Tech Stack for Website

Frontend

We examined three popular front end technologies, React.js, Angular and Vue.js. We decided on React, but primarily due to time constraints, our team used plain Vanilla Javascript.

Ease of Development

Our team's base knowledge included basic Javascript, HTML, and CSS. However, React.js and Vue.js are known to be relatively easy to learn. Angular, on the other hand, is a heavyweight framework, with a lot to learn upfront. We felt this might be overkill for a simple calculator app. Another benefit of React or Vue is their component-based structure, so it can be integrated with existing Javascript projects. You can add as little or as much as you need.

Maturity & Popularity

Angular is the most mature of the three technologies frameworks/libraries. It benefits from a strong developed community with active contributions and well-written documentation. Vue is the youngest, with a much smaller development community. React is also relatively new. We have to be critical of new technologies because while they lack convention and standards, as well as thorough documentation. As well, with the fast pace development, there is a need to constantly keep up-to-date with new changes or features.Nevertheless, React has a thriving community, and growing popularity. Since React is also backed by tech giant Facebook, we see that React is not a technology that will die down. We were definitely attracted to React for this reason.

Domains

Learning React allows us to replicate many of the same features in the mobile application version through React Native. This would allow us to develop and roll out a product on multiple platforms more quickly. This is especially beneficial if we wanted to get as much exposure as possible in a short period of time. Angular also provides a cross-platform solution for desktop applications, and mobile through Ionic which can be used to make progressive web apps.

Performance, Scale and Speed

React and Angular are both used by many large companies, thus they are proven to perform on a large scale. Both Vue and React utilize a virtual DOM, which makes DOM manipulation much faster and is not as resource intensive as direct DOM manipulation (which would be the case with Javascript).

Backend

Options for the backend that we considered were Node.js, Django and Ruby on Rails, but we ultimately decided on using Node.js.

Ease of Development

Our team is very familiar with Python. Further, Django also provides many standard features, such as scientific calculations, protocols for image processing etc., so there's reduced manual work. Ruby syntax is intuitive and readable, but we would need to learn it. Node.js uses Javascript, which simplifies the tech stack (since frontend also uses JS) and reduces the amount of learning involved. This is one of the biggest deciding factors for us, and in general, this makes for quicker and cheaper development.

Domains

Node.js is often used in cloud services and IofT. Python is very versatile and has many uses, including machine learning, neural networks and voice and face recognition. It also has significant applications in data science, which makes it a good option in FinTech. It is also used in 3D modelling software, which is great for games. However, these features are unnecessary for our simple app.

Popularity & Maturity of Technology

Ruby on Rails and Django are both very mature libraries with a well-established development community and excellent documentation. There is little hype around Node.js and Ruby on Rails, but they still continue to be used in many contexts.

Performance, Scale and Speed

Node.js is known to have fast performance, as it is interpreted in Google's V8 Engine. Node.js was popular for its event-driven, non-blocking I/O model, making it efficient for real-time applications that process multiple concurrent requests across many devices. It would be ideal for a messaging app or web games. However, Node.js is not built to handle CPU intensive computations. Django and Ruby on rails would be more desirable to build large, complex, CPU intensive apps. In our simple case, Node.js is sufficient.

Database

The options that we examined were MongoDB, Firebase and SQLite. Since our original plan was to set a central server and database for both web and mobile, there is a more thorough analysis of databases in our mobile tech stack solution. We decided on MongoDB initially since it allowed for flexible storage of data types. Further, we had experience working with JSON objects.

CI/CD

We also planned to use the same CI/CD tools as with our mobile app, so an in-depth analysis can be found in that report.

Tech Stack for Mobile Feature

Frontend

Firstly, we made the assumption that only an Android app was needed. Our final decision was to go with native Android development using Android Studio. Although, we also considered React Native and Ionic as alternatives.

Ease of Development

React Native and Ionic are more easily adaptable if you are coming from a web background. Generally speaking, this has many benefits including reducing labour costs as web developers can act as mobile developers. React Native is also cross platform, so there's a common codebase for iOS and Android devices. This results in faster and less expensive development, and lower maintenance costs. The thought was that once the web front was designed, we could more quickly migrate the same features to the mobile version. One general downside of native development is knowledge of specific languages are required, thus development costs tend to be higher. However, it was actually a favourable option for us since our team was already comfortable with Java. Ultimately, we did decide to go with Android Studio as we had more experience with Java and this IDE.

Domains

Ionic and React Native have a web background. As discussed in web tech stack, Javascript can be used for both front end and back end, which reduces cost of development.

Performance, Scale & Speed

Native apps offer the best performance. It handles user input and output very smoothly, and you get access to the full features of the specific OS (including push notifications, vibration etc). Ionic uses web elements, which doesn't offer the best experience. React Native uses mobile specific elements. In most cases, the difference in speed between natively developed apps and React Native is unnoticeable. However, native would definitely be preferable for games. Due to the simplicity of our product, we didn't foresee an issue with performance regardless of which option.

Popularity & Maturity of Technology

Of the technologies, React Native is the youngest. As with React, it might lack some standards, conventions and documentation. It is flexible and subject to more continual change. Javascript is one of the most commonly used languages, but Java is not far behind. Android Studio ranked first as the most commonly used development environment amongst mobile developers. React Native has also gained a lot of traction and popularity over a short period of time. There is a strong development community and massive support for all options. What this means for a customer is that the technology behind the product will stay relevant and won't die out.

Backend

Our plan was to use the same backend for both the web front and mobile front. The reason being that the backend would have very simple functions. Since simple calculations and logic can be weaved into the frontend, the backend would handle communication with the database. There is a more detailed analysis of backend tools in the web report. However, specifically for the mobile app, we ultimately omitted a backend. Since we shifted to an internally stored database, it was simpler to allow the frontend to communicate directly with it.

Database

Like with the backend, we planned to use the same database for both web and mobile. Our concept was to store a catalog of items in the database, then the website and mobile app would make requests to the same server to obtain the required data. The main benefit of this is simplifying the two tech stacks. The options that we examined were MongoDB, SQLite and Firebase.

Ultimately the concept of our product changed, and we wanted the mobile app to work offline. We shifted to creating a standalone app using SQLite, which integrates easily into the Android development process.

Ease of Development

Firebase is said to be easy to set up and use, as it doesn't require complex configurations. Additionally, both MongoDB and Firebase are NoSQL databases. An advantage to this is that you don't need to define a structure for the data being stored right away, as is the case with SQLite (and other RDBMS alike). This can speed up the development process since there's less upfront planning, thus you can get a working product out faster. This is also beneficial when data requirements aren't clear, or are likely to change. This is a big reason why we initially chose MongoDB and kept it for our web application. Since the mobile version came later, it became clear the types of data that we needed to store and we were able to easily define a schema. SQLite was also very easy to set up with few configurations.

Domains

RDBMS like SQLite are often used in e-commerce software, accounting software, and CRM tools, due to the relational nature of the data being stored. The type of data that we wanted to store (items in a user's cart) was also intuitively relational. Firebase and MongoDB are quite useful for cloud-based software systems that need to store different data types, as it provides the flexibility of adding new data types in the future.

Performance, Scale, Speed

While scalability wasn't a major deciding factor for our simple product, it is still worth exploring. NoSQL databases are much more scalable than RDBMS. Specifically, since each object in NoSQL can be independent and self-contained, it allows us to scale horizontally, storing data across multiple different servers, thus massive amounts of data can be stored. That being said, queries are not as efficient or fast as that of RDBMS. Thus, it would not be suitable for a program that requires heavy data analysis. Although, MongoDB offers some similar characteristics to RDBMS. It has a querying language of its own, that makes querying much more effective.

Popularity & Maturity of Technology

According to the Stack Overflow 2019 Report, SQLite and MongoDB are amongst the top most commonly used, and 2x times more commonly used than Firebase. Nevertheless, Firebase still has a large community with 1.5 million apps based on Firebase globally. SQLite has been used for a lot longer (as with all SQL databases in general), while NoSQL databases are still relatively new. Each of the options have strong support. It comes down to the type of data being stored, and what it will be used for.

Other Considerations

While Firebase and SQLite are both easy to set up and integrate with Android apps, ultimately we went with SQLite. Firebase is a cloud-service and data is hosted on a third-party cloud provider. Unfortunately, Firebase cloud real estate is not free. Furthermore, even though Google is the provider, we should be cautious that the reliability of your data storage depends on a third party. SQLite stores data on the devices, and it is free to use. Security is also important for databases, depending on the sensitivity of the data being stored. MongoDB and Firebase offer more extensive security features compared to SQLite. However, in this context, that was an acceptable trade-off since we weren't storing any personal or classified information.

CI/CD Tools

We considered Jenkins, CircleCI, and the one we ultimately decided on, Travis CI.

Ease of Development and Setup

Travis CI interested us because it has direct connectivity with GitHub projects, and is simple to set up. It automatically builds on each push to the connected repository. Further, it is free for open source projects. Jenkin requires a more elaborate setup, with many configurations that need to be manually set up. Jenkins is also open source and free to use. CircleCI provides an out-of-the-box solution, meaning easier set-up, but it isn't free.

Popularity and Maturity

Jenkins is a mature tool, and one of the most popular CI/CD tools, so there is a large active community. There is a shift towards container-based CI/CD pipelines. CircleCI builds in dedicated containers so that each build doesn't get affected by other dependencies and files populating on the server. Travis CI similarly recreates the virtual machine after every build.

Domains

Jenkins' is known for its cross-platform capabilities and extensive plugins, which allows for greater flexibility and customization. Although Travis CI and CircleCI are Saas-based, cloud-based tools that provide a simpler solution for smaller projects.

Performance, Scalability & Speed

Cloud-based server plans are more scalable, robust and allow faster deployment. You can set up parallel builds in both Travis and CircleCI for faster execution of multiple processes. Since they are cloud-based, you don't require a dedicated server, cutting the cost of maintenance. Jenkins requires a dedicated server that needs constant maintenance. Further, Jenkins' web UI is outdated, and with the many plugins that are involved, it might become slow.

Debugging & Maintenance

Jenkins servers need to be maintained by a dedicated team, who manually install dependencies. With CircleCI, dependencies are automatically installed with every build so the process. Travis CI's free cloud hosting also eliminates a lot of maintenance work. CircleCI offers the feature "Debug via SSH" which makes debugging quicker and easier. This differs with Jenkins where you are required to manually debug by clicking on jobs. Overall, CircleCI offers more effective automation. CircleCI is a worthwhile investment for small and large companies alike. For large projects, Jenkins provides an excellent free open-source option. For a small project like ours, TravisCI seemed to be the simplest and free option.