System Requirements Specification

Index

For

User Management API

User Management API
System Requirements Specification

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **User Management API** is an Express application designed to manage user records in-memory. It provides RESTful endpoints to perform CRUD operations and allows filtering users by name or email. The application demonstrates basic Express middleware usage.

Following is the requirement specifications:

	User Management API
Modules	
1	User

User Module Functionalities	
1	Get all users
2	Get user by ID
3	Filter users
4	Create user
5	Update user
6	Delete user

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When retrieving a user by ID, the operation should throw a custom exception with the message: **"User not found."**
- When updating a user by ID, if the user with the provided `id` does not exist, the operation should throw a custom exception with the message: **"User not found."**
- When deleting a user by ID, if the user with the provided `id` does not exist, the operation should throw a custom exception with the message: **"User not found."**
- When accessing an invalid route, if no matching endpoint is found for the request, the operation should throw a custom exception with the message: **"Not Found."**

Common Constraints

- All RestEndpoint methods and Exception Handlers must return data in json format.
- **For any invalid route**, a standardized `404 Not Found` response must be returned in JSON.

3 Template Code Structure

3.1 App.js

Resources

File	**Description**	**Status**
app.js	Need to add: ● Root route for basic health check ● Implement user routes: list, filter, create, update, delete ● Catch-all error route	Partially implemented

4 Method Descriptions

4.1 App.js - Method Descriptions:

Method	**Task**	**Implementation Details**
rootRoute	Health check / welcome route	- The request type should be **GET** with URL `/` - Returns a static JSON response with message: `"Welcome to the User API!"` - Status: **200 OK**

getAllUsers	Fetch all users	- The request type should be **GET** with URL `/users/filter` - Accepts query parameters `name` and/or `email` - Starts with the complete `users` array - If `name` is provided: - Convert both `u.name` and query `name` to lowercase - Use `.includes()` to check if user's name contains the query string - If `email` is provided: - Convert both `u.email` and query `email` to lowercase - Use `.includes()` to filter by email - Respond with the filtered array and **status 200 OK**
getAllUsers	Fetch all users	- The request type should be **GET** with URL `/users` - Directly return the full in-memory array `users` as a JSON response - Respond with **status 200 OK**
getUserById	Get a user by ID	- The request type should be **GET** with URL `/users/:id` - Extract the `id` from request parameters and convert it to an integer - Use `.find()` to search for a user with the matching ID - If found, respond with user object and **status 200 OK** - If not found, respond with **status 404** and message: `"User not found"`
createUser	Create a new user	- The request type should be **POST** with URL `/users` - Extract `name` and `email` from request body - Create a new user object with `id.` - Push the new user object into the `users` array - Respond with the newly created user object and **status 201 Created**
updateUserById	Update an existing user by ID	- The request type should be **PUT** with URL `/users/:id` - Extract `id` from params and find the user using `.find()` - If not found, return **status 404** with `"User not found"` - Extract `name` and `email` from request body - If provided, update user fields (`user.name`, `user.email`) conditionally - Respond with the updated user object and **status 200 OK**

deleteUserById	Delete a user by ID	- The request type should be **DELETE** with URL `/users/:id` - Parse `id` and use `.findIndex()` to locate the user - If index is `-1`, return **status 404** with `"User not found"` - Use `.splice()` to remove the user from the array - Return a JSON message `"User deleted successfully"` with **status 200 OK**
notFoundRoute	Catch-all for undefined endpoints	- Applies to all undefined routes - Responds with **status 404** - Returns a JSON object: `{ "message": "Not Found" }` - Helps ensure consistency in error handling for unmatched routes

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. **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.**
5. **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.**
6. **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**