Research and Development Document: The Resume Builder Web App

by Gui Prá

Introduction

The development of the Resume Builder Web App represents a unique integration of personal projects and innovative solutions in web development. This document outlines the key stages and decisions in the research and development process of the project.

Development Overview

Utilizing Personal Projects

- **Webfoundry**: The initial phase of UI development leveraged Webfoundry, a personal website builder project. By constructing the UI in Webfoundry and then transferring the HTML to the app source code incrementally, significant progress in UI development was achieved rapidly. A demonstration of Webfoundry's capabilities can be found on my portfolio: https://cv.guiprav.com/portfolio.html.
- **Dominant Over Traditional Frameworks**: Instead of using mainstream frameworks like React or VueJS, I opted to use Dominant, another personal project. Dominant, a compact and efficient React alternative, comprises less than 1000 lines of well-commented code. This choice not only showcased the capabilities of Dominant but also ensured a lightweight and streamlined application development. More about Dominant can be found here: https://github.com/guiprav/dominant.
- **Windy - Tailwind Integration**: In terms of styling, I utilized Windy, my personal implementation of Tailwind CSS. Windy is a JavaScript-only solution that employs DOM MutationObservers, allowing Tailwind class names to be used seamlessly. Windy dynamically generates CSS rules on-the-fly as it detects class names, providing a powerful and unobtrusive approach to styling. Windy's source-code can be found here: https://github.com/guiprav2/windy/blob/master/index.js.

Research: PDF Export Functionality

- The key research component involved the PDF exporting functionality. Initial trials with jsPDF were unsatisfactory due to issues with lack of text metadata, layout overflow, and poor image compression. Consequently, the decision was made to utilize the browser's built-in print/PDF export dialog. Unfortunately this approach requires a little careful adjustment of settings by the end-user, such as removing margins and enabling background colors, to achieve the best results. A JavaScript solution might have been better, but I couldn't find one.

Development Process

Phase 1: Sidebar Development

- The first development stage focused on creating a functional sidebar. This sidebar allows users to manage lists of templates and candidates, including features to create, delete, and rename these entities. These actions were implemented using HTML5 modals, providing a user-friendly interface for input and confirmation prompts.

Phase 2: Template Text Editor

- Given the project's time constraints, a simple text editor was chosen for template editing. The Ace text editor from Cloud9 was selected for its efficiency and ease of use. This choice enabled a straightforward and effective means to edit templates, which are structured using HTML+Handlebars for simplicity and effectiveness.

Phase 3: Candidate Editor Panels

- The development of the candidate editor panels followed, focusing on a user-friendly form design. The form, divided into sections Basic Details, Experiences, Education, and Skills, was positioned alongside a resume preview panel. This layout facilitated immediate visual feedback as users inputted their information.

Data Management and Automatic Saving

- Data storage was handled using localStorage, with keys prefixed with `mrb:` to prevent conflicts in shared domains such as GitHub Pages. The application features automatic saving and live preview updates, debounced to 200ms to reduce unnecessary processing during data input.

Refactoring and Optimization

- Initially, localStorage management was directly integrated within the UI component files. However, upon establishing full functionality, I undertook a series of refactors to enhance code organization and efficiency.
- The first refactor involved abstracting the localStorage management using the repository pattern. This separation streamlined the data handling aspect of the application, making the code more maintainable and scalable.
- Subsequently, I shifted non-UI logic to a message-based "app controller". This controller manages the application state (e.g., loaded template IDs) and handles state modifications through an `appCtrl.post` method. This method is responsible for dispatching actions and parameters to appropriate handler methods, further decoupling logic from the UI components.

Conclusion

The development of the Resume Builder Web App was characterized by a blend of personal ingenuity and pragmatic decision-making. Through the strategic use of personal projects like

Webfoundry, Dominant, and Windy, and some thoughtful research, the application not only serves its intended purpose efficiently but also stands as a testament to innovative development practices in the realm of web applications.