System Requirements Specification

Index

For

Trip Trove Application

Version 1.0

TABLE OF CONTENTS

TRIP TROVE APPLICATION

System Requirements Specification

BACKEND-SPRING DATA RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Trip Trove Application** is implemented using Spring Data with a MySQL database, designed to enhance the travel planning experience through dynamic interaction and secure data management. This app enables seamless navigation through a variety of tour options, booking services, and user engagements, all structured to support intricate data interactions and real-time updates.

You are tasked with developing a platform that allows users to easily register, customize their profiles, and engage with various tour options. The application will facilitate the creation, updating, deletion, and detailed management of tours, bookings, and reviews. Users should be able to book tours, manage their itineraries, and access their past and upcoming bookings dynamically. Additionally, the platform will support review functionalities, allowing users to share and manage feedback on tours. Ensure all functionalities support transactional processes to maintain data integrity and provide a seamless user experience in managing their travel adventures.

Following is the requirement specifications:

	Trip Trove Application
Modules	
1	User
2	Tour
3	Booking
4	Review
User Module Functionalities	
1	Get user profile
2	Register a user
3	Update user profile

Tour Module Functionalities	
1	Create tour
2	Get all tours
3	Get tour by id
4	Update tour
5	Delete tour
6	Search tour
7	Filter tour

Booking Module Functionalities	
1	Book a tour for a user
2	Get user bookings
3	Get booking by id
4	Cancel a booking
5	Get upcoming bookings
6	Get past bookings

Review Module Functionalities	
1	Add a review to a tour
2	Get reviews by tour id
3	Update review
4	Delete review
5	Get reviews by user id

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

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

2.2 Tour Constraints

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

2.3 Booking Constraints

- When booking a tour:
    1) If the tour ID does not exist, the service method should throw a NotFoundException with the message "Tour not found".

2) If the user ID does not exist, the service method should throw a NotFoundException with the message "User not found".

- When retrieving a booking by ID, if the booking ID does not exist, the service method should throw a NotFoundException with "Booking not found" message.
- When canceling a booking, if the booking ID does not exist, the service method should throw a NotFoundException with "Booking not found" message.

2.4 Review Constraints

- When adding a review:
  1) If the tour ID does not exist, the service method should throw a NotFoundException with the message "Tour not found".
  2) If the user ID does not exist, the service method should throw a NotFoundException with the message "User not found".
- When updating a review, if the review ID does not exist, the service method should throw a NotFoundException with the message "Review not found".
- When deleting a review, if the review ID does not exist, the service method should throw a NotFoundException with the message "Review not found".

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check 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 User

- Id must be of type id.
- Username should not be blank, min 3 and max 50 characters and unique in the system.
- Password should not be blank and must be at least 6 characters long.
- Email should not be blank and must be of type email.
- Fullname should not be blank.

3.2 Tour

- Id must be of type id.
- Title should not be blank.

- Description should not be blank.
- Location should not be blank.
- Price should not be null and must be positive.
- Rating should not be null and must be positive.

3.3 BOOKING

- Id must be of type id.
- User id should not be null.
- Tour id should not be null.
- Booking date should not be null.
- Tour date should not be null.

3.4 REVIEW

- Id must be of type id.
- User id should not be null.
- Tour id should not be null.
- Comment must not be blank and must be less than 500 characters.
- Rating must not be null and must be positive.

4 REST ENDPOINTS

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

4.1 USERCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/users/profile
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (userId)</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Retrieves the profile details of a specific user</td></tr>
<tr><td>2. /api/users/register
<table><tr><td>Http Method</td><td>POST

The user data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Parameter</td><td>-</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Register a new user</td></tr>
<tr><td>3. /api/users/profile</td><td></td></tr>
</table>

Http Method	PUT **The user data to be updated must be received in the controller using @RequestBody.**	Updates the profile details of a specific user
Parameter 1	Long (userId)	
Return	UserDTO	

4.2 TourController

URL Exposed		Purpose
1. /api/tours		
Http Method	POST **The tour data to be created must be received in the controller using @RequestBody.**	Creates a new tour
Parameter 1	-	
Return	TourDTO	
2. /api/tours		
Http Method	GET	Fetches a list of all tours
Parameter 1	-	
Return	List<TourDTO>	
3. /api/tours/{id}		
Http Method	GET	Retrieves details of a specific tour by its ID
Parameter 1	Long (id)	
Return	TourDTO	
4. /api/tours/{id}		
Http Method	PUT **The tour data to be updated must be received in the controller using @RequestBody.**	Updates details of a specific tour

Parameter 1	Long (id)	
Return	TourDTO	

5. /api/tours/{id}		Deletes a specific tour
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

6. /api/tours/search		Searches tours based on a title or description
Http Method	GET	
Request Parameter 1	String (query)	
Return	List<TourDTO>	

7. /api/tours/filter		Filters tours based on location, price range, and rating
Http Method	GET	
Request Parameter 1	location	
Request Parameter 2	priceRange	
Request Parameter 3	rating	
Return	List<TourDTO>	

4.3 BOOKINGCONTROLLER

URL Exposed		Purpose
1. /api/bookings		Retrieves all bookings made by a user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<BookingDTO>	
2. /api/bookings		Books a tour for a user
Http Method	POST **The booking data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	BookingDTO	
3. /api/bookings/{id}		Retrieves a specific booking by its ID
Http Method	GET	
Parameter 1	Long (id)	
Return	BookingDTO	

4. /api/bookings/{id}		Cancels a specific booking
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

5. /api/bookings/upcoming		Retrieves all upcoming bookings for a user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<BookingDTO>	

6. /api/bookings/past		Retrieves all past bookings for a user
Http Method	GET	
Parameter 1	Long (userId)	
Return	List<BookingDTO>	

4.4 REVIEWCONTROLLER

URL Exposed		Purpose
1. /api/reviews/tour/{tourId}		Retrieves all reviews for a specific tour
Http Method	GET	
Parameter 1	Long (tourId)	
Return	List<ReviewDTO>	
2. /api/reviews		Adds a review to a tour
Http Method	POST **The review data to be created must be received in the controller using @RequestBody.**	
Parameter 1	-	
Return	ReviewDTO	
3. /api/reviews/{id}		Updates a specific review
Http Method	PUT **The review data to be updated must be received in the controller using @RequestBody.**	
Parameter 1	Long (id)	

Return	ReviewDTO	

4. /api/reviews/{id}		
Http Method	DELETE	Deletes a specific review
Parameter 1	Long (id)	
Return	-	

5. /api/reviews/user/{userId}		
Http Method	GET	Retrieves all reviews made by a specific user
Parameter 1	Long (userId)	
Return	List<ReviewDTO>	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.TRIPTROVE

Resources

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

5.2 PACKAGE: COM.TRIPTROVE.REPOSITORY

Resources

Class/Interface	Description	Status

UserRepository (interface)	• Repository interface exposing CRUD functionality for User Entity.
	• It must contain the methods for:
	  ○ Finding all users by username.
	  ○ Finding all users by email.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

TourRepository (interface)	• Repository interface exposing CRUD functionality for Tour Entity.
	• It must contain the methods for:
	  ○ Searches tours based on a title or description.
	  ○ Finding/Filtering tours based on location, price range, and rating.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

BookingRepository (interface)	• Repository interface exposing CRUD functionality for Booking Entity.
	• It must contain the methods for:
	  ○ Finding all list of bookings done by user id.
	  ○ Finding all upcoming bookings for a user.
	  ○ Finding all past bookings for a specific user.
	• You can go ahead and add any custom methods as per requirements.
		Partially implemented.

ReviewRepository (interface)	• Repository interface exposing CRUD functionality for Review Entity. • It must contain the methods for: ○ Finding list of reviews by tour id.	Partially implemented.

	○ Finding list of reviews by user id. ● You can go ahead and add any custom methods as per requirements.	

5.3 PACKAGE: COM.TRIPTROVE.SERVICE

Resources

Class/Interface	Description	Status
TourService (interface)	● Interface to expose method signatures for tour 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.
BookingService (interface)	● Interface to expose method signatures for booking related functionality. ● Do not modify, add or delete any method.	Already implemented.
ReviewService (interface)	● Interface to expose method signatures for review related functionality. ● Do not modify, add or delete any method.	Already implemented.

5.4 PACKAGE: COM.TRIPTROVE.SERVICE.IMPL

Class/Interface	Description	Status
TourServiceImpl (class)	• Implements TourService. • Contains template method implementation. • Need to provide implementation for tour 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.
BookingServiceImpl (class)	• Implements BookingService. • Contains template method implementation. • Need to provide implementation for booking related functionalities. • Do not modify, add or delete any method signature	To be implemented.
ReviewServiceImpl (class)	• Implements ReviewService. • Contains template method implementation. • Need to provide implementation for review related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.TRIPTROVE.CONTROLLER

Resources

Class/Interface	Description	Status
TourController (Class)	• Controller class to expose all rest-endpoints for tour 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
BookingController (Class)	• Controller class to expose all rest-endpoints for booking related activities. • May also contain local exception handler methods	To be implemented
ReviewController (Class)	• Controller class to expose all rest-endpoints for review related activities. • May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.TRIPTROVE.DTO

Resources

Class/Interface	Description	Status
TourDTO (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.
BookingDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.
ReviewDTO (Class)	Use appropriate annotations for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM.TRIPTROVE.ENTITY

Resources

Class/Interface	Description	Status
Tour (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 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.
Booking (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 **booking table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.
Review (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 **review table**. • Generate the **id** using the IDENTITY strategy.	Partially implemented.

5.8 PACKAGE: COM.TRIPTROVE.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the user/ tour/booking/review 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 **NotFoundException** 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.**