System Requirements Specification

Index

For

Simple Login and Logout System

Version 1.0

TABLE OF CONTENTS

Simple Login and Logout System
System Requirements Specification

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

"**Login and Logout System**" is an express js application designed to to have a simple Login and Logout System using hotel management system. It leverages the ExpressJs with In-Memory as the database. This platform aims to provide APIs on the hotels, allowing users to browse, search for, have logging into and logging out features.

<u>Following is the requirement specifications</u>:

	Simple Login and Logout System
Modules	
1	User
2	Hotel
3	Middleware

User Module Functionalities	
1	Register a new user

Hotel Module Functionalities	
1	Get all hotels
2	Get hotel by ID
3	Create a new hotel
4	Update hotel information

Middleware Module Functionalities	
1	Implement logic to check and validate user credentials using Basic Authentication.
2	Implement centralized error handling to return consistent error responses. (Already implemented)

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When **registering a user**, if a user with the provided email already exists, the operation should throw a custom exception with the message: **“User already exists.”**

2.2 Hotel Constraints

- When **creating a hotel**, the `name`, `location`, and `pricePerNight` fields should be mandatory. If any are missing, it should throw a custom exception with the message: **“Failed to create hotel.”**
- When **fetching a hotel by ID**, if the hotel ID does not exist, the operation should throw a custom exception with the message: **“Hotel not found.”**
- When **updating a hotel**, if the hotel ID does not exist, the operation should throw a custom exception with the message: **“Hotel not found.”**

Common Constraints

- All RestEndpoint methods and Exception Handlers must return data in json format.
- 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 User Restpoints

URL Exposed		**Purpose**
1. /api/users/register		Registers a new user
Http Method	POST	
Body	`name` (string) `email` (string) `password` (string) `age` (number)	
Return	Created user along with message: "User	

		registered successfully"	

3.2 Hotel Restpoints

Note:

Any route which must be authenticated (there must be valid token) are marked with authMiddleware

URL Exposed			Purpose
1. /api/hotels	Http Method	GET	Fetches all the hotels
	Parameter	-	
	Return	list of hotels	
2. /api/hotels/:id	Http Method	GET	Fetches a hotel by its ID
	Parameter	id	
	Return	fetched hotel	
3. /api/hotels	Http Method	POST	Creates a new hotel (protected) **[authMiddleware]**
	Body	`name` (string) `location` (string) `pricePerNight` (number)	
	Return	Newly created hotel along with message as “Hotel created successfully”	
4. /api/hotels/:id	Http Method	PUT	Updates an existing hotel (protected) **[authMiddleware]**
	Parameter	id	
	Body	`name` (string) `location` (string) `pricePerNight` (number)	
	Return	Updated hotel along	

	with message as "Hotel updated successfully"	

4 Template Code Structure

4.1 User code structure

1. User: controller

Resources

File	Description	Status
userController **(Class)**	This is the controller class for the user module.	To be implemented

2. User: models

Resources

File	Description	Status
user	Models for user	Already implemented

3. User: routes

Resources

File	Description	Status
userRoutes	Routes for user	To Be iimplemented

4.2 Hotel code structure

1. Hotel: controller

Resources

File	Description	Status
hotelController **(Class)**	This is the controller class for the hotel module.	To be implemented

2. Hotel: models

Resources

File	Description	Status
hotel	Models for hotel	Already implemented

3. Hotel: routes

Resources

File	Description	Status
userRoutes	Routes for hotel	To be implemented

4.3 MiddleWare

Resources

File	Description	Status
authMiddleware	Middleware that performs Basic Authentication using username and password validation..	To be implemented
errorHandler	Centralized error-handling middleware that returns consistent error responses across the application.	Already implemented

5 METHOD DESCRIPTIONS

5.1 Controller - Method Descriptions:

1. UserController Class - Method Descriptions:

Method	**Task**	**Implementation Details**
registerUser	To register a new user	- The request type should be POST with URL /api/users/register. - Accepts name, email, password, and age from the request body. - Checks if a user with the given email already exists in the in-memory users array (read from user model). - If a user exists, return 400 status with message: 'User already exists'. - If not, hashes the password using bcrypt with a salt round of 10. - Creates a new user object with a hashed password and pushes it into the users array. - Return 201 status with message: 'User registered successfully' and the user data (excluding plain password). - On error during hashing, throws the error.

2. HotelController Class - Method Descriptions:

Note:- Data must be read from the hotel model as this is an in-memory project.

Method	**Task**	**Implementation Details**
getAllHotels	To retrieve all hotels (public route)	- The request type should be GET with URL /api/hotels. - Responds with status 200 and the list of all hotels. - This is a public route with no authentication required.

getHotelById	To retrieve a hotel by its ID (public route)	- The request type should be GET with URL parameter /api/hotels/:id. - Searches the hotel list using the provided ID. - If found, respond with status 200 and the hotel object. - If not found, respond with status 404 and message: 'Hotel not found'.
createHotel	To create a new hotel (protected route)	- The request type should be POST with URL /api/hotels. - Accepts name, location, and pricePerNight from request body. - Creates a new hotel with a unique ID and adds it to the array. - Responds with status 201 and message: 'Hotel created successfully' along with the created hotel.
updateHotel	To update an existing hotel by ID (protected route)	- The request type should be PUT with URL parameter /api/hotels/:id. - Finds the hotel by ID (param) and updates provided fields: name, location, pricePerNight (from body). - If not found, respond with 404 and message: 'Hotel not found'. - Otherwise, respond with 200 and message: 'Hotel updated successfully' along with the updated hotel.

5.2 Routes - Descriptions:

1. User Routes:

Method	**Task**	**Implementation Details**
registerUser	To register a new user	- Create a POST route at /register. - Call userController.registerUser inside the route. - Use express.Router() to define and export the route.

2. Hotel Routes:

Method	**Task**	**Implementation Details**
getAllHotels	To fetch all hotels	- Define a GET route at /hotels. - Call hotelController.getAllHotels.
getHotelById	To fetch a hotel by its ID	- Define a GET route at /hotels/:id. - Call hotelController.getHotelById.
createHotel	To create a new hotel	- Define a POST route at /hotels. - Use authMiddleware for Basic Auth. - Call hotelController.createHotel.
updateHotel	To update an existing hotel by its ID	- Define a PUT route at /hotels/:id. - Use authMiddleware for Basic Auth. - Call hotelController.updateHotel.

5.3 MiddleWare - Method Descriptions:

Middleware	Purpose	Implementation Details
authMiddleware	To authenticate users using Basic Authentication	- Checks for the 'Authorization' header and validates its format. - If missing or malformed, returns 401 with message: 'Authorization header is missing or malformed'. - Decodes the base64-encoded credentials to retrieve username and password. - If username or password is missing, responds with 400 and message: 'Username and password are required'. - Looks up the user from an in-memory user list by email.

		- If user not found, responds with 400 and message: 'Invalid username or password'. - Compares the provided password with the stored password hash using bcrypt.compare. - Logs whether the password is correct or not as “Password is correct!” or “Password is incorrect!” respectively. - If match is found, continues to the next middleware using next(). - On bcrypt error, logs the error and does not proceed.

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.