System Requirements Specification

Index

For

Tour Guide Application

Version 1.0

TABLE OF CONTENTS

TOUR GUIDE APPLICATION

System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Tour Guide Application** is implemented using Spring Data with a MySQL database. The application aims to provide a comprehensive platform for finding and exploring all packages across different regions.

Following is the requirement specifications:

	Tour Guide Application
Modules	
1	TourGuide
2	User
TourGuide Module Functionalities	
1	List all packages **(must return all packages by placeName in ascending order and that also in pages)**
2	Get packages by id
3	Create package
4	Update package by id
5	Delete package by id
6	List all packages by number of days **(must use dynamic method)**
7	List all tour packages booked by user **(must use custom query)**

User Module Functionalities	
1	Get all users
2	Get user by id
3	Create user
4	Update user by id **(must be transactional)**
5	Delete user by id

2 Assumptions, Dependencies, Risks / Constraints

2.1 TOURPACKAGE CONSTRAINTS

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

2.2 USER CONSTRAINTS

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

TourPackage

- Place name should not be blank.
- Number of days should be a positive value.
- Price should be a positive value.
- Places to visit should not be blank.

User

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

4 REST ENDPOINTS

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

4.1 TOURPACKAGECONTROLLER

URL Exposed		Purpose
1. /api/packages Http Method: GET Parameter: - Return: Page<TourPackageDTO>		Fetches all the tour packages
2. /api/packages/by-days Http Method: GET Request Parameter 1: int (numberOfDays) Return: List<TourPackageDTO>		Get all tour packages by number of days
3. /api/packages/by-user/{userId} Http Method: GET Parameter: Long (userId) Return: List<TourPackageDTO>		Get all tour packages by user id
4. /api/packages/{id} Http Method: GET Parameter 1: Long (id) Return: TourPackageDTO		Get existing tour package by id
5. /api/packages Http Method: POST **The tour package data to be created must be received in the controller using @RequestBody.** Parameter 1: - Return: TourPackageDTO		Create a new tour package

6. /api/packages/{id}	
Http Method	PUT **The tour package data to be updated must be received in the controller using @RequestBody.**
Parameter	Long (id)
Return	TourPackageDTO
Updates an existing tour package by id	

7. /api/packages/{id}	
Http Method	DELETE
Parameter 1	Long {id}
Return	TourPackageDTO
Delete a new tour package by id	

4.2 UserController

URL Exposed	Purpose
1. /api/users Http Method: GET Parameter: - Return: List<UserDTO>	Fetches list of all users
2. /api/users/{id} Http Method: GET Parameter 1: Long (id) Return: UserDTO	Get user by id
3. /api/users Http Method: POST Parameter: - Return: UserDTO	Creates a new user
4. /api/users/{id} Http Method: PUT Parameter: Long (id) Return: UserDTO	Updates an user by id
5. /api/users/{id} Http Method: DELETE Parameter: Long (id) Return: -	Deletes an user by id

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.TOURAPPLICATION

Resources

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

5.2 PACKAGE: COM.TOURAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status
TourPackageRepository (interface)	● Repository interface exposing CRUD functionality for Tour Package Entity. ● You can go ahead and add any custom methods as per requirements. ● It must contain the methods for: ○ finding all tours by number of days. ○ finding all tour packages by user id.	Partially implemented.
UserRepository (interface)	● Repository interface exposing CRUD functionality for User Entity. ● You can go ahead and add any custom methods as per requirements.	Partially implemented.

5.3 PACKAGE: COM.TOURAPPLICATION.SERVICE

Resources

Class/Interface	Description	Status
TourPackageService (interface)	• Interface to expose method signatures for tour package 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.

5.4 PACKAGE: COM.TOURAPPLICATION.SERVICE.IMPL

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

5.5 PACKAGE: COM.TOURAPPLICATION.CONTROLLER

Resources

Class/Interface	Description	Status
TourPackageController (Class)	• Controller class to expose all rest-endpoints for tour package 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

5.6 PACKAGE: COM.TOURAPPLICATION.DTO

Resources

Class/Interface	Description	Status
TourPackageDTO (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.TOURAPPLICATION.ENTITY

Resources

Class/Interface	Description	Status
TourPackage (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 **tour package 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.

5.8 PACKAGE: COM.TOURAPPLICATION.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the entity info which does not exist. • Need to create Exception Handler for same wherever	Already implemented.

	needed (local or global)	

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**

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.**