System Requirements Specification

Index

For

Post Management API

Post Management API

System Requirements Specification

BACKEND-EXPRESS RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Post Management API** is an Express application designed to manage blog post records in-memory. It provides RESTful endpoints to perform CRUD operations and allows filtering posts by title or content. The application showcases basic Express.js features, including routing, middleware usage, and error handling.

Following is the requirement specifications:

	Post Management API
Modules	
1	Post

Post Module Functionalities	
1	Get all posts
2	Get post by ID
3	Filter posts
4	Create post
5	Update post
6	Delete post

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When retrieving a post by ID, the operation should throw a custom exception with the message: **"Post not found."**
- When updating a post by ID, if the post with the provided `id` does not exist, the operation should throw a custom exception with the message: **"Post not found."**
- When deleting a post by ID, if the post with the provided `id` does not exist, the operation should throw a custom exception with the message: **"Post 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 post 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 Post API!"` - Responds with **status 200 OK**

filterPosts	Filter posts by title/content	- The request type should be **GET** with URL `/posts/filter`
		- Accepts optional query parameters `title` and/or `content`
		- Starts with the full `posts` array
		- If `title` is provided:
		  - Convert both `p.title` and query `title` to lowercase
		  - Use `.includes()` for partial match
		- If `content` is provided:
		  - Convert both `p.content` and query `content` to lowercase
		  - Use `.includes()` for partial match
		- Responds with filtered array and **status 200 OK**

getAllPosts	Fetch all posts	- The request type should be **GET** with URL `/posts` - Returns the entire in-memory `posts` array as a JSON response - Responds with **status 200 OK**
getPostById	Get a post by ID	- The request type should be **GET** with URL `/posts/:id` - Extracts `id` from the route parameter and converts to integer - Uses `.find()` to locate a post with matching `id` - If found, responds with the post and **status 200 OK** - If not found, responds with **status 404** and message: `"Post not found"`
createPost	Create a new post	- The request type should be **POST** with URL `/posts` - Extracts `title` and `content` from the request body - Creates a new post with `id = posts.length + 1` - Adds it to the `posts` array using `.push()` - Returns the newly created post and **status 201 Created**
updatePostById	Update an existing post by ID	- The request type should be **PUT** with URL `/posts/:id` - Extracts `id` from URL and locates the post using `.find()` - If not found, responds with **status 404** and message: `"Post not found"` - Extracts `title` and/or `content` from the body - Updates the post fields if provided - Returns the updated post with **status 200 OK**

deletePostById	Delete a post by ID	- The request type should be **DELETE** with URL `/posts/:id` - Parses `id` and uses `.findIndex()` to locate the post index - If not found, responds with **status 404** and message: `"Post not found"` - If found, deletes post using `.splice()` - Responds with JSON message `"Post deleted successfully"` and **status 200 OK**
notFoundRoute	Catch-all for undefined endpoints	- Catches all unhandled routes using middleware - Responds with **status 404** - Returns a standardized JSON object: `{ "message": "Not Found" }`

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**