CSC301 Mobile App Report

Choosing the frontend:

For the frontend of the mobile app, I chose between Vue, React, and Angular, ultimately deciding on using Vue native. Vue was developed by an ex-Google employee, whereas React and Angular were developed by Facebook and Google respectively. As a result of lacking a large company to back it, Vue is mainly driven by the open-source community. Going by statistics from Github however such as the number of watchers, forks, and contributors, all three show significant development activity, and so this was not a major concern.

One of the biggest considerations for choosing Vue native over the other two was its easiness to pick up. Compared to the other two, Angular has a very steep learning curve, requiring a strong understanding of associated concepts like Typescript and MVC. React uses JSX, a syntax extension of Javascript, and is heavily dependent on third-party components, and naturally, on a wide-reaching understanding of them. Vue on the other hand, separates concerns in a very straightforward manner, between HTML, JavaScript, and CSS, all of which I had prior knowledge in. This allowed for a high degree of customizability and flexibility, letting me easily build what I needed from the ground up rather than scour through libraries to find the right components.

Choosing the backend:

The backend for the mobile app amounted to a decision between Python (flask), NodeJS, and Go, with Python ultimately winning out. Introduced by Google in 2009, Go is extremely powerful in terms of raw performance and computation, handles concurrency well, and is superior to NodeJS and Python in terms of scalability. Merits that may have had more weight in an assignment that wasn't a checkout calculator. Dwarfed in popularity by both Python and NodeJS, Go is comparatively lacking in online resources to learn from. It is a relatively new language with specific concepts, processes, rules, and interfaces to research.

NodeJS is javascript based and so is easy to pick up for someone with prior knowledge in Javascript, but also introduces some advanced topics like event-driven architecture. Nevertheless, it is easy to pick up. Attached to the immense popularity of Javascript is a wealth of resources and support for learning to proficiently use NodeJS.

Ultimately, the decision came down to personal preference. As someone with prior experience in Python, Flask was simply a very convenient option. It possesses easy

unit-testing with Python, its basic foundation API was immensely useful, and it was great for handling HTTP requests, with the mobile app reading JSON data from the server deployed on Heroku to check which items exist to be added. There is also Flask-RESTful, an extension for Flask that adds support for quickly building REST APIs. On top of Flask's high flexibility and ease of deployment, it ended up being an excellent choice for the mobile app's backend.

Choosing the database:

For the database, the main three options I considered were Oracle, MySQL, and PostgreSQL. Postgres is rich with features for handling complex queries and databases. MySQL on the other hand is relatively simpler (lighter in features) and easier to set up and manage. It is fast, reliable, and has a lower learning curve, making it better suited for simpler endeavors. Comparing PostgreSQL 10 to MySQL 8, both have come to share many features such as Common Table Expression, declarative partitioning, full-text Search, and JSON.
The third option, Oracle, is one of if not the best option in the industry in terms of databases, boasting a set of unique enterprise features like portability, quick recovery, high speed, and performance.. It is however, ultimately too expensive (Oracle license) and heavy (in terms of size, requisite software needed to use it), making it unsuitable for a simple student project.

For the mobile-app, we ultimately ended up going with none of these. Instead, the data in catalogue.json containing all the possible items is stored on a backend hosting server located in Heroku's cloud. The app reads the JSON data from the backend in order to know which items can be added by the user to the calculator.

Choosing the CI/CD tool:

The three main options considered for the CI/CD tool to use were Jenkins, GitHub Actions, and Heroku. Jenkins is one of the oldest, commanding a dominating market share of 71%. It possesses excellent community support, with over 1 million users. Its popularity means that it integrates with all the major version control tools such as CVS and Git. On top of that, there are 1400+ plugins available to serve every possible CI/CD need. The greatest detractor is the cost of the license to use it.
GitHub actions in comparison, is free. It provides the best GitHub integration possible, making it very suitable for the assignment. The GitHub action workflows are also separated into separate Docker runs, which are very easy to reason about and debug. Unfortunately due to the GitHub actions limit, the workflow I wrote is unable to run, and so for the mobile-app, Heroku was used instead.

Heroku provides integration with the GitHub repository. Any change made to the repository will be received by Heroku, which will then automatically checkout the repository, and then rebuild and redeploy. Heroku requires 3 files to deploy a python-based web server. These files are read from the GitHub repository. “requirements.txt” informs Heroku of the required packages and dependencies, “runtime.txt” gives the runtime requirements, and “Procfile” describes the actions executed by the app on startup.