GoodMark

Simple markdown preview and editing web-tool.

Project Summary

This project aims to provide a simple, lightweight Markdown previewer. Users can change and preview Markdown syntax in real time, assisting beginners and experts alike. The program simplifies Markdown to make it accessible.

Project Description

GoodMark is a web-application that converts plain-text Markdown into rendered HTML in real time to facilitate Markdown writing. It will be useful for both beginners and experts who seek a lightweight Markdown tool that renders in real time without a complete editor.

Main features:

1. A live preview that updates as Markdown is typed.
2. Code snippet syntax highlighting.
3. A simple, clean user interface, see User Interface Design.
4. Dark and light modes for your convenience.

Design

This project's design and construction involve multiple stages. I began with User Interface design to visualise my project. After designing the Software's architecture, I planned its implementation.

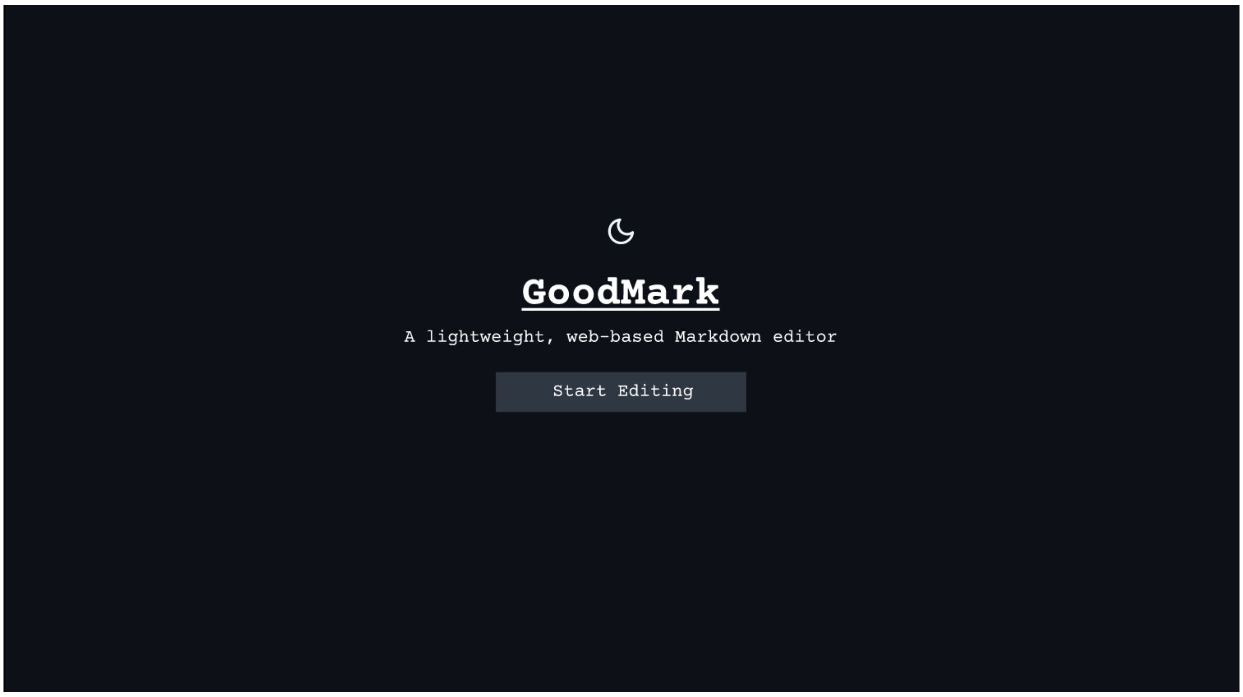
User Interface

The user interface design was created in Figma. The design prototype is here.

Product Design

GoodMark Welcome Screen

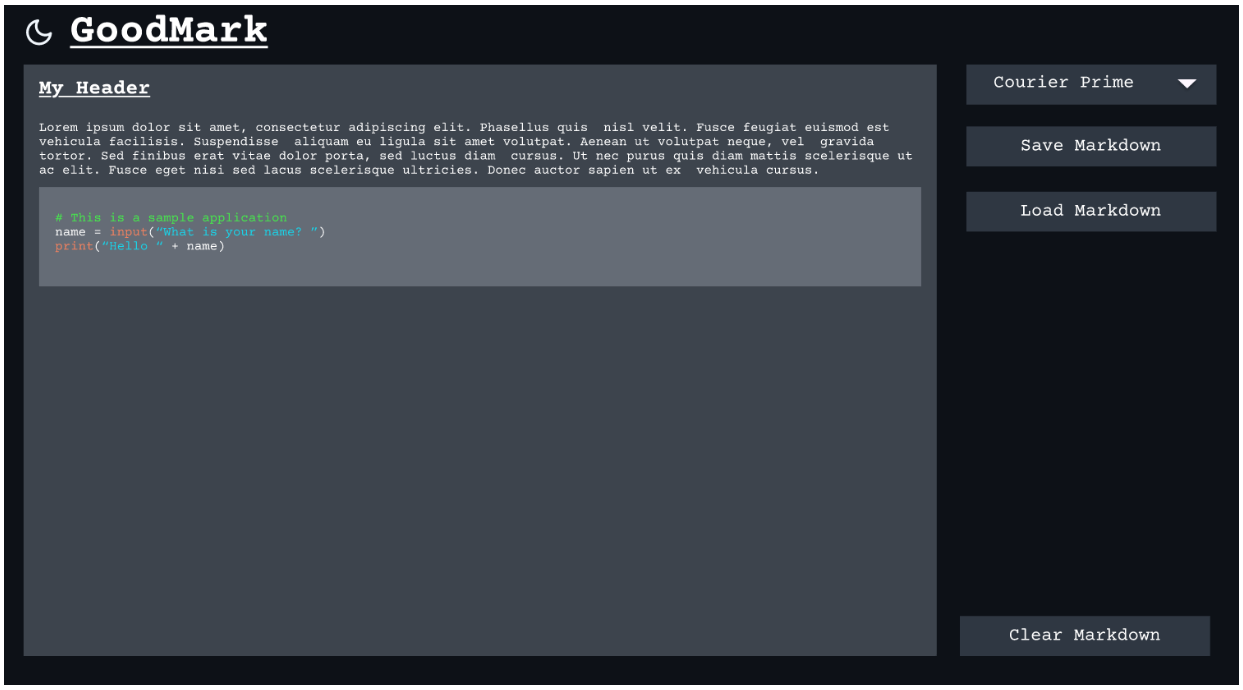
Websites should start with product splash screens. Like many apps, they produce software. Video games usually contain pre-game menus. This idea was appealing because it didn't overwhelm new users with an editing screen and gave them a choice. Using a browser cookie, I'll track if the user visited the webpage and selected "Start Editing". Regular users cannot see the splash screen again without removing their cookies, preventing them from repeating the procedure while showing newer users. This panel lets users choose bright or dark mode as remembered in a browser cookie. Dark by default.



Editor Screen

When the user hits "Start Editing" or has visited the website and stored the cookie, the Markdown editing screen opens. Never-changing sample text is shown here. The last text supplied will fill the textbox and be saved in a cookie. Headline, regular text, and Markdown code block will be in the sample text. They'll also have a combo-box to select a font from a pre-defined list, a "Save Markdown" button to save their Markdown to their machine, and a "Load Markdown" button to load any Markdown file into the text field The bottom right "Clear Markdown" button removes Markdown, but data loss is avoided. This screen's top left icon lets clients choose light or dark mode like the welcome screen.

Dark Mode

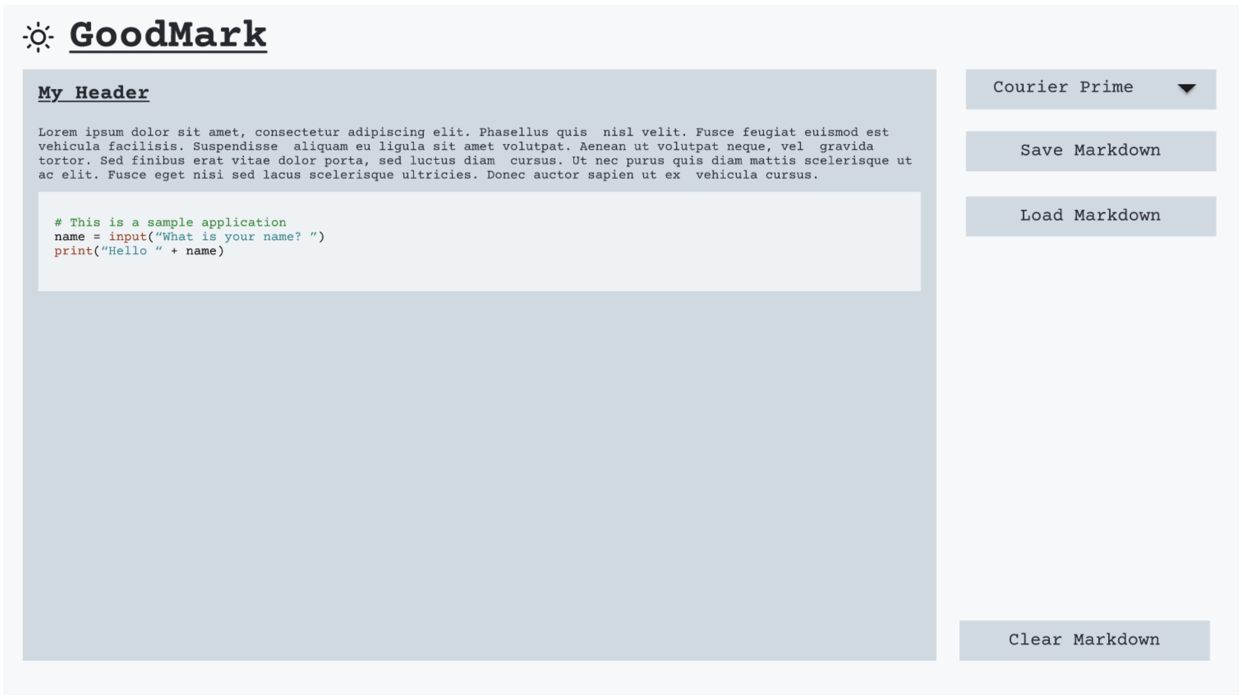


My Header

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus quis nisl velit. Fusce feugiat euismod est vehicula facilisis. Suspendisse aliquam eu ligula sit amet volutpat. Aenean ut volutpat neque, vel gravida tortor. Sed finibus erat vitae dolor porta, sed luctus diam cursus. Ut nec purus quis diam mattis scelerisque ut ac elit. Fusce eget nisi sed lacus scelerisque ultricies. Donec auctor sapien ut ex vehicula cursus.

```
# This is a sample application
name = input("What is your name? ")
print("Hello " + name)
```

Light Mode



My Header

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus quis nisl velit. Fusce feugiat euismod est vehicula facilisis. Suspendisse aliquam eu ligula sit amet volutpat. Aenean ut volutpat neque, vel gravida tortor. Sed finibus erat vitae dolor porta, sed luctus diam cursus. Ut nec purus quis diam mattis scelerisque ut ac elit. Fusce eget nisi sed lacus scelerisque ultricies. Donec auctor sapien ut ex vehicula cursus.

```
# This is a sample application
name = input("What is your name? ")
print("Hello " + name)
```

Software Architecture

The React frontend will provide a responsive user interface. A hosted backend will render Markdown-to-HTML, allowing us to use server-side processing.

Backend Architecture

The backend will use Express.js and have two endpoints:

- GET `/render` converts Markdown to HTML.
- GET `/fonts` returns a list of available fonts.

Implementation Approach

Use Test-Driven Development to ensure program resilience. We will develop integration unit tests to test each endpoint and function and mock externals to test our code to ensure backend functionality. GitHub Actions will establish a CI/CD pipeline that builds, tests, and lints code before merging a PR. Post-merge pipeline pushes PR changes to live software.

Planning

For a well-rounded project, I want to use Agile sprints to provide the solution in stages. The following tools will help me design the solution.

Personas and User Stories

User stories will be my initial development tool. First, in high-level stories that explain the project's core requirements, then in user stories for each feature. To help with this, I've established three personas with their own aims and difficulties, as well as empathy maps of each. When developing a feature, I'll designate whose persona it's for. The personas are available here.

MoSCoW Prioritisation

Must Have	Should Have	Could Have	Wont Have
Live Markdown	Dark and Light	Export Options	Collaboration

Must Have	Should Have	Could Have	Wont Have
Preview	Mode		Features
Lightweight Design	Syntax Highlighting	Offline Mode	
Web Accessibility		Templates	
Accessible Design		Responsible Web Design	

Project Management

All project management approach will be done using GitHub tools, which is great because it keeps Source Code Management and Project Management together. This project will be managed utilising the following items, how they will be used, and why.

Issues, Epics and Stories

Issues on GitHub will be the main way to assign and prioritise bugs, features, infrastructure modifications, and other activities. The repositories' issue page allows you to describe any work.

Five issues types will be used in the project:

1. Feature - user story for a new feature.
2. Epics - contain feature and infrastructure issues.
3. Bugs - report bugs.
4. Infrastructure - infrastructure modifications.
5. Other - anything else.

Sizing

Issues will be sized, using a T-shirt sizing model, Small, Medium, Large, Extra Large. This should be based on the amount of work expected.

Prioritisation

Issues will also be prioritised, Low, Medium, High, and Urgent. This should be based on their importance to the end users.

Sprints

Development will be done in Sprints, each Sprint will be 1 day long and contain at least 1 Epic. Sprint 0 will be longer than 1 day.

GitHub Projects

We'll manage and view work on GitHub Projects' Kanban board. This repository's GitHub Project: https://github.com/users/Dinoosawruss/projects/1. The board will have backlog, in progress, blocked, reviewed, and done columns. Moving work across the board to the relevant column should show the current condition of the GitHub Project. An Epics tab on the project board lists epics, their status, and sprint. All project pull requests are listed on a PR tab.

Risks

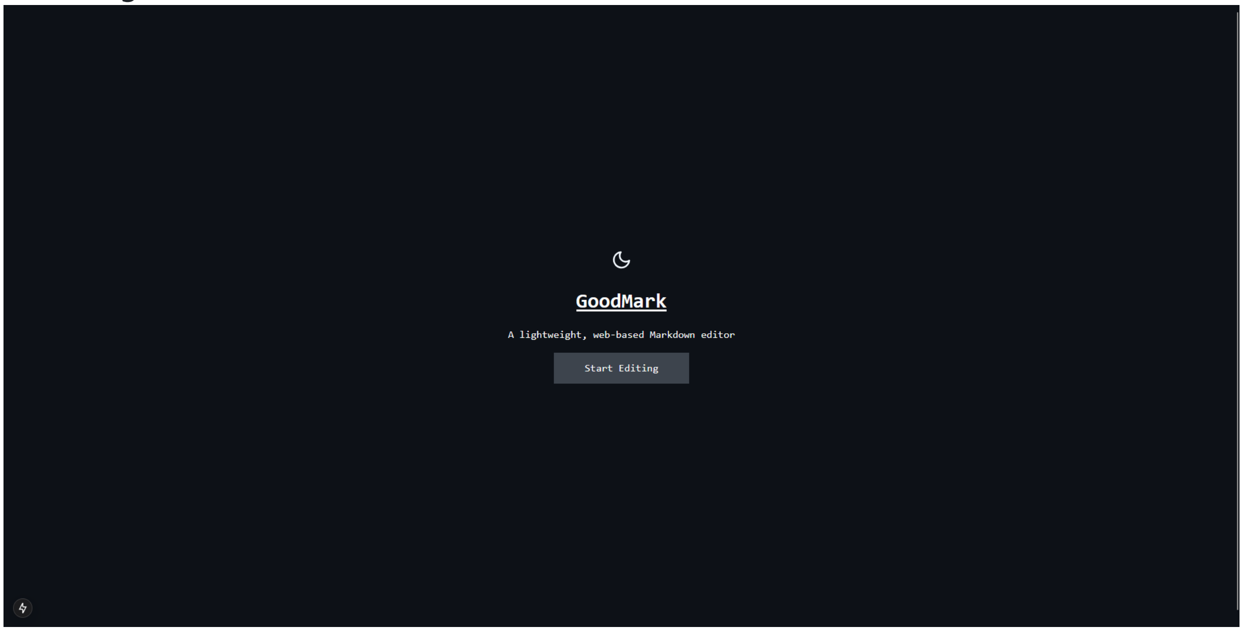
- Time Limitation - each Sprint is only one day
- Performance - low performance will create a bad user experience

Documentation

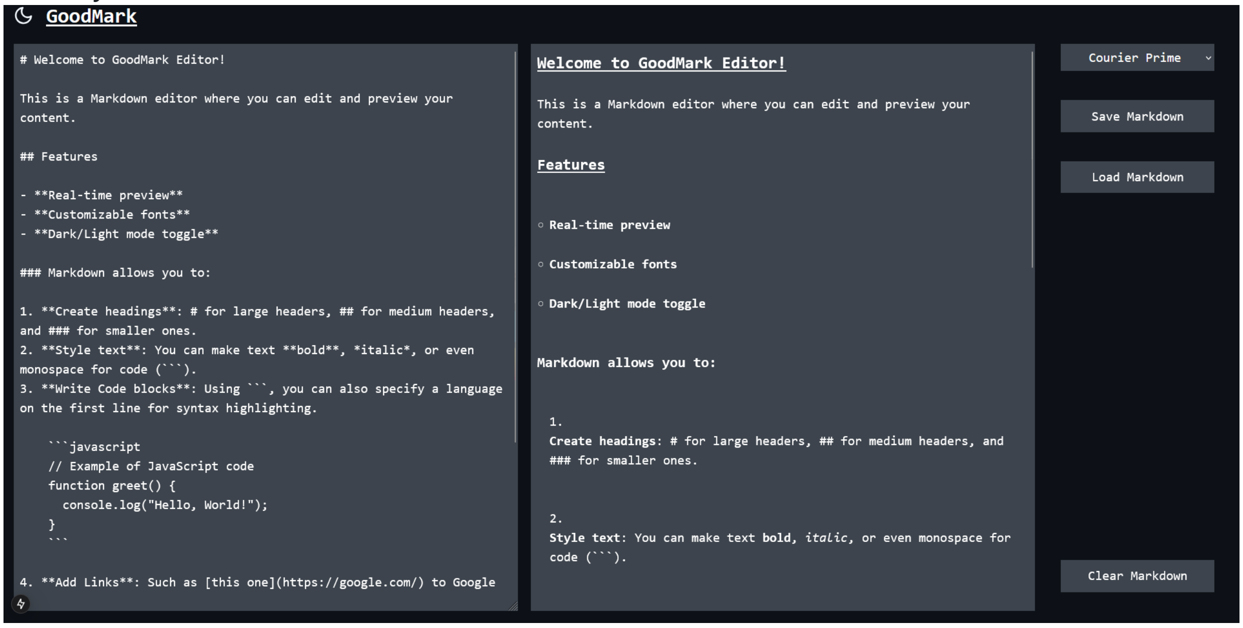
User

Render hosts GoodMark on the web (https://software-engineering-summative-1-frontend.onrender.com/). If you load the URL please allow at least 1 minute for it to work and render your initial Markdown.

On your first visit, the Welcome Screen lets you choose light/dark mode and access the editor. Your OS preference will determine your initial preference. You'll never see this screen again.



The Editor Screen allows real-time Markdown editing and preview. Click "Load Markdown" to load Markdown from your computer and "Save Markdown" to save the Markdown. Switch between light and dark modes using the top-left icon. You can also select your font.



~~~
# Welcome to GoodMark Editor!

This is a Markdown editor where you can edit and preview your
content.

## Features

- **Real-time preview**
- **Customizable fonts**
- **Dark/Light mode toggle**

### Markdown allows you to:

1. **Create headings**: # for large headers, ## for medium headers,
and ### for smaller ones.
2. **Style text**: You can make text **bold**, *italic*, or even
monospace for code (```).
3. **Write Code blocks**: Using ```, you can also specify a language
on the first line for syntax highlighting.

    ```javascript
    // Example of JavaScript code
    function greet() {
      console.log("Hello, World!");
    }
    ```

4. **Add Links**: Such as [this one](https://google.com/) to Google
~~~

Welcome to GoodMark Editor!

This is a Markdown editor where you can edit and preview your content.

Features

- Real-time preview
- Customizable fonts
- Dark/Light mode toggle

Markdown allows you to:

1. Create headings: # for large headers, ## for medium headers, and ### for smaller ones.
2. Style text: You can make text bold, *italic*, or even monospace for code (```).

Technical

The Markdown editor is a single repository with `/frontend` and `/backend` sections. The frontend uses Next.js and has two main pages: index and `/editor`. The backend uses Express.js and has two endpoints: `POST /render` and `GET /fonts`. All Markdown rendering is done server-side, while the frontend receives HTML.

Setup Instructions

Setting up GoodMark locally requires:

1. Install Node.js.
2. Install git (https://git-scm.com/).
3. Clone the git repository with `git@github.com:Dinoosawruss/software-engineering-summative-1.git`.
4. In the `/backend` directory, run `npm install`.
5. Launch the backend server with `npx nodemon index.js`.
6. In the `/frontend` directory, run `npm install && npm run build`.
7. Create a `.env.local` file with `NEXT_PUBLIC_BACKEND_URL=http://localhost:5000`.
8. Use `npm run dev` to start the frontend server.
9. The frontend and backend should now be available at http://localhost:3000 and http://localhost:5000.

Testing

A complete unit test suite exists. Test-driven development should be used for all development to ensure code coverage and suite strength. Use `npm test` to run tests on both the frontend and backend. Both use jest for testing, with the frontend utilising `swc/jest`.

Narrative

Sprint 0

Sprint 0 covered project setup and Software Engineering "admin". This included issue templates, branch protection rules, README writing, and build, dependency test, and deployment CI/CD pipelines. This sprint was the longest due to pre-work, but it lay the groundwork for future sprints. Figma UI designs here and project Epic, Feature, and Infrastructure issues were also created. I allocated all sprints.

Sprint 1

CI/CD pipeline issues in Sprint 0 slowed Sprint 1, but after they were rectified, I could start the project and add basic features. To create the backend, I used Express.js and constructed the `POST /render` endpoint using `marked`, a tool that converts Markdown to HTML TDD helped me construct the endpoint. I first tested `# Hello` to confirm it returns `<h1>Hello</h1>`, then implemented the solution. I then set up the Next.js React frontend and deployment since Sprint 0 had simply set up backend. Finally, I added Markdown rendering and calls to the `POST /render` backend. Sprint 1's most crucial assignment yielded a satisfying result. I added and tested objects using TDD like the backend. I added Save, Load, and Clear Buttons. Last Sprint 1 addition: Prism.js syntax highlighting. Due to its heavy initialisation and implementation effort, this sprint overlapped Sprint 2.

Sprint 2

Sprint 1 ended, starting Sprint 2. Syntax highlighting and Save, Load, and Clear buttons were included. I added font selection after this carry-over. To control font availability, the site established a backend endpoint `GET /fonts` and modified the front end to request, load, and modify fonts. Trying to use "Courier Prime" monospace took longer than intended, but a missing comma prevented the website from rendering. I created a light and dark mode setting depending on your system choice and let you switch between them after fixing the issue. Accessibility features including ARIA components, keyboard navigation, and shortcuts `Ctrl + S` and `Ctrl + O` were added after significant effort. I checked Chrome and Windows Narrator accessibility. Finally, I made the site responsive for portrait and landscape views to stack the editor and markdown preview on portrait devices.

Sprint 3

Sprint work may begin. This Sprint focused on Welcome Screen, quality of life, and performance. I started with the Welcome Screen, which was straightforward to build because I could reuse many Editor screen features but required some advanced file structure and testing suite tweaks. Fixing that allowed me to develop the Welcome Screen and automated forwarding after the first visit. I then added quality-of-life features like preserving dark/light-mode, last Markdown, and last font choices. Implementation was simple using `localStorage` instead of cookies, offering a simple key-value API. In the Bugs bust, I fixed some important testing suite issues, which was harder because I had built a massive testing suite and any issues would compound and create many failures. Next was backend scalability performance. The deployed instance baseline was set via scalability testing. The deployed instance has limited resources due to its free status, so a baseline response time was needed. I improved performance via caching, error handling, and async API calls. Sanitising the HTML answer prevented Cross Site Scripting. After these performance modifications, I ran a scalability test and found the deployment could handle 90 requests per second without HTTP losses. Performance was good, and MVP was given.

Bugs

TDD helped me catch many regressions and errors during development, minimising the number of issue tickets I had to file and fix outside the sprint. However, some emerged. Early in development, the frontend used `http://localhost:5000` for the backend, even on deployed instances. I implemented `dotenv` to pull the URL from a `.env` file, avoiding users from deploying the backend on their own machine. This worked well and required little deployment. No tickets were needed because I only detected problems during the development cycle, however during Sprint 3, I uncovered bugs and filed issues. I fixed these issues in one huge Pull Request. App bugs include the dark and light mode selector not saving the last state, the markdown preview centring after forwarding from the welcome page, malfunctioning keyboard controls, the Clear Markdown button not clearing the preview, and the default Markdown being added when the local storage key is empty. It was excellent to resolve all these issues. I wondered how these issues had gone through, and then I saw they were in locations with few unit tests and no test cases, so I added test cases to prevent regressions. In portrait mode, the font is hidden. Several CSS tweaks failed to fix this issue, so I left it open.

Evaluation

I believe this project succeeded. It helped me learn React, Express.js, and Test Driven Development. I like my app's testing framework, usability, and User Stories. I've tracked my work and updated the board using GitHub's project management tools. I overcame bugs, testing issues, and learning Next.js and Express.js to construct the tool. I think my project implementation was good, but I could have chopped Sprint 1 and added a Sprint to decrease time pressure. For additional time to finish, I should have made the Sprints 2 or 3 days longer. Other than these few issues, the project has been successful, and an open source contributor might easily review this README and use the tools to contribute. I think my personas appreciate the tool and that I met their user stories.