COMPSCI 326 Web Programming

Milestone 03 Back-End

04/26/2024 Version 1.0.0 (Tim Richards): Initial draft.

Table of Contents

Introduction

In this third phase of our project assignment, you will shift your focus from front-end design to back-end functionality, implementing the server-side of your web application using Express.js and Node.js. This milestone challenges you and your team to construct a robust back-end infrastructure that not only handles HTTP requests efficiently but also manages data through PouchDB, a lightweight JavaScript database. Your tasks include developing a variety of Express routes to support the operational needs of your front-end, specifically ensuring that there are at least one GET, POST, PUT, and DELETE route, though a well-rounded application will typically require more. Additionally, you will transition the front-end from using mocked data to making real HTTP requests with the **fetch()** function, thereby integrating it seamlessly with your newly developed back-end. This milestone is crucial for learning how to manage server-side logic and database operations, ensuring your application is dynamic, interactive, and capable of handling real-world data and traffic.

Objectives

The objectives for Milestone 03 are designed to provide a clear roadmap for your progression through the process of developing a back-end. Each objective targets a specific skill or area of knowledge that is crucial for building a well-rounded full-stack web application. By focusing on these objectives, you will gain hands-on experience with server-side programming, data management, and the integration of front-end and back-end technologies. This structured approach ensures that you develop a comprehensive understanding of how different components of a web application interact with each other in a real-world scenario. Additionally, adhering to these objectives will help you produce a robust and maintainable codebase, preparing you for future challenges in the field of web development. The objectives are not just checkpoints but steppingstones that build upon one another, culminating in a fully functional and user-responsive application with a back-end that you can be proud of.

1. **Implement Express.js Server**: Establish a server using Express.js that can handle HTTP requests and responses efficiently.
2. **Database Integration with PouchDB**: Set up and configure PouchDB to store and manage application data. Understand how to perform CRUD operations with the database.
3. **Develop RESTful Routes**: Create a range of Express routes to support the functionality of the front-end. Ensure that there are at least one GET, POST, PUT, and DELETE route, with the expectation of additional routes to support a fully functional application.
4. **Integrate Front-end with Back-end**: Modify the front-end code to replace mocked data interactions with actual HTTP requests using the **fetch()** function, ensuring seamless communication between the front-end and back-end. If you designed this right in your front-end from Milestone 02, this should be easy to make this transition.
5. **Error Handling**: Implement robust error handling in the server-side code to manage and respond to errors effectively, enhancing the reliability of the web application.
6. **Testing Backend Functionality**: Conduct thorough testing of the back-end to verify that all routes function correctly and interact as expected with the front-end.
7. **Code Documentation and Commenting**: Add sensible comments throughout the code to explain the functionality of each section. Use JSDoc to document all functions and modules, providing clear, professional documentation that aids in maintenance and scalability.

Requirements

It is important to understand the specific requirements needed for a successful submission. This phase of the project is all about bringing to life the server-side functionality of your web application using Express.js and integrating it with a PouchDB database. You will demonstrate your ability to set up a robust back-end infrastructure that can handle real-world data and user requests effectively. This milestone not only tests your technical skills in developing API routes and managing database operations but also assesses your capability to integrate these back-end services with the front-end components you've previously built. Successful completion of this milestone will show your proficiency in creating a dynamic, interactive, and fully functional web application. Pay close attention to the detailed requirements that follow, as these will guide you in building a robust back-end.

1. **Project Structure**:
    - Organize your front-end code within a **src/server** directory. This structure is essential for maintaining a clean and manageable codebase as your project grows. All front-end code should reside in **src/client** and all back-end code should reside in **src/server**. It is fine if you have other folders to support the organization of your application, but this must be clearly documented in the **README.md** file to provide guidance to submission reviewers.
2. **Server Setup**:

   - Deploy an Express.js server that can start without errors.
   - Ensure the server listens on a specified port and can handle basic HTTP requests.
3. **Database Configuration**:
   - Implement PouchDB within the server environment.
   - Ensure the database is connected properly and can perform CRUD operations.
4. **API Routes**:
   - Implement at least the following four routes with corresponding HTTP methods:
     - **GET**: At least one route to retrieve data.
     - **POST**: At least one route to create new entries.
     - **PUT**: At least one route to update existing entries.
     - **DELETE**: At least one route to delete entries.
   - Include additional routes as necessary to support all required functionalities of the application.
5. **Front-end Integration**:
   - Modify the existing front-end code to replace mock data with actual server calls using the **fetch()** method.
   - Ensure all front-end pages or components that require back-end data can retrieve and send data to the server correctly.
6. **Front-End Storage:**
   - Include at least one place in your front-end that utilizes PouchDB to store data. This data should be data fetched from the server but stored in the browser for quick and efficient access. This data is typically data that can only change on the front-end but would also be saved on the back-end if changed. For example, user profile data, a dictionary of words, a shopping cart, a list of items, etc.
7. **Error Handling**:
   - Implement error handling for each route to manage and respond to different errors, such as 404 (Not Found) and 500 (Server Error).
   - Ensure that errors are logged on the server console and appropriate responses are sent to the front-end.
8. **Development Script**:
   - Add a **start** script to your **package.json** that runs your application.
   - This script enhances the ease of testing and presentation of your project.
9. **Documentation**:
   - Provide comprehensive JSDoc comments for all functions, routes, and modules.
   - Include/extend a **README.md** file in the root of your project with:
     - Instructions on how to set up and run the server.
     - A brief description of the API routes and their functionalities.
10. **Code Quality**:
    - Ensure the code is well-organized and follows consistent naming conventions.
    - Code should be modular, with separate files for routes, configurations, and database operations.
11. **Submission Format**:

- Submit all source code files and necessary configuration files.
- Include a **package.json** file with all dependencies specified, ensuring the project can be installed and run by simply executing **npm install** and **npm start**.

Team Collaboration and Contribution

It is imperative for the success of your project that all team members actively contribute to the development process, including committing and pushing changes to your GitHub repository. This collaborative effort is not only a cornerstone of agile development practices but also ensures a diversified skill application throughout your project's life-cycle.

We will be monitoring contributions to the GitHub repository to gauge individual contributions and participation. If it becomes apparent that a team member has not contributed to the project, points will be deducted from that individual's total score. This policy is in place to encourage active involvement from all team members and to ensure a fair and equitable distribution of work.

Testing and Debugging

To ensure the robustness and reliability of your web application, thorough testing across the front-end, back-end, and full-stack is crucial. Here are some guidelines on how to effectively test each component of your application using various tools and techniques:

Testing the Front-End

1. **Using Developer Tools:**
    - Leverage the developer tools in your browser to inspect elements, debug JavaScript, and monitor network requests. This can help you understand how your HTML, CSS, and JavaScript are interacting in the live environment.
    - Use the developer tools console in the browser to observe any errors that might occur when you load and interact with your application. It should be your first step in identifying and diagnosing errors.
    - Use the console to catch and analyze JavaScript errors and logs. Regularly include **console.log()** statements in your code to track values and application flow during debugging.
2. **Interface Testing:**
    - Manually test the responsiveness of your user interface on different devices and screen sizes using the device emulation feature in your browser's developer tools.
    - Verify that all user inputs, buttons, and navigational elements function as expected and provide the correct feedback to the user.

Testing the Back-End

1. **Using Postman:**

  - Utilize Postman to test your API endpoints. Make sure to test all routes with appropriate HTTP methods (GET, POST, PUT, DELETE) and ensure they return the correct status codes and responses.
  - Test error handling by sending requests that should fail (e.g., requests with invalid data or to non-existent endpoints) and check that your API responds appropriately.

2. **Console Logging:**
   - Implement **console.log()** within your server-side code to output debugging information and internal state variables directly to the console. This is particularly useful for monitoring the flow of data and understanding how requests are being processed.

Testing Full-Stack Integration

1. **End-to-End Testing:**
   - Perform tests that simulate real user interactions from the front-end to the back-end. Ensure that the front-end correctly displays data fetched from the back-end and that user inputs correctly update the back-end.
   - Use both the browser's developer tools and Postman to monitor how data flows through your application and ensure that the integration points are working as expected.
2. **Error Handling Tests:**
   - Test how your application handles failures, such as database errors, network errors, and bad user inputs. Ensure that errors are gracefully handled in the back-end and that appropriate error messages are displayed in the front-end.

User Stories for Testing and Debugging the Front-End

Creating a user story for testing and debugging browser-based user interfaces involves outlining specific scenarios that simulate how end-users interact with the application. This approach helps to ensure that the interface is functional, user-friendly, and behaves as expected across various conditions and platforms. Here's a step-by-step guide to create a user story for this purpose:

Step 1: Define the User Role

Identify who the user is in the context of the user story. This could be a general user, an admin, or a specific persona with characteristics relevant to the scenario.

Step 2: Outline the Goal or Need

Clearly state what the user wants to achieve through this interaction with the UI. This helps focus the story on delivering value to the user.

Step 3: Describe the Interaction and Steps

Detail the specific actions that the user will take in the UI and what they expect to happen as a result. This part should be clear and concise, mimicking real-world tasks.

Step 4: Specify Conditions and Context

Include any relevant conditions or context that affect the user's interaction, such as the device being used, the browser type, or network conditions.

Step 5: Define Acceptance Criteria

List the criteria that must be met for the task to be considered successfully completed. These criteria will be used to test and debug the UI.

Example User Story for Testing and Debugging a Browser-based UI

Title: Testing the checkout process on an e-commerce site

User Story:

- **As a** regular user,
- **I want to** successfully complete a purchase using the website's checkout interface.
- **So that** I can receive the products I need without inconvenience.

Steps:

1. User navigates to the shopping cart and reviews their selected items.
2. User clicks on the 'Proceed to Checkout' button.
3. User fills out the shipping and payment details and submits the form.
4. User receives a confirmation page with a summary of the order and an order number.

Conditions:

- Tests should be performed on multiple browsers (Chrome, Firefox, Safari) and devices (desktop, tablet, smartphone).
- The checkout process should also be tested under different network conditions (Wi-Fi, 4G).

Acceptance Criteria:

- The checkout form should load and submit without errors.
- Form validations (e.g., for payment information) should work as expected and provide useful feedback.
- The confirmation page must display the correct items, prices, and total.
- The process should not exceed 30 seconds from start to finish under normal network conditions.
- Use **console.log()** to debug and track values throughout the process.
- Utilize browser developer tools to inspect element styles, monitor network requests, and check console for JavaScript errors.

This structured approach helps in setting clear expectations and provides a thorough framework for testing and debugging the UI to enhance overall usability and performance. Although this can mostly be thought of as UI testing, the UI is often dependent on the back-end to deliver data thus developing user stories can also help you weed out bugs in the back-end.

We do not ask you to submit user stories, but it is an effective method to test and debug. We highly encourage you to use this technique to thoroughly test your front-end UI and back-end.

Rubric

The rubric provided below serves as a detailed guide for evaluating the submissions for Milestone Three of our web programming project. It is designed to assess each critical component of the back-end development process, from server setup and database integration to API functionality and front-end integration. This systematic evaluation approach ensures that all aspects of the server-side application are thoroughly reviewed, highlighting areas of strength, and identifying opportunities for improvement. Each category in the rubric represents a fundamental element of back-end development, with points assigned based on the completeness, functionality, and sophistication of the work submitted. By using this rubric, students can clearly understand the expectations and standards they need to meet, while instructors can maintain consistency and fairness in grading across the cohort.

Total Points: 100

1. Server Setup and Configuration (15 points)

- **15 points**: Server is fully operational with no startup errors and handles HTTP requests as expected.
- **10 points**: Minor issues present but the server operates mostly as expected.
- **5 points**: Server setup is incomplete or contains significant errors affecting functionality.
- **0 points**: Server does not start or handle basic requests.

2. Database Integration (15 points)

- **15 points**: PouchDB is fully integrated and performs all CRUD operations correctly.
- **10 points**: PouchDB integration is mostly correct; minor issues with some CRUD operations.
- **5 points**: Significant issues with database operations; only some functionalities are working.
- **0 points**: Database is not integrated or does not function.

3. Implementation of API Routes (20 points)

- **20 points**: All required routes (GET, POST, PUT, DELETE) and additional routes are implemented correctly.
- **15 points**: Most routes are implemented correctly; minor issues in one or two routes.
- **10 points**: Significant issues with multiple routes; basic functionality is provided.
- **0 points**: Routes are incorrectly implemented or missing.

4. Front-end Integration and Browser Storage (15 points)

- **15 points**: Front-end seamlessly integrates with the back-end; all features work correctly with real data produced by the back-end. PouchDB is used on the front-end to store data that changes on the front-end but synchronizes with the back-end.

- **10 points**: Minor integration issues; most front-end features work with real data. PouchDB usage exists but doesn't synchronize with the back-end or isn't used in the front-end to display data or information in the user interface.
- **5 points**: Major integration issues; only some features work with real data. The use of PouchDB is minimal or doesn't exist.
- **0 points**: No integration or front-end does not work with back-end.

5. Error Handling (10 points)
   - **10 points**: Comprehensive error handling is implemented; appropriate errors are returned for all edge cases.
   - **7 points**: Basic error handling is implemented; minor issues in covering all edge cases.
   - **4 points**: Error handling is minimal or only covers obvious cases.
   - **0 points**: No error handling implemented.
6. Project Structure (10 points)
   - **10 points**: The project structure is exemplary and meets all specified requirements. The project includes clearly defined **src/client** and **src/server** directories. All front-end code is organized within the **src/client** directory, and all server code is found in the **src/server** directory, demonstrating clear separation of concerns. The **README.md** file is in the root of the project and has been thoroughly updated to include details about the back-end implementation, setup instructions, and how to interact with the server, enhancing project clarity and usability.
   - **7 points**: The project structure meets most of the specified requirements but has missing items. The **src/client** and **src/server** directories are not clearly defined or correctly used for all corresponding files. The **README.md** file lacks some necessary details about the back-end or does not fully document the setup and use of the server.
   - **4 points**: The project structure is partially complete but significantly deviates from the requirements in two or more of the following ways. The separation between front-end and server code is not clear, with files misplaced or mixed between the **src/client** and **src/server** directories. The **README.md** file is present but provides minimal information, missing essential details on back-end functionalities and setup.
   - **0 points**: The project structure does not meet the basic requirements. There is no clear distinction or organization with **src/client** and **src/server** directories. The **README.md** file is missing, incomplete, or does not update relevant information about the project structure or back-end details.
7. Documentation and Code Quality (15 points)
   - **15 points**: Code is well-documented with JSDoc; README is detailed; code is clean and well-organized.
   - **10 points**: Documentation is mostly complete; some issues with clarity or organization.
   - **5 points**: Minimal documentation; code lacks organization or clarity.
   - **0 points**: No documentation or poorly documented code.

Submission

You must submit only a single file to Gradescope called **github.txt** that contains a single link—the URL to your team's GitHub repository. It is essential that this repository is public, allowing for straightforward cloning without access issues. The organization and clarity of your repository are critical in ensuring an efficient review process.

- Include/update a **README.md** file at the root of your repository that details the project structure, setup instructions, and documentation needed to understand and navigate your application effectively. Update this file according to any changes in this milestone.
- Ensure that a **package.json** file is present and contains the **start** script as previously specified. All dependencies your application requires are present.
- Remember, all submissions must be completed by the given deadline. Extensions will only be granted under special circumstances through direct communication with the instructor by email.

This milestone is not just about coding; it's about integrating and fine-tuning your server-side logic to effectively communicate with your front-end, creating a seamless and dynamic user experience. By addressing these real-world technical challenges, you're laying a solid foundation for robust application development. We are eager to see your innovative approaches and the impactful solutions you devise.