CSC301 A1 Report

By Christine Lee and Jacky Yang

In this report we document our thought process in developing and deploying our PokéMart web and mobile shopping application. Our analysis of stacks and technologies will include points from varying perspectives: That of a consultant building this for a client, as well as ourselves, building something we are proud of within the time crunch of A1.

Mobile

It is becoming increasingly important for an online application to be accessible through all platforms. Especially for a shopping application, where our target users are not restricted to a single platform, we need to research how to deploy on both Android and iOS.

Native languages are an obvious place to start our research. For iOS, that means Objective-C or Swift. Objective-C is familiar as it is based on C which we learned in CSC209. Being based on C, many of the skill sets acquired working with it can also translate to many different applications coded in the C-family such as hardware for our future projects. Additionally, Objective-C is mature, well-tested and stable compared to the fairly new and evolving Swift. But at the same time, Swift's new and modern design gives it an edge in popularity and future potential. This means resources and support for Objective-C may move to Swift, especially since Apple is backing Swift as 'the future'. Swift also has a friendlier syntax and is shown to be faster in many areas.

As consultants, finding developers to maintain the application may prove difficult if less developers are learning Objective-C. As developers ourselves, the decrease of popularity causes concern for future support.

On a humorous note, there's a popular joke that C-programmers and non C-programmers all have one thing in common: Hating C. So it could be said another big downside for Objective-C is *it is based on C*, and that it may be easier to learn Swift than review C syntax and intricacies.

For Android, we have a choice between Java or Kotlin. Our familiarity with Java from developing an Android application in CSC207 along with Java's many applications outside of Android development (like creating Minecraft) makes it a more attractive choice for us over Kotlin. However there are still a number of advantages as well as disadvantages to be considered for both Kotlin and Java. Despite being fairly new, Kotlin is well known for being more efficient than Java in many areas, as well as being stable and with good Android support from Google and Jetbrains. But with its younger age, Kotlin's community and available resources is incomparable to Java. Outside of Android, Java has vastly more use-cases, applications across platforms, and more support. While Kotlin has a growing cross-platform community, it still remains incomparably small. Within Android however, Kotlin holds comparable resources and market share to Java and is quickly growing. Since our shopping application is small, mostly static, and with little potential or plans to expand, it will be better to reap the immediate benefits of Java's familiarity, support, and versatility versus the subtle optimization and future potential of Kotlin.

Using native solutions such as Swift and Java would bring extra optimizations, full device integration, security, and others. But they are unnecessary for our simple, mostly frontend, store application. Thus we would likely see a better return on investment looking into cross-platform stacks.

For cross-platform technologies, we mainly looked at two--React Native, and Xamarin. Xamarin is the most interesting as it allows us to build iOS and Android apps with Microsoft's .NET platform. .NET is very versatile and has applications in gaming, IoT, ML, etc. It's mature, with a lot of resources being maintained by Microsoft amongst others. Along with Xamarin, other tools also exist for building applications on most platforms. However, due to our extreme lack of familiarity with .NET, Xamarin was not a viable option.

React Native sits in the opposite branch of familiarity. As we are already acquainted with React and Javascript fundamentals, React Native's familiarity immediately attracted us. It would be fast to learn and start building. Being based in javascript, it'll also opens the possibility of sharing components and functions between our mobile and web applications. The framework will simplify the frontend and sits on top of javascript that can be used for frontend or backend grunt work. In terms of maturity, the React Native framework is still in Beta, and can't compare to the well-tested .NET. React Native's lack of maturity is concerning for our development process as our short assignment time frame denies additional delays for troubleshooting. That being said, there already exists a plethora of tools, resources, and a giant community we can utilize. It also remains the most widely adopted tool for mobile, although competitors like Flutter will come to challenge that. A notable downside that came up in our research is the performance of React Native compared to Native languages and even close competitors like Flutter.
In all our research, React Native stood out as the perfect technology for our PokeMart shopping application. It's familiarity and simplicity was a decisive factor in allowing us to

start and build quickly in order to meet our deadline. It’s immense popularity meant we would be able to receive quick support from Google or YouTube through our development. Lastly, its use of javascript would allow us to further decrease build time and opens up possibilities of web and mobile developers in our team to help the other.

Web

Our research into web stacks went in parallel with our mobile language analysis. It was a very interesting process to finally compare popular frameworks that we’ve been hearing as computer science students. We researched popular frontend javascript frameworks such as Vue, Angular, and React. As well as backend frameworks like Ruby on Rails, Django, Flask, and Symfony.

Vue, Angular, and React are all based in Javascript which would allow sharing of functions with React Native. We would also be able to use our existing experiences with Javascript, versus diving into a completely foreign territory like Ruby on Rails. In terms of familiarity of the frameworks, our team is better versed in React, which allows us to skip the initial learning curve. Aside from React, it’ll be easier to transition our knowledge to Vue as it is comparatively simple and flexible, while Angular will require learning new skills like TypeScript and MVC.

A framework’s maturity is very important for us to gather resources and libraries to aid our development. Vue is the youngest of the frameworks, followed by React, and Angular being the oldest and most mature. While Angular has a notable lead in age against React, both language’s maturity are comparable from the backing of tech giants (Microsoft and Facebook respectively) and immense popularity. React has more tools and community contributions, but that is attributed to its minimal design, versus Angular’s all-in-one approach. Vue on the other hand lacks a giant organization’s backing (especially in the West) and depends on open-source contributions.

When considering the expandability of a project, we must also consider the versatility of a framework and its domain size. Vue is commonly regarded as the most flexible, especially in its design for handling HTML templates. It would allow for a comparatively easier migration over to the other frameworks, and other customizations. React is second, with a vast amount of libraries and tools. Angular is the most strict in its enforcement of things, allowing for enforcement of better code practices, at the cost of reducing versatility and customizability.

Popularity is a huge factor in our decision as it will provide us with a general gauge for the availability of support through forums, tutorials, or libraries/tools throughout our learning and developing process. React and Angular are by far more popular than Vue. The trends through the years do not provide a decisive winner, but still give a good illustration that we cannot go wrong in picking either.

While performance is not an initial concern of our tiny application, it will determine future scalability. Especially for our specific purpose of building a shopping application, Vue appears to hold the advantage due to its tiny library size. Multiple articles have cited results where Vue finishes ahead in startup and memory tests. Angular holds a different advantage in DOM manipulation because of its MVC design. But we won't need much DOM manipulation for our mostly static application. Overall, the framework's performances are relatively similar, and while research gives us a good perspective, we wouldn't expect to recreate these results within our unique use cases.

Frontend conclusion

In the end, we decided to go with React Native and React to build our frontends for iOS, Android, and web. We recognize the performance may be lacking compared to many of the other technologies we researched. However, we believe the benefits from familiarity and transferable code between mobile and web will outweigh the cons.

Backend and Databases

For the backend, we considered first Django, Ruby on Rails and Flask. We are most familiar with Python and thus Flask and Django. However it is Ruby on Rails that is the most mature, with large libraries of plugins available and a massive community. As for versatility, Django covers a larger domain, being able to handle full-stack development, not just server side. We see it as a huge advantage to consolidate development of both front and backend. In terms of popularity, Flask and Django lead in 2020, but overall, all 3 are closely tied. Lastly, all 3 frameworks are considered very fast in different areas, as we will mostly be doing simple processing, performance was not much of a consideration.

For databases, we considered multiple relational and non-relational databases. Of these, MySQL, PostgreSQL, and MonogoDB were contenders for their familiarity and popularity. We are most familiar with PostgreSQL from CSC343 databases course. It's object-relational design will also make it easier to learn using previous experience, as opposed to a strictly relational database like MySQL. In terms of maturity, it surprised us the gap between MySQL and PostgreSQL and MongoDB. While all these choices have over 2 decades of usage and are all mature with big communities and tools, MySQL and PostgreSQL have a slight advantage of being more popular and much older. For performance, the simple potential lookups of our shopping application doesn’t require much consideration, but while all have their areas of expertise, our application can reap potential optimizations from a relational database such as MySQL or PostgreSQL.

As we researched multiple backend frameworks and technologies, we realized that our shopping application has no need for a complex one. We also do not need a complex API or database as we don’t have a system yet for storing differ user’s data. Instead, we use Redux in our frontend to store and update temporary information, such as the user’s cart, and kept product information static/hard-coded in a data file as we did not have the need for a constant addition/subtraction of new products.

CI/CD

We considered Github Actions, CircleCI and TravisCI. Through multiple sources, all these options have similar popularity, performance, and basic functionality. The biggest distinguishing factor appears to be price. Any differences in performance and functionality was beyond our scope of use and knowledge. Thus for our use case, the learning curve is the deciding factor. Github Actions is the most familiar and convenient out of all since it is also our development environment. Unfortunately, we were not able to produce a working CI/CD workflow for our applications as the action limit for Github Actions had been reached.