System Requirements Specification

Index

For

My Travel Buddy

Version 1.0

TABLE OF CONTENTS

MY TRAVEL BUDDY

System Requirements Specification

You need to only work on the backend part. Please ignore the frontend angular part.

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

"**My Travel buddy**" is an express js application designed to provide a seamless online travel planning experience. It leverages the Express Js framework with MongoDB as the database. This platform aims to connect travelers with nature, allowing users to browse, search for, and plan for a wide range of destinations.

Following is the requirement specifications:

	My Travel Buddy
Modules	
1	Auth
2	Booking
3	Destination
4	Review
5	TripPlan
6	User
Auth Module Functionalities	
1	Login user
2	Change password

Booking Module Functionalities	
1	Get all bookings
2	Create booking
3	Search booking
4	Get upcoming bookings list
5	Get booking details by id
6	Updated booking details by id
7	Delete booking by id

Destination Module Functionalities	
1	Create destination
2	Get all destinations
3	Get top rated destinations
4	Search destinations by name
5	Search destinations by budget
6	Get destination details
7	Update destination
8	Delete destination

Review Module Functionalities	
1	Get all reviews
2	Create review
3	Search reviews by destination
4	Search reviews by rating
5	Get review details
6	Update review details
7	Delete review

TripPlan Module Functionalities	
1	Create a new trip
2	Get popular trip plans
3	Search trip plan by destination
4	Get trip plan by user
5	Get all trip plans
6	Get trip plan by by id
7	Update trip plan by id
8	Delete trip plan by id

User Module Functionalities	
1	Create a new user
2	Get upcoming trips for that user
3	Get past trips for that user
4	Get trip plans for that user
5	Get bookings for that user
6	Get reviews for that user
7	Get user details

8	Update user details
9	Delete user

2 Assumptions, Dependencies, Risks / Constraints

2.1 Auth Constraints

1. When logging, email and password must have properties, on failing it should throw a custom exception.
2. When changing password, userId and newPassword must have properties, on failing it should throw a custom exception.

2.2 Booking Constraints

1. All functionalities are authenticated. That means all routes must be authenticated (must be accessed with a valid token).
2. When creating booking user and destination information are mandatory fields, on failing it should throw a custom exception.
3. When searching booking status and destinationName must be part of the query, failing should throw a custom exception.
4. When updating a booking, if the booking ID does not exist, the operation should throw a custom exception.
5. When removing a booking , if the booking ID does not exist, the operation should throw a custom exception.

2.3 Destination Constraints

1. When creating a destination name is mandatory fields, on failing it should throw a custom exception.
2. When fetching a destination by ID, if the destinationId does not exist, the operation should throw a custom exception.
3. When updating a destination, if the destinationId does not exist, the operation should throw a custom exception. And it must be an authenticated request.
4. When removing a destination, if the destinationId does not exist, the operation should throw a custom exception. And it must be an authenticated request.
5. When searching a destination, name and category are mandatory fields, on failing it should throw a custom exception.
6. When searching a destination by budget, min and max fields are mandatory in query, on failing it should throw a custom exception.

2.4 Review Constraints

1. When creating a review, user, destination and rating are mandatory fields, on failing it should throw a custom exception. And it must be an authenticated request.
2. When searching a review by destination, if the destinationName does not exist in the query, the operation should throw a custom exception.
3. When searching a review by rating, if the min & max does not exist in the query, the operation should throw a custom exception.
4. When getting a review, if the reviewId does not exist, the operation should throw a custom exception.
5. When updating a review, if the reviewId does not exist, the operation should throw a custom exception.
6. When deleting a review, if the reviewId does not exist, the operation should throw a custom exception. And it must be an authenticated request.

2.5 Trip Plan Constraints

1. When creating a trip-plan, user, and destination are mandatory fields, on failing it should throw a custom exception.
2. When searching a trip-plan, if the destinationName or min & max fields do not exist, the operation should throw a custom exception.
3. When getting a trip-plan, if the tripPlanId does not exist, the operation should throw a custom exception.
4. When updating a trip-plan, if the tripPlanId does not exist, the operation should throw a custom exception.
5. When removing a trip-plan, if the tripPlanId does not exist, the operation should throw a custom exception.

2.6 User Constraints

1. When creating an user, username, email and password are mandatory fields, on failing it should throw a custom exception.
2. When logging, if the email and password does not exist, it should throw a custom exception.

Common Constraints

1. All the database operations must be implemented in serviceImpl file only.
2. Do not change, add, remove any existing methods in the service file.
3. In the service layer, custom methods can be added as per requirements.
4. All RestEndpoint methods and Exception Handlers must return data in json format.
5. Any type of authentication and authorisation must be added in routes file only.

3 REST ENDPOINTS

Rest End-points to be exposed in the routes file and attached with controller method along with method details for the same to be created. Please note, that these all are required to be implemented.

3.1 AUTH RESTPOINTS

URL Exposed	Purpose
1. /api/auth/login Http Method: POST Parameter: - Return: token	Logins the user
2. /api/auth/changePassword Http Method: POST Parameter: - Return: Password changed successfully as message	Changes the password

3.2 BOOKING RESTPOINTS

Note: All routes must be authenticated.

URL Exposed	Purpose
1. /api/bookings Http Method: GET Parameter: - Return: list of bookings	Fetches all the bookings
2. /api/bookings Http Method: POST Parameter: - Return: new booking	Creates a new booking
3. /api/bookings/search Http Method: GET Parameter: - Return: list of bookings	Search bookings
4. /api/bookings/upcoming Http Method: GET Parameter: - Return: list of bookings	List of all upcoming bookings

5. /api/bookings/:bookingId Http Method: GET Parameter: bookingId Return: booking	Fetches the booking details
6. /api/bookings/:bookingId Http Method: PUT Parameter: bookingId Return: updated booking	Updates a booking

7. /api/bookings/:bookingId Http Method: DELETE Parameter: bookingId Return: deleted booking	Deletes a booking

3.3 DESTINATION RESTPOINTS

Note: Authenticated routes are marked as "isSecured".

URL Exposed	Purpose
1. /api/destinations Http Method: POST Parameter: - Return: created destination	Creates a new destination
2. /api/destinations Http Method: GET Parameter: - Return: list of destinations	Fetches all destinations

3. /api/destinations/top-rated Http Method: GET Parameter: - Return: list of destinations	Fetches all top rated destinations
4. /api/destinations/search Http Method: GET Parameter: - Query: name, category Return: list of destinations	Searches the destination by name

5. /api/destinations/search/budget Http Method: GET	Searches the destinations by budget

Parameter	-	
Query	min, max	
Return	list of destinations	
6. /api/destinations/:destinationId		Gets a destination
Http Method	GET	
Parameter	destinationId	
Return	destination	

7. /api/destinations/:destinationId		Updates a destination **[isSecured]**
Http Method	PUT	
Parameter	destinationId	
Return	updated destination	

8. /api/destinations/:destinationId		Deletes a destination **[isSecured]**
Http Method	DELETE	
Parameter	destinationId	
Return	deleted destination	

3.4 Review Restpoints

Note: Authenticated routes are marked as "isSecured".

URL Exposed		Purpose
1. /api/reviews		Fetches all reviews
Http Method	GET	
Parameter	-	
Return	list of all reviews	
2. /api/reviews		Creates a new review **[isSecured]**
Http Method	POST	
Parameter	-	
Return	created review	

3. /api/reviews/search		Fetches all searched reviews
Http Method	GET	
Parameter	-	
Query	destinationName	
Return	list of reviews	
4. /api/reviews/search/rating		Searches the reviews by rating
Http Method	GET	

Parameter	-	
Query	min, max	
Return	list of reviews	

5. /api/reviews/:reviewId		Gets a review
Http Method	GET	
Parameter	reviewId	
Return	review	

6. /api/reviews/:reviewId		Updates a review
Http Method	PUT	
Parameter	reviewId	
Return	updated review	

7. /api/reviews/:reviewId		Deletes a review **[isSecured]**
Http Method	DELETE	
Parameter	reviewId	
Return	deleted review	

3.5 Trip Plan Restpoints

Note: Authenticated routes are marked as "isSecured".

URL Exposed		Purpose
1. /api/trips		Creates a new trip plan
Http Method	POST	
Parameter	-	
Return	created trip plan	
2. /api/trips/popular		Fetches a list of all trip plans
Http Method	GET	
Parameter	-	
Return	list of trips	

3. /api/trips/search		Fetches all searched trips by destinationName or min and max budget
Http Method	GET	
Parameter	-	
Query	destinationName, min max	
Return	list of trips	
4. /api/trips/me		Fetches all trips of that user **[isSecured]**
Http Method	GET	

Parameter	-	
Return	list of trips	

5. /api/trips/		Fetches list of all trips
Http Method	GET	
Parameter	-	
Return	list of all trips	

6. /api/trips/:tripPlanId		Fetches a trip plan
Http Method	PUT	
Parameter	tripPlanId	
Return	trip plan	

7. /api/trips/:tripPlanId		Updates a trip plan
Http Method	PUT	
Parameter	tripPlanId	
Return	updated trip plan	

8. /api/trips/:tripPlanId		Deletes a trip plan
Http Method	DELETE	
Parameter	tripPlanId	
Return	deleted trip plan	

3.6 User Restpoints

Note: Authenticated routes are marked as "isSecured".

URL Exposed		Purpose
1. /api/users		Creates a new user
Http Method	POST	
Parameter	-	
Return	created user	
2. /api/users/upcoming-trips		Fetches a list of all upcoming trips **[isSecured]**
Http Method	GET	
Parameter	-	
Return	list of upcoming trips	

3. /api/users/past-trips		Fetches all list of past trips **[isSecured]**
Http Method	GET	
Parameter	-	
Return	list of past trips	

4. /api/users/all-trips		
Http Method	GET	Fetches all trips of that user **[isSecured]**
Parameter	-	
Return	list of all trips	

5. /api/users/bookings		
Http Method	GET	Fetches all bookings of that user **[isSecured]**
Parameter	-	
Return	list of all bookings	

6. /api/users/reviews		
Http Method	GET	Fetches all reviews of that user **[isSecured]**
Parameter	-	
Return	list of all reviews	

7. /api/users		
Http Method	GET	Fetches user **[isSecured]**
Parameter	-	
Return	user	

8. /api/users		
Http Method	PUT	Updates user **[isSecured]**
Parameter	-	
Return	updated user	

9. /api/users		
Http Method	DELETE	Deletes user **[isSecured]**
Parameter	-	
Return	deleted user	

4 TEMPLATE CODE STRUCTURE

4.1 Auth code structure

1) MODULES/AUTH: controller

Resources

AuthController (Class)	This is the controller class for the auth module.	To be implemented

2) MODULES/AUTH: middleware

Resources

File	Description	Status
authGuard (function)	Function to implement authentication	To be implemented

3) MODULES/AUTH: route

Resources

File	Description	Status
Auth routes	Routes for auth	Partially implemented.

4.2 Booking code structure

1) MODULES/BOOKING: controller

Resources

BookingController (Class)	This is the controller class for the booking module.	To be implemented

2) MODULES/BOOKING: dao

Resources

File	Description	Status
models/booking model	Model for booking	Already implemented
schemas/booking schema	Schema for booking	Already implemented

3) MODULES/BOOKING: routes

Resources

File	Description	Status
Booking routes	Routes for booking	Partially implemented.

4) MODULES/BOOKING: service

Resources

Class	Description	Status
BookingService	• Defines BookingService	Already implemented.

5) MODULES/BOOKING: service/impl

Resources

Class	Description	Status
BookingServiceImpl	• Implements BookingService.	To be implemented.

4.3 Destination code structure

1) MODULES/DESTINATION: controller

Resources

DestinationController (Class)	This is the controller class for the destination module.	To be implemented

2) MODULES/DESTINATION: dao

Resources

File	Description	Status
models/destination model	Model for destination	Already implemented
schemas/destination schema	Schema for destination	Already implemented

3) MODULES/DESTINATION: routes

Resources

File	Description	Status
Destination routes	Routes for order	Partially implemented.

4) MODULES/DESTINATION: service

Resources

Class	Description	Status
DestinationService	• Defines DestinationService	Already implemented.

5) MODULES/DESTINATION: service/impl

Resources

Class	Description	Status
DestinationServiceImpl	• Implements DestinationService.	To be implemented.

4.4 Review code structure

1) MODULES/REVIEW: controller

Resources

ReviewController (Class)	This is the controller class for the review module.	To be implemented

2) MODULES/REVIEW: dao

Resources

File	Description	Status
models/review model	Model for review	Already implemented
schemas/review schema	Schema for review	Already implemented

3) MODULES/REVIEW: routes

Resources

File	Description	Status
Review routes	Routes for review	Partially implemented.

4) MODULES/REVIEW: service

Resources

Class	Description	Status
ReviewService	• Defines ReviewService	Already implemented.

5) MODULES/REVIEW: service/impl

Resources

Class	Description	Status
ReviewServiceImpl	• Implements ReviewService.	To be implemented.

4.5 Trip Plan code structure

1) MODULES/TRIP PLAN: controller

Resources

TripPlanController (Class)	This is the controller class for the trip plan module.	To be implemented

2) MODULES/TRIP PLAN: dao

Resources

File	Description	Status
models/trip plan model	Model for trip plan	Already implemented
schemas/trip plan schema	Schema for trip plan	Already implemented

3) MODULES/TRIP PLAN: routes

Resources

File	Description	Status
TripPlan routes	Routes for trip plan	Partially implemented.

4) MODULES/TRIP PLAN: service

Resources

Class	Description	Status
TripPlanService	• Defines TripPlanService	Already implemented.

5) MODULES/TRIP PLAN: service/impl

Resources

Class	Description	Status
TripPlanServiceImpl	• Implements TripPlanService.	To be implemented.

4.6 User code structure

1) MODULES/USER: controller

Resources

UserController (Class)	This is the controller class for the user module.	To be implemented

2) MODULES/USER: dao

Resources

File	Description	Status
models/user model	Model for user	Already implemented
schemas/user schema	Schema for user	Already implemented

3) MODULES/USER: routes

Resources

File	Description	Status
User routes	Routes for user	Partially implemented.

4) MODULES/USER: service

Resources

Class	Description	Status
UserService	• Defines UserService	Already implemented.

5) MODULES/USER: service/impl

Resources

Class	Description	Status
UserServiceImpl	• Implements UserService.	To be implemented.

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**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

3. **This editor Auto Saves the code.**
4. **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.**
5. **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.**
6. **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.**
7. **You can follow series of command to setup express environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min**
   b. **npm run start -> To compile and run the project.**
   c. **npm run jest -> to run all test cases and see the summary of all passed and failed test cases.**
   d. **npm run test -> to run all test cases and register the result of all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**
8. **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.**