PER SCHOLAS

ALAB 318.3.1:

Expanding a RESTful API

Version 1.0, 07/23/23
Click here to open in a separate window.

Introduction

This assignment will ask you to expand the example REST API application that was explored during the lesson, adding additional routes and features that are common with an API of its kind.

Feel free to reference the lesson materials, express documentation, and other resources when creating these additional features, but think critically about how you want to approach your own solution. Sometimes, the way things are presented the first time are not the best way to do them!

When working on a team, especially with legacy code, you can never assume that something is correct just because it was previously the accepted way of doing it. Look for ways to increase the efficiency of the application and the efficiency of the development process!

Objectives

- Add additional features to an existing RESTful Express API.
- Refactor existing code for efficiency, organization, and/or performance.

Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Your submission should include:

- A link to the GitHub repository for your project.

Instructions

You will begin with the example application that was explored during the lesson.

- If you have already created a working local copy of that application, wonderful! Continue with that as the starting point for this assignment.
- Otherwise, download the final example CodeSandbox using the top-left menu.

- Once downloaded, rename the directory to an appropriate project name.
- Run git init to establish a git repository within the directory.
- Run npm install to install the application dependencies.
- Test the application by running nodemon index.js and navigating to localhost:3000.

Throughout the assignment, commit frequently to your Git repository.

Part 1: Exploring Existing Routes

Take a few minutes to explore the existing code again. Make sure that you are familiar with the structure and functionality of what you will be working with. Whenever you are given somebody else's code to work on, it is important to take the appropriate time to understand it before attempting modifications and improvements.

The application has the following routes as a starting point:

- `GET /`
  - `GET /api`
    - `GET /api/users`
    - `POST /api/users`
      - `GET /api/users/:id`
      - `PATCH /api/users/:id`
      - `DELETE /api/users/:id`
    - `GET /api/posts`
    - `POST /api/posts`
      - `GET /api/posts/:id`
      - `PATCH /api/posts/:id`
      - `DELETE /api/posts/:id`

Part 2: Adding Additional Routes

Your task is to add the following additional routes and any code necessary to make them work as described. The pre-existing routes should remain functional! Assume that you have clients actively using the routes above, so maintaining them is necessary for the health of the application.

If you do not finish every route below, that is okay! These are for practice purposes, and will not be graded. You should attempt to finish as many as possible in order to better prepare yourself for future projects, assignments, and assessments.

Create the following routes, using good organizational and coding practices:

- `GET /api/users/:id/posts`
  - Retrieves all posts by a user with the specified id.
- `GET /api/posts?userId=<VALUE>`
  - Retrieves all posts by a user with the specified postId.

    - It is common for APIs to have multiple endpoints that accomplish the same task. This route uses a "userId" query parameter to filter posts, while the one above uses a route parameter.
- `GET /comments`
    - Note that we do not have any comments data yet; that is okay! Make sure that you create a place to store comments, but you do not need to populate that data.
- `POST /comments`
    - When creating a new comment object, it should have the following fields:
        - **id**: a unique identifier.
        - **userId**: the id of the user that created the comment.
        - **postId**: the id of the post the comment was made on.
        - **body**: the text of the comment.
- `GET /comments/:id`
    - Retrieves the comment with the specified id.
- `PATCH /comments/:id`
    - Used to update a comment with the specified id with a new body.
- `DELETE /comments/:id`
    - Used to delete a comment with the specified id.
- `GET /comments?userId=<VALUE>`
    - Retrieves comments by the user with the specified userId.
- `GET /comments?postId=<VALUE>`
    - Retrieves comments made on the post with the specified postId.
- `GET /posts/:id/comments`
    - Retrieves all comments made on the post with the specified id.
- `GET /users/:id/comments`
    - Retrieves comments made by the user with the specified id.
- `GET /posts/:id/comments?userId=<VALUE>`
    - Retrieves all comments made on the post with the specified id by a user with the specified userId.
- `GET /users/:id/comments?postId=<VALUE>`
    - Retrieves comments made by the user with the specified id on the post with the specified postId.

Quickly, you can begin to see how data is interconnected, leading to difficulties with scaling an application as the categories of data continue to grow. How might we have created this in a way that allows it to scale more easily?

This illustrates the importance of thinking about how the data will be structured *before* you begin your journey of creating an API to handle the data.

For example, could we have created a single piece of middleware to handle all possible filtering via query parameters? Yes, we could have! Now, however, it would require refactoring much of our code in order to do so. If you are up for the challenge (and have the time), feel free to explore this possibility.

Part 3: Testing

Test your routes!

Make sure that everything works as expected, and remember that any changes to the data will be reset when the server restarts since we are not using a persistent database.

If code does not work, leave comments for yourself explaining potential next steps so you can revisit the application in the future and approach the problem again from a new perspective.

If code prevents the application from running, comment it out before submission.

Part 4: Completion

Upload your project to a GitHub repository, and submit it according to the submission instructions at the beginning of this document.