Databases, Networks and the Web

Coursework for Midterm

Blogging Tool

Website Architecture:

the website uses a three-tier architecture, that consists of the Presentation Layer (Client), the Application Layer (Server) and the Data Layer (database).

Presentation Layer (Client): The client-side of the website is responsible for displaying the user interface and handling the user interactions. It is built using the HTML, CSS and JavaScript. The client interacts with the application-layer through the RESTful API endpoints. The user can view articles, create new articles, edit existing articles and perform other author-related actions through the client interface.

Application Layer (Server): The server-side of the website is responsible for processing client requests, handling business logic and interacting with the database. It is built using Node.js and Express.js. The server side provides RESTful API endpoints that the client can communicate with. These endpoints handle CRUF operations for articles and manage author settings.

Data Layer (Database): The data layer of the website stores all the relevant data, including articles and author settings. it uses a database system, and in this implementation, we use SQLite. The server communicates with the database to perform read and write operations for articles and settings.

End Points (Client to Server):

/: GET request to retrieve a list of published articles.

/new: GET request to display the create page for a new article.

/new: POST request to create a new article.

/edit/:id: GET request to display the edit page for an existing article.

/edit/:id: POST request to edit an existing article.

/publish/:articleId: POST request to publish a draft article.

/delete/:articleId: DELETE request to delete an article.

/settings: GET request to display the author settings page.

/settings: POST request to update the blog settings.

Extension Description: Quill – Rich Text Editor

I have implemented a rich text editor extension using the Quill library to enhance the author-editing experience. The implementation is visible in the 'author-edit.ejs'. The script initializes the Quill editor, specifying the modules and toolbar options for formatting the text. The extension allows authors to create and edit articles with formatted text such as bold, italic, lists, heading and more. Authors are also able to include images, links and videos to support their articles.

The implementation can be seen below.

```
  // Initialize Quill with all available modules in the toolbar
  var quill = new Quill(editorContainer, {
   theme: 'snow', // Used the 'snow' theme for a more traditional editor appearance
   modules: {
    toolbar: [
     // Include necessary toolbar options
     [{ 'header': [1, 2, 3, 4, 5, 6, false] }],
     [{ 'font': [] }],
     ['bold', 'italic', 'underline'],
     [{ 'color': [] }],
     [{ 'align': [] }],
     [{ 'list': 'ordered' }, { 'list': 'bullet' }],
     ['link', 'image', 'video'],
     [{ 'script': 'sub' }, { 'script': 'super' }],
    ],
   }
  });

  // Get the hidden textarea for form submission
  var hiddenTextarea = document.querySelector('textarea[name="body"]');

  // Update the hidden textarea content when the editor content changes
  quill.on('text-change', function () {
   hiddenTextarea.value = quill.root.innerHTML;
  });
 });
</script>
```

I have included the Quill.js CDN link in the EJS and created a container to initialize the Quill Editor. (e.g.,newQuill(editorcontainer)). In the corresponding JavaScript, i have initialized the Quill editor with the desired modules and toolbar options. The theme is also set to "snow".

Additionally, I have set up event listeners to update the hidden form field with the editor's content whenever the content changes (e.g., when an author types or formats text).

These extensions greatly contribute to the website's functionality and user experience. Express.js facilitates efficient server-side operations, while Quill.js enhances content creation, and Bootstrap streamlines front-end development and design. Together, they create a feature-rich and visually appealing blogging platform.

Noteworthy Aspects of the Website:

1. **Bootstrap Integration**

I have integrated bootstrap in the front-end development of my website. It significantly enhances my design and development process, resulting in visually appealing and responsive user interface.

Responsive Design: I have used Bootstrap's responsive design features to ensure that my website adapts seamlessly to different screen sizes and devices, hence offering an optimal user experience for both desktop and mobile users. This is evident in the implementation of the card layout in partials/article-card.ejs (line 2), where articles are displayed in a responsive grid that adjusts based on the user's screen size. The btn-group in author-home.ejs (line 21) also showcases how Bootstrap is used to align buttons neatly on different devices.

Bootstrap Components: The use of Bootstrap components, such as buttons, forms, and cards, contributes to a consistent and polished user interface. For instance, the buttons in author-home.ejs (lines 33-42) and reader-article.ejs are styled using Bootstrap classes, ensuring a visually cohesive design across the entire website.

Bootstrap Grid System: I have also used the grid system in Bootstrap to simplify the layout design and ensure that content is appropriately organized and aligned. This is evident in reader-home.ejs (lines 17-28), where articles are displayed in a three-column grid layout, creating a visually balanced display of published articles.

Utility Classes: I have used Bootstrap's utility classes, such as d-flex, align-items-center, and justify-content-between in various places throughout the code to achieve specific layout and alignment requirements. For instance, in author-home.ejs (lines 19-23), d-flex and align-items-center classes are applied to the header, ensuring that the "Blog Settings" button is vertically centered with the header text.

2. **Code Organization & Readability**

My code organization and readability make my website exceptional. I have adhered to several best practices that makes it easy to maintain and update the website in future.

Comments and Documentation: In my code, comments are thoughtfully placed to provide explanations for complex or crucial logic. For instance, in author.js, comments clarify the purpose of each route, such as retrieving articles, creating new articles, and updating blog settings. These comments act as valuable references for developers who work on the project in the future.

Reusable Partial EJS Templates: The use of partial EJS templates, such as partials/article-card.ejs, demonstrates my code reusability. This file contains the markup and logic to display article cards and i have utilized it in both the author and reader home pages. The practice of reusing components reduces code duplication, simplifies maintenance, and promotes consistency in the user interface.

Minimizing Callback Hell: I have used well-structured and concise callback functions to minimize callback hell, making the code more maintainable and less error-prone. For example, in my author.js, well-organized callback functions handle different operations like rendering templates, updating articles, and managing settings.