System Requirements Specification

Index

For

Trip Tuner Application

Version 1.0

TABLE OF CONTENTS

TRIP TUNER APPLICATION

System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Trip Tuner Application** is implemented using Spring Data with a MySQL database, designed to optimize travel planning. This app serves as an extensive travel planning tool that assists users in curating and managing customized itineraries and destinations dynamically.

You are tasked with developing a system that allows users to smoothly create, update, and manage their travel itineraries and destinations dynamically and transactionally. The application will offer functionalities such as adding new destinations to itineraries, updating details, and removing destinations or itineraries as needed. Additionally, users will be able to search for specific itineraries or destinations and filter them by various criteria, such as date ranges or destination types, ensuring a tailored and efficient planning experience that supports transactional operations for critical data manipulations.

Following is the requirement specifications:

	Trip Tuner Application
Modules	
1	Itinerary
2	Destination
Itinerary Module Functionalities	
1	List all itineraries **(must return all itineraries by name in ascending order and that also in pages)**
2	Get itinerary by id
3	Create itinerary
4	Update itinerary by id
5	Delete itinerary by id
6	Search itineraries by name **(must use dynamic method)**
7	Filter itineraries by date range **(must use custom query to return list of itineraries within the specified date range)**

Destination Module Functionalities	
1	Add destination to itinerary **(must be transactional)**
2	Get all destinations for itinerary **(must use dynamic method)**
3	Get destination by id within itinerary **(must use dynamic method)**
4	Update destination by id **(must be transactional)**
5	Delete destination from itinerary **(must be transactional)**
6	Search destinations within itinerary by name **(must use dynamic method)**

7	Filter destinations within itinerary by type of destination **(must use dynamic method)**

2 Assumptions, Dependencies, Risks / Constraints

2.1 Itinerary Constraints

- When fetching an itinerary by ID, if the itinerary ID does not exist, the service method should throw a ResourceNotFoundException with the message "Itinerary not found with id: [itineraryID]".
- When updating an itinerary, if the itinerary ID does not exist, the service method should throw a ResourceNotFoundException with the message "Itinerary not found with id: [itineraryID]".
- When removing an itinerary, if the itinerary ID does not exist, the service method should throw a ResourceNotFoundException with the message "Itinerary not found with id: [itineraryID]".

2.2 Destination Constraints

- When adding a destination to an itinerary, If the itinerary ID does not exist, the service method should throw a ResourceNotFoundException with the message "Itinerary not found with id: [itineraryId]".
- When fetching a destination by ID within an itinerary, If the destination ID or itinerary ID does not exist or does not match, the service method should throw a ResourceNotFoundException with the message "Destination not found with id: [destinationId] for itinerary id: [itineraryId]".
- When updating a destination within an itinerary, If either the destination ID or itinerary ID does not exist or does not match, the service method should throw a ResourceNotFoundException with the message "Destination not found with id: [destinationId] for itinerary id: [itineraryId]".
- When deleting a destination from an itinerary, If either the destination ID or itinerary ID does not exist or does not match, the service method should throw a ResourceNotFoundException with the message "Destination not found with id: [destinationId] for itinerary id: [itineraryId]".

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

Itinerary:

- Id must be of type id.
- Name should not be blank, min 1 and max 255 characters.
- Start Date should not be null, must be either today's date or a future date.
- End Date should not be null, must be either today's date or a future date.

Destination:

- Id must be of type id.
- Name should not be blank, min 1 and max 255 characters.
- Type should not be blank, min 1 and max 255 characters.
- itineraryId 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 ItineraryController

URL Exposed		Purpose
1. /api/itineraries Http Method Parameter Return	 GET - Page<ItineraryDTO>	Retrieves all itineraries with pagination
2. /api/itineraries/{itineraryId} Http Method Parameter 1 Return	 GET Long (itineraryId) ItineraryDTO	Get itinerary by id
3. /api/itineraries Http Method Parameter Return	 POST **The itinerary data to be created must be received in the controller using @RequestBody.** - ItineraryDTO	Creates a new travel itinerary

4. /api/itineraries/{itineraryId}	
Http Method	PUT **The itinerary data to be updated must be received in the controller using @RequestBody.**
Parameter 1	Long (itineraryId)
Return	ItineraryDTO
Updates existing itinerary by its id	

5. /api/itineraries/{itineraryId}	
Http Method	DELETE
Parameter 1	Long (itineraryId)
Return	-
Deletes an itinerary by id	

6. /api/itineraries/search	
Http Method	GET
Parameter 1	String (name)
Return	List<ItineraryDTO>
Searches for itineraries by name	

7. /api/itineraries/filter	
Http Method	GET
Parameter 1	Date (startDate)
Parameter 2	Date (endDate)
Return	List<ItineraryDTO>
Filters itineraries based on a specified start and end date	

4.2 DESTINATIONCONTROLLER

URL Exposed		Purpose
1. /api/itineraries/{itineraryId}/destinations		
Http Method	POST **The destination data to be created must be received in the controller using @RequestBody.**	
Parameter	Long (itineraryId)	
Return	DestinationDTO	
		Adds a new destination to a specific itinerary

2. /api/itineraries/{itineraryId}/destinations/{destinationId}		Updates an existing destination within an itinerary
Http Method	PUT **The destination data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (itineraryId)	
Parameter 2	Long (destinationId)	
Return	DestinationDTO	
3. /api/itineraries/{itineraryId}/destinations/{destinationId}		Removes a destination from an itinerary by its ID
Http Method	DELETE	
Parameter 1	Long (itineraryId)	
Parameter 2	Long (destinationId)	
Return	-	
4. /api/itineraries/{itineraryId}/destinations		Retrieves all destinations associated with a specific itinerary
Http Method	GET	
Parameter 1	Long (itineraryId)	
Return	List<DestinationDTO>	
5. /api/itineraries/{itineraryId}/destinations/{destinationId}		Retrieves details of a specific destination by its ID within a given itinerary
Http Method	GET	
Parameter 1	Long (itineraryId)	
Parameter 2	Long (destinationId)	
Return	DestinationDTO	
6. /api/itineraries/{itineraryId}/destinations/search		Searches for destinations within an itinerary based on the destination name
Http Method	GET	
Parameter 1	Long (itineraryId)	
Parameter 2	String (name)	
Return	List<DestinationDTO>	
7. /api/itineraries/{itineraryId}/destinations/filter		Filters destinations within an itinerary based on the type of destination
Http Method	GET	
Parameter 1	Long (itineraryId)	
Parameter 2	String (type)	
Return	List<DestinationDTO>	

5 Template Code Structure

5.1 Package: com.triptuner

Resources

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

5.2 Package: com.triptuner.repository

Resources

Class/Interface	Description	Status

ItineraryRepository (interface)	• Repository interface exposing CRUD functionality for Itinerary Entity.
	• It must contain the methods for:
	  ○ Search for itineraries by name using a case-insensitive search.
	  ○ List all itineraries ordered by name in ascending order and that also in pages.
	  ○ Retrieves the itineraries that start on or after the specified start date and end on or before the specified end date.
	• You can go ahead and add any custom methods as per requirements.
		To be implemented.

DestinationRepository (interface)	• Repository interface exposing CRUD functionality for Destination Entity.
	• It must contain the methods for:
	  ○ Finding destinations by the itinerary ID and it must return data in the list.
	  ○ Searching destinations within an itinerary by name and it must return data in the list.
	  ○ Filtering a list of destinations within an itinerary by type.
	  ○ Retrieving a destination by its ID and associated itinerary ID.
	• You can go ahead and add any custom methods as per requirements.
		To be implemented.

5.3 PACKAGE: COM.TRIPTUNER.SERVICE

Resources

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

5.4 PACKAGE: COM.TRIPTUNER.SERVICE.IMPL

Class/Interface	Description	Status
ItineraryServiceImpl (class)	• Implements ItineraryService. • Contains template method implementation. • Need to provide implementation for itinerary related functionalities. • Do not modify, add or delete any method signature.	To be implemented.
DestinationServiceImpl (class)	• Implements DestinationService. • Contains template method implementation. • Need to provide implementation for destination related functionalities. • Do not modify, add or delete any method signature.	To be implemented.

5.5 PACKAGE: COM.TRIPTUNER.CONTROLLER

Resources

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

5.6 PACKAGE: COM.TRIPTUNER.DTO

Resources

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

5.7 PACKAGE: COM.TRIPTUNER.ENTITY

Resources

Class/Interface	Description	Status
Itinerary (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 an **itinerary table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
Destination (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 **destination table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.

5.8 PACKAGE: COM.TRIPTUNER.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the itinerary or destination info which does not exist. • Need to create Exception Handler for the same wherever needed (local or global).	Already implemented.
ErrorResponse (Class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class. • Use this as a reference for creating exception handler for other custom exception classes.	Already implemented.
RestExceptionHandler (Class)	• RestControllerAdvice Class for defining rest exception handlers. • Contains Exception Handler for **ResourceNotFoundException** class. • Use this as a reference for creating exception handler for other custom exception classes.	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. 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**

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 any of the above commands you might encounter any warnings.

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

    c. mysql -u root -p

    The last command will ask for password which is 'pass@word1'

13. Mandatory: Before final submission run the following command:

    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.