CSC301 Fall 2020 A1 Group 81 - Web App Report

1. Frontend Choice

1.1 Vanilla HTML, CSS, JavaScript

Using the 'vanilla' web tools would have been the lightest weight tools in terms of size and performance. From a developer standpoint, they are also the easiest to work with; both of our members are comfortable with web programming basics. Unfortunately, a self-checkout style app has significant interactivity and page/DOM manipulation, areas in which the 'vanilla' tools struggle, and where frameworks/libraries excel.

1.2 Django

Like the previous tools, Django requires relatively less learning, as both members of our team are very familiar with Python. Its loosely coupled architecture also makes it very good for rapid development and scalability. Unfortunately, these advantages are negated by the small size of the project (as a simple checkout calculator) and Django struggles with single page applications, which is important for the product being made

1.3 Angular

With the importance of the app being a single page app, React and Angular were the main contenders for our front-end. Both are modern and rising in popularity, relatively small with component-based architectures. The reusability of these components in mobile applications was also a big bonus, given the required support of mobile platforms. Unfortunately, Angular's use of Typescript (a language neither member was familiar with) and overall lack of Angular experience on our team meant React was ultimately chosen.

1.4 React - Final Choice

React's use of JSX and Node, our eventual backend choice, and Poplar's experience with the framework made it our final choice over Angular. While its unidirectional data binding caused a few issues, this disadvantage was well outweighed by the lack of need to learn a new framework, potential to reuse code with React Native (although ultimately not used) and kinder learning curve.

2. Backend Choice

2.1 Ruby on Rails

Ruby on Rails was once the most popular web application backend, and its age has resulted in healthy community support and solid front-end compatibility. Unfortunately, its runtime is slow compared to newer alternatives and, as RoR continues to decline in popularity, it made little sense to learn Ruby for this backend.

2.2 Django/Flask

Python backends through frameworks such as Django and Flask have the aforementioned advantage of being built on Python, a language we were very comfortable with. It has a plethora of pre-included features such as user authentication and the admin panel. None of these were required by the project, however, and since our frontend choice was React due to the single-page goal, Python was not chosen.

2.3 PHP and its Libraries

PHP and its libraries (Symfony, Laravel, Zend) form the most popular scripting tools for web projects. Being over 3 decades old, it also has a significant number of different libraries and tools to speed up application development. However, not only is PHP completely foreign to both members of our team, this robust feature set is not necessary for a small scale project such as our checkout calculator.

2.4 Node.js - Final Choice

Thus, Node.js/Javascript was our final choice for our backend technology. Its disadvantages, such as unstable API and asynchronous programming model, are hardly impactful in a small scale project such as our checkout calculator. Its natural integration with our Front-end choice of React (keeping our tech stack simple), as well as both member's experience with Javascript made this the obvious choice.

3. CI/CD Choice

3.1 Jenkins

Jenkins has very powerful automated build processes and great process control. However, its high learning curve and nature as being non-cloud based (requiring infrastructure to be maintained by oneself) meant it was far more powerful than needed, at the cost of complexity.

3.2 Travis

Travis is arguably the most popular CI/CD solution deployed today, and its maturity means many features such as solid integration with GitHub. However, its maturity and popularity has the drawback of making change difficult; its poor design and lack of innovative features made it a questionable choice for our group members who were completely new to CI/CD.

3.3 CircleCI

CircleCI has quick integration with GitHub, and has strong YAML/text based configurations much like Github Actions. However, the largest issue was the highly limited nature of its free plan.

3.4 GitHub Actions - Final Choice

Naturally, GitHub Actions has good integration with GitHub and required little addition to our existing tech stack (just adding a .github directory and YAML files that were available preconfigured online). This ease of use and much more generous learning curve compared to the arguably more powerful and configurable alternatives discussed above.

4. Database Choice

4.1 Relational: PostGreSQL

For full scale databases, the two main types are Relational and NoSQL databases. The most popular choice for Relational databases is PostGreSQL. Relational databases are ideal for logic-related data where integrity is critical, with strict structures that can be identified upfront. However, they are outclassed by NoSQL when unrelated, rapidly evolving data is being used.

4.2 NoSQL: MongoDB

In the NoSQL world, MongoDB is the most popular solution. As mentioned above, it excels in rapidly changing data without requirements for relationships, but data integrity and structure is lacking compared to its relational counterpart. For our app, however, the only data which is stored is product information. Thus, neither of the two main database types were necessary for such a simple product.

4.3 JSON File

Given that all the required data could be stored in a single table, and our usage of React/Node.js, a simple JSON file would be sufficient to store product information (id, name, price). Its simplicity makes it far less complicated, simplifying our tech stack. However, its simplicity also makes it quite limited in functionality; it could not be easily shared across platforms.

4.4 Firebase - Final Choice

With this in mind, Firebase was our final choice for our database. Firebase has the simplicity of a JSON file (being a JSON file itself) but with the added security provided by Firebase and, most importantly, the ability to share a single database between our Web App and Android deployments. Thus, a company could feasibly use both products and only have to edit their database once.

Word Count: 945