CPSC 349 - Web Front-End Engineering

Project 3, Fall 2022

Last updated Wednesday November 16, 15:35 PST

In this project you will design and implement a client-side application that works entirely in the browser, requiring only HTML, CSS, and JavaScript.

The following are the learning goals for this project:

1. Working with HTML, CSS, and JavaScript
2. Configuring NPM scripts
3. Writing code to implement a client-side application in JavaScript
4. Implementing event listeners in JavaScript to handle user input
5. Working with the Document Object Model to alter an HTML page
6. Using the Web Storage API to persist data between page visits
7. Publishing a static site to GitHub Pages

Project Teams

This project must be completed in a team of three or four students. The instructor will assign teams for this project in Canvas.

See the following sections of the Canvas documentation for instructions on group submission:

- How do I submit an assignment on behalf of a group?

Platform

Per the Syllabus, this project is written for Tuffix 2020. Instructions are provided only for that platform. While it may be possible for you to run portions of this project on other platforms, debugging or troubleshooting any issues you may encounter while doing so are entirely your responsibility.

Libraries, Tools, and Code

This project requires only HTML, CSS, and vanilla JavaScript, During development you will also need to configure the following components:

1. An NPM script to start Browsersync for testing

2. An NPM script for testing, including HTML validation and JavaScript Standard Style checking.
3. An NPM script to run the Tailwind CLI.

   Optionally, you may also configure a Tailwind component framework such as daisyUI or Tailwind Elements.

Code from the assigned reading and examples provided by the instructor may be reused. All other code must be your own original work or the original work of other members of your team. In particular, this means that you **may not** use third-party JavaScript libraries. All code must use the standard Client-Side Web APIs built into your browser.

Choosing a game to implement

In this project you will implement a simple turn-based game, either for a single player or for multiple players taking turns using the keyboard and mouse.

Note: For game nerds, this means that the game will not be real-time and will not require a back-end.

You are free to choose any simple game. For examples of the kinds of games you might choose, see the following pages for inspiration:

- Classic Games
- BASIC Computer Games (and Vintage Basic's Basic Computer Games)
- Early mainframe games
- Usborne 1980s Computer Books

Implementing the game

The game implementations listed above have either primitive interfaces (i.e. the equivalent of C++ `cout` and `cin`), or require a back-end server. In contrast, your game should have a modern web interface in HTML, CSS, and JavaScript, and the game logic should run solely inside the browser.

Note: that does not mean that there will not be a web server involved, just that the server does not generate HTML dynamically or provide a back-end API (i.e. it acts as a static web server).

Requirements

Your application should consist of a single HTML page along with supporting CSS and JavaScript files. User input should be handled through form controls and events.

In addition, your application must meet the following requirements:

- The HTML file must be valid.
- JavaScript code must use JavaScript Standard Style.
- You may not use inline styles or style attributes. (Tailwind CSS classes are permissible.)
- You may not use inline scripts or inline event handlers. All JavaScript must be loaded from a module.
- Game state must be saved using Window.localStorage, allowing the user to close the browser and resume the game later.
- You may not use third-party JavaScript libraries or fetch data from a server. All other built-in Client-side web APIs are permissible.

  Note: this includes the Canvas API. You are welcome to add graphics to your program, but don't get carried away.

Note: If you need to generate large amounts of HTML, consider combining Element.innerHTML and template literals. While there are security considerations when using `innerHTML` with data retrieved from a server, it is safe to use in applications that execute only client-side,

Note: Sanitizing HTML generated from data retrieved from a server is one of the reasons to use a templating language like Liquid or Handlebars.

Publishing to GitHub Pages

When finished, your application should be published to and playable via GitHub Pages. The site may be published either as a User site (`http://`*`username`*`.github.io/`) or a Project site (`http://`*`username`*`.github.io/repository`).

Note: Do **not** take down your site from Project 1 to do this. Either publish a different repository or for a different user, or add the files for this application to a separate subdirectory and submit that URL instead.

Submission

Part 1 - Submit the published URL

Submit the URL for your published application through Canvas by the due date. Only one submission is required for a team.

The Canvas submission deadline includes a grace period. Canvas will mark submissions after the first submission deadline as late, but your grade will not be penalized. If you miss the second deadline, you will not be able to submit and will not receive credit for the project.

Reminder: do not attempt to submit projects via email. Projects must be submitted via Canvas, and instructors cannot submit projects on students' behalf. If you miss a submission deadline, contact the instructor as soon as possible. If the late work is accepted, the assignment will be re-opened for submission via Canvas.

Part 2 - Project feedback

After the project is due, the next week's online reflection will invite you to play a game or two and post thoughtful and constructive feedback on the work of other teams.

Part 3 - Team member evaluation

A Canvas Quiz due the week after the project will ask you to provide candid feedback on the relative contributions of each member of the team.

Grading

The project will be evaluated on the following five-point scale, inspired by the general rubric used by Professor Michael Ekstrand at Boise State University:

Exemplary (5 points)

Project is a success. All requirements met. Quality of the work is high.

Basically Correct (4 points)

Project is an overall success, but some requirements are not met completely, or the quality of the work is inconsistent.

Solid Start (3 points)

The project is mostly finished, but some requirements are missing or the quality of the work does not yet meet professional standards.

Serious Issues (2 points)

The project has fundamental issues in its implementation or quality.

Did Something (1 point)

The project was started but has not completed enough to fairly assess its quality, or is on entirely the wrong track.

Did Nothing (0 points)

Project was not submitted, contained work belonging to someone other than the members of the team, or was of such low quality that there is nothing to assess.