Databases, Network and the Web Coursework for Midterm:

Blogging tool

Introduction

Your task is to create a deployable blogging tool which a single user could deploy on their own server and use to maintain a blog. The tool should provide discrete author and reader pages. The author must be able to write articles, save them as drafts and publish them. Readers must be able to browse and read published articles as well as comment on and like them. Please read the base requirements carefully for the full detail.

Technical specification

All submissions **MUST** use the template project provided with the assignment. Express.js must be used as the basis for the server side functionality. SQLite must be the basis for the data tier. Client side pages must be delivered through server-side rendering using Embedded Javascript Templates. As such the following tools are pre-requisite:

Tool	Description	URL
EJS	Tool for Embedded Javascript Templates. Allows for dynamic rendering of HTML.	https://www.npmjs.com/package/ejs
SQLite3	Query your data tier from within your express app	https://www.npmjs.com/package/sqlite3
ExpressJS	A framework for simplifying building a web server	https://expressjs.com/
NodeJs	Javascript engine for building web servers	https://nodejs.org/en/
SQLite	Portable, single file-based SQL database.	https://www.sqlite.org/index.html

You may make use of NPM addons and front end libraries provided that they are justified and don't fundamentally alter the above specification. Some example packages are listed below:

Tool	Description	URL
Tailwind	A popular CSS library. NB. requires some quite tricky build tools	https://tailwindcss.com/
Bootstrap	An older but still popular CSS library which is easier to use straight off	https://getbootstrap.com/
Axios	A helpful library for making requests from both client and server	https://www.npmjs.com/package/axios
Assert	Internal node module. Useful for testing and basic data validation	https://nodejs.org/api/assert.html
Joi	Really helpful for improving that data validation.	https://www.npmjs.com/package/joi
Express-Validator	A more scalable approach to data validation.	https://express-validator.github.io/docs/
Express Sessions	Persistent and secure storage of user details when they are logged in	https://www.npmjs.com/package/express-session
Date-Fns	Date/Time formatting tool	https://date-fns.org/

- All source code must be human readable
- Do not use bundlers such as Webpack
- You can use CSS precompilers such as Tailwind provided the built CSS is provided with the submission
- We will run ```npm install``` followed by ```npm run build-db``` and ```npm run start``` to run your project. Please ensure that your project runs from these commands alone without need for further compilation or installation of dependencies.

Data requirements:

- Pages must populate with user data dynamically retrieved from the server. This can be done through templating (as taught in this course), or through other means (eg. javascript requests and DOM manipulation).
- Features relying on hard-coded client-side user data will be disregarded
- All user data must be persistently stored in the SQLite database

Code Style Requirements:

- Frontend code comprises template files (.ejs) in the views folder and optionally assets and .js files in the public folder
- Backend code comprises an index.js file with routes implemented as middleware in the routes folder.

- Each route should be preceded by a comment describing its purpose, inputs, and outputs
- Routes should be rationally organised making appropriate use of GET, POST methods.
- Each database interaction should be commented describing the purpose, inputs, and outputs.
- Code should be laid out clearly with consistent indenting.
- Functions and variables should have meaningful names, with a consistent naming style
- All variables should be declared with careful attention to scoping

Base Requirements:

Implement these basic features fully to achieve a passing grade.

Author - Home Page:
This is where the Author can create, review and edit articles. The minimum requirements for the page are as follows:

- It should be accessed through a URL which is distinct from the Reader Home Page
- It should display the blog title, subtitle, and author name
- It should have a second heading which makes it clear that this is the author page
- It should have a link which points to the settings page
- It should have a "Create new draft" button
    - When pressed this should create a new draft article and redirect to it's edit page
- It should display a dynamically populated list of published articles
    - The list should display useful information about the articles such as when they were created, published, and last modified, and number of likes
    - For each article the list should display a sharing link which points to the relevant Reader - Article Page
    - For each article there should be a delete button for the article. When pressed this should:
        - Remove the article from the database
        - Reload the page to display the updated information

- It should display a dynamically populated list of draft articles
    - The list should display useful information about the articles such as when they were created, published, and last modified
    - Each article in the list should be accompanied by a link which points to its edit page
    - For each article there should be a publish button. When pressed this should:
        - Update the article's state from draft to published
        - Timestamp the publication date
        - Reload the page to display the updated information
    - For each article there should be a delete button.

Author - Settings Page:

This is where the Author can change the blog title , subtitle and author name. The minimum requirements for the page are as follows:

- A title which indicates that this is the settings page
- A form with text inputs for Blog title, subtitle, and author name.
    - The form should be dynamically populated with the current settings for the page
    - The form should have a submit button which updates the settings with the new values and redirects to the Author - Home Page.
    - Form validation should be used to ensure that all fields have been completed ahead of submission.
- A back button which points to Author - Home Page.

Author - Edit Article Page:
This is where the author writes, amends and publishes individual articles. The minimum requirements for the page are as follows:

- Data about the article (Eg. created, last modified)
- A form containing the following input fields and controls:
    - Article title
    - Article subtitle
    - Article text
    - A submit changes button
- The form should be populated with the current article data
- When changes are submitted the article's last modified date should be changed
- A back button which points to Author - Home Page.

Reader - Home Page:
This is the front page which readers use to access the blog site. The minimum requirements for the page are as follows:

- It should be accessed through a URL which is distinct from the Author - Home Page
- It should display the blog title, subtitle, and author name
- A list of published articles
    - The article title and publication date should be visible for each item
    - Articles should be ordered by publication date with the latest publication appearing at the top
    - Clicking on an item in the list should take the user to the Reader - article page for that particular article

Reader - Article Page:
This is where the reader can read and interact with individual articles. The minimum requirements for the page are as follows:

- It should display a single article determined by the url
- It should display information about the article including, article title and subtitle, publication date, number of likes

- It should display the text of the article
- There should be a like button to react to the article
- There should be form for adding a comment to the article containing
    - a text input for the comment.
    - A submit comment button
- When the user submits a comment the page should be reloaded so that their comment appears in the list.
- There should be a list of previous reader comments which should be ordered by the date when they were written
- There should be a back button which redirects the user to the Reader - Home Page.

Extension:

You should also enhance your project by completing an extension from the following options. You are free to complete as many extensions as you want but we will ONLY consider the first extension which is specified in your commentary.

Front end styling and GUI:

- Use CSS styling to create a modern professional look and feel to your application
- You might do this using utility frameworks such as Bootstrap or Tailwind, or write the CSS from scratch

Add extra functionality:

- Add a range of extra functionality to the app on both author and reader sides. For example:
    - Log article stats such as number of reader visits and read time.
    - Add a moderation feature for comments.
    - Add further reactions for articles
    - Allow authors to add tags to articles and implement a filter by tag for readers

Password access for author pages and routes:

- Secure the author pages with password authentication and write middleware to prevent any unauthorised access to author endpoints
- You'll need to create an author login page which authenticates the author against a naively stored server-side password (eg. in an environment variable)
- You'll also need to use a package such as express-session to create secure sessions

Add a rich text editor:

- Research and add an open source rich text editor such as https://summernote.org/ to allow authors greater control over their content.
- You'll need to make sure that the rich text also gets rendered to the Reader Page as well.

Best practices:

Move your code base towards production level by following some recommended best practices below:

- Implement applicable security best practices from here https://expressjs.com/en/advanced/best-practice-security.html
- Implement code-based performance best practices from here https://expressjs.com/en/advanced/best-practice-performance.html
- Implement data sanitisation using a library such as https://joi.dev/ or https://express-validator.github.io
- Be sure to list everything you've done in your write up.

Deliverables

- The source code of your web application in zip format
    - Make sure node_modules is removed from your project
    - Make sure package.json is included and that all additional packages are listed in your package.json file
        - HINT: make sure you used ```npm install –save package_name``` when you installed them
    - Do not include your SQLite database.db file. We will build this by running ```npm build-database```
    - Update the README.md to include any settings that should be adjusted in configuration files and concise instructions for how to access the reader and author pages once the app is running

- Commentary in PDF format
    - A high level schematic diagram for your website demonstrating all three tiers of your architecture, the end points that connect client to server.
        - Use a free tool such as figma.com to produce this
    - Extension description
        - Specify which extension you implemented.
        - Discuss what you did to implement it.
        - Highlight aspects which you particularly want us to pay attention
        - Use screenshots of code or refer to filenames and line numbers where appropriate

- A video screen cast of your web application
    - This should not be longer than 2.5 minutes.
    - It should demonstrate all of the functionality
    - We recommend that you capture the video in mp4 format using software such as OBS
    - This is not a graded requirement rather it is a backup in case we are unable to install and run your code. We strongly recommend submitting a video to guarantee that we can grade your code.