementation for sell related functionalities. ● Do not modify, add or delete any method signature	To be implemented.

2.5 PACKAGE: COM. DAILYJOGGERS.CONTROLLER

Resources

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

UserController (Class)	• Controller class to expose all rest-endpoints for user related activities. • May also contain local exception handler methods	To be implemented

2.6 PACKAGE: COM. DAILYJOGGERS.DTO

Resources

Class/Interface	Description	Status
DailyActivityDTO (Class)	Use appropriate annotations from the Validation API for validating attributes of this class.	Partially implemented.
UserDTO (Class)	Use appropriate annotations from the Validation API for validating attributes of this class.	Partially implemented.

2.7 PACKAGE: COM. DAILYJOGGERS.ENTITY

Resources

Class/Interface	Description	Status

DailyActivity (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 **daily activity table**. ● Generate the **id** using the IDENTITY strategy	Partially implemented.
User (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 **user table**. ● Generate the **id** using the IDENTITY strategy	Partially implemented.

2.8 PACKAGE: COM. DAILYJOGGERS.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	● Custom Exception to be thrown when trying to fetch or delete the user/daily activity info which does not exist. ● Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

2.9 Properties files

Resources

File	Description	Status
application.properties	● This file is treated as default properties file for this application. ● You need to write properties to add actuator support. ● You need to write property to expose all end points. ● You need to write property to exclude /beans endpoint.	Partially implemented.
Application-qa.properties	● This file will be treated as qa profile properties file. ● You need to write properties for H2 database and other empty kept properties. ● You need to write properties to add actuator support. ● You need to write property to expose all end points. ● You need to write property to exclude /beans endpoint.	Partially implemented.

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.