Introduction

Project repository

- GitHub: https://github.com/csc301-summer-2020/assignment1_pair-1-abmohan-nataliamoran

Project Environments

- Staging environment: https://afternoon-brook-48620.herokuapp.com/
- Production environment: https://csc301-a1-nmab.herokuapp.com/

Project android app

- APK: https://github.com/csc301-summer-2020/assignment1_pair-1-abmohan-nataliamoran/tree/master/a1-mobile/apk

Project Video Preview

- Web - Desktop: https://youtu.be/hEerC0v8Leo
- Web - Tablet/Mobile: https://youtu.be/OENyNA1mKvo
- Mobile: https://youtu.be/Nyif3UUj6Dg

We considered Assignment 1 to be an opportunity to prepare better for the team project. Thus, developing the Checkout Calculator application, we chose technologies which would both fit the purpose of the Calculator app and which we could later use for our team project.

Because the team project is going to be web-based to meet the Partner's goals, we focused on web frameworks. To have both web and mobile Checkout Calculator applications we needed to choose a web framework which ecosystem would also allow us to build a mobile application. To be consistent we didn't consider using two different frameworks for web and mobile.

All calculator logic is developed on the backend to prevent web users from adjusting the calculations. Having one backend, database and deployment flow for both web and mobile applications, we can ensure that:

- Calculator backend is robust: application logic (and thus all calculations) and data stored in the database are identical for web and mobile
- Checkout Calculator application is sustainable: updating logic would not require double work and would not create inconsistency between web and mobile
- Backend is not dependent on the frontend - additional frontend applications could be developed and connected to the backend, if needed

With this decision we also combined web and mobile reports for backend, CI/CD and database to choose the same technology for web and mobile.

Frontend

Web Application

Chosen Technology: React
Key Reasons:

- Past experience with React in CSC309
- Better suited for smaller projects of the scale given in the assignment
- 67% of developers prefer React to Angular per StackOverflow survey [1]

TECHNOLOGY	PROS	CONS	CHOSEN
React	• Easy to learn. • Provides good performance. • Offers server-side rendering, which is beneficial for applications with the focus on the content. • Provides the framework ecosystem for adapting an application for different platforms. • Good for SEO	• No class-based components, which will require those who are used to OOP to adjust. • No single source of truth for development best-practices.	✓
Vue	• Easy to learn. • Has detailed documentation. • HTML and JS knowledge is supposed to be sufficient to develop an application in Vue. • Is lightweight and fast. • Good for SEO	• Has a smaller community than React.	✗
Angular	• Development best-practices are well documented. • Structured project allows easier changes.	• Has worse performance (due to two-way data binding). • Model-View-Controller architecture, aiming to improve project scalability, might be an overcomplication for both A1 app and Upendo Honey project. • During early learning stages, requires more time to learn.	✗

[1] https://becominghuman.ai/angular-vs-react-what-to-choose-in-2020-1a0e47e8f810

Mobile Application

Chosen Technology: Ionic React
Key Reasons:
- Keeping React ecosystem while supporting mobile look & feel.
- Code can be deployed on all platforms.
- Project in assignment is not CPU intensive, therefore performance is not an issue.

TECHNOLOGY	PROS	CONS	CHOSEN
React Native	• Higher performance (because React Native uses standard iOS and Android controls)	• It is difficult to customize native controls. • Separate screens need to be built for each platform.	✕
Ionic React	• Develop once - deploy to all platforms (web, Android, iOS) • Progressive Web App (beneficial for SEO and user engagement) • React DOM is supported by a significantly larger number of libraries, compared to React Native.	• App will run in a webview on phones - slower than native code	✓
Flutter	• Compiles to native code - good performance	• Steep learning curve • Does not compile to web	✕
Native (iOS, Android)	• Best performance • Easy to test with native development tools	• Multiple sources for each platform - Difficult to make changes • Steep learning curve	✕

Backend

Chosen Technology: Node.js Express
Key Reasons: We can use the same tooling for our frontend and backend, and don't need the admin interface or ORM provided by Django or the enterprise-level functionality of Java.

TECHNOLOGY	PROS	CONS	CHOSEN
Python Django	● We're somewhat familiar with the technology ● Strong admin interface ● Strong Object Relational Mapper	● No string typing ● Slightly limited support for concurrency and parallelism (limitations of Python's global interpreter lock)	✕
Java Spring	● Strong type checking ● String support for floating point arithmetic ● Strong support for concurrency and parallelism	● Clunky ● Complicated setup ● Longer lead time to spin up prototypes	✕
Node.js Express	● We're already familiar with the technology ● Same tooling can be used for both the front end and the back end ● Large development community ● Plethora of tooling for quickly spinning up prototypes	● JavaScript has challenges with floating point arithmetic ● No strict typing (but workarounds exist, e.g. TypeScript) ● Limited support for concurrency and parallelism	✓

Hosting

Chosen Technology: Heroku
Key Reasons:

- We don’t need the advanced functionality of the Infrastructure as a Service providers
- Heroku comes built in with build pipelines, which allow us to set up a staging and production server without having to run and manage two servers
- No infrastructure management at all (Heroku configures the servers and ensures they are secure and up to date, on an infrastructure level)
- Allows us to focus on application-layer code

TECHNOLOGY	PROS	CONS	CHOSEN
Heroku	• Free for small apps • Sensible defaults allow us to get up and running quickly • Build pipelines built in	• Expensive for larger apps • Harder to fine-tune for complicated apps	✓
Digital Ocean	• Free tier available • Strong containerization support • Middle ground between Heroku and larger	• More difficult to set up • No build pipeline support	✕
Infrastructure As a Service providers (Google Cloud / AWS / Azure / IBM)	• Free tier available • Dozens of high-quality and scalable tools • Capable of handling web-scale production traffic • State of the art tooling	• Much more complicated to set up for simple apps	✕

CI/CD

Chosen Technology: Github Actions
Key Reasons:

- It is simple and functional enough to perform our required functionality:
    - Testing the build process and running tests on pull requests
    - Deploying to staging server upon detecting commits to the master branches
    - Deploying to production server upon creating releases in github
- It does not require us to configure git hooks, which would have required more coordination with the course instructors to set up the necessary permissions on our repository

TECHNOLOGY	PROS	CONS	CHOSEN
Github Actions	● Free ● Easy to set up ● Does not require access to Github repo's admin settings	● New technology, so less developed community ecosystem ● Slower than the alternatives ● Harder to configure ● Limited plugin ecosystem	✓
CircleCI / Travis	● Middle ground between configurability and ease of set up ● Fast	● Requires access to Github repo's admin settings (to configure the hooks)	✕
Jenkins	● Highly configurable ● Large community ecosystem, including dozens of plugins ● Can be as quick as we like (due to its configurability and our control over the infrastructure layer)	● Requires us to maintain our own infrastructure on which to run it ● Requires access to Github repo's admin settings (to configure the hooks)	✕

Database

Chosen Technology: No database. State is maintained in-memory within the express.js server
Key Reasons:

- A database was not a requirement for the assignment
- We only needed somewhere to persist state for short periods of time (i.e. while the user was adding items to the shopping cart)
- We were willing to tolerate the cart being erased if Heroku put the server to sleep due to inactivity
- Low likelihood of race conditions (i.e. only one person is likely to be using the site at a time)

TECHNOLOGY	PROS	CONS	CHOSEN
None (express server in-memory objects)	• Fine for proofs of concept that doesn't require state to persist • No work required to set up • Lower cost	• Race conditions will occur if multiple people are editing state • Heroku shuts down the server during low traffic periods, causing state to be lost	✓
Non-relational (e.g. MongoDB)	• Good choice when data is not relational (e.g. log data) • Data is stored in the same form it's transmitted (i.e. JSON)	• Usually inefficient to join across entities • Difficult/inefficient to implement ACID compliance (and prevent race conditions)	✗
Relational (e.g.Postgres)	• Makes sense when data is relational, and/or when joins would frequently occur across different entities	• Data needs to be serialized/deserialized into JSON • Data needs to be normalized	✗