TravelSnap

TDS200 Cross-Platform

Candidate 2009

Høyskolen Kristiania

04/12/2023

As the Tech Lead for the development of TravelSnap, I recommend a mobile-first approach to ensure the app is user-friendly and accessible. This aligns with the company's primary objective of creating a mobile application that enhances travel experiences. To determine which platform to prioritize, I suggest conducting market research on the distribution of Android and iOS users in Norway. This information will help determine which platform has a more extensive user base and should be prioritized for development. Considering the rising interest in travel blogs and the importance of time-to-market, I recommend using a hybrid mobile app development framework such as React Native or Flutter.

It is essential to consider the distribution of Android and iOS users when developing a mobile application. Market statistics from 2023 reveal that in Norway, roughly 65% of web traffic comes from iOS users, while Android users account for approximately 35%. (*Publish Online*, n.d., p. 74) Other sources also indicate similar trends, with iOS representing roughly 62% and Android around 38%. (*iPhone Market Share by Country 2023*, n.d.). Given this data, prioritizing iOS development for the TravelSnap app is likely the best strategy to reach a wider audience.

I will briefly discuss the core concept of how these two ecosystems work. React Native uses fabric to render views; the underlying computations are done in C++. However, developers can write JavaScript code and declare it as a source of truth to generate C++ structures to operate with host platforms. In practice, this means JavaScript code is directly translated into native views and components specific to each platform, for instance, objective-c in iOS or Java for Android. This is the core concept of the bridge between JavaScript and the host platform, and React Native facilitates this bridging. In JavaScript, views are referred to as a tree representation, the building block. The layout constraint uses Yoga (*Yoga Layout | A Cross-Platform Layout Engine*, n.d.). The actual calculation of each component can be broken down into something

called React Shadow Tree. This is a node packed with layout information such as x, y, width, and height. This can be visualized as a heap of layouts upon each other, and each of the planar layouts may consist of components such as text or images. Legacy React used JSON to communicate with host platforms, i.e., Android or iOS. But this has been improved and transitioned into Fabric render, a JavaScript interface API to embed the JavaScript engine in C++. (*Cross Platform Implementation · React Native*, 2022). Most fabric rendering happens on the UI thread or main thread and runs on a JavaScript engine like Hermes.

On the other side of the spectrum, we have something that is a relatively young and growing framework, Flutter, that uses Dart language. Dart is a compiled language that can compile all the way into the host platform's native code. This outperforms JavaScript bridging. Due to Dart's advanced compiler technology, Flutter is a versatile framework that offers cross platforms from native devices using Dart engine Just-In-Time and Ahead-Of-Time, which offers fast and responsive hot-reload. The difference is, just as the name implies, JIT compiles on the fly, which makes hot reload superfast, and AOT is pre-compiling whatever is necessarily needed for the environment to work. It also has a web compiler, especially targeting JavaScript for web development. (*Dart Overview*, n.d.).

Needless to say, both frameworks are flexible and easy to adapt, which is an essential factor for being productive. One specific advantage of using Flutter is you can control every widget as they call their components. In contrast, in React Native, you can control code to a lesser extent unless you can fully understand native code on all host platforms. (Osetskyi, 2020) If, for some reason, a bug seems too cryptic to understand unless you are very technical with native code, it is tough to look for solutions and not productive at all. (Peal, 2023a, para. 19) Besides, it also ruins the purpose of hybrid development and the production experience. We can conclude that many

decide to adapt to a hybrid development workflow to write less code that can target many audiences, and most importantly, it is effective for developers to adapt to the environment, as most don't have expertise in all kinds of native code. (Peal, 2023b, para. 5) When working within time constraints, we must ship a one-off solution, making it too risky to explore new frameworks, so we don’t have a choice but to use React Native.

To conclude, the most appropriate app development approach is having sufficient experience and, as a whole, development team, how much coverage you can do. What are the team’s technical background and their expertise? Everything has to be planned and adjusted accordingly. Without a proper plan, the workflow and experience will not be smooth, leading to many pitfalls. As a technical lead for our project, making the right decision is critical. Knowing that our production team comes from a JavaScript/TypeScript background, we may have a more effortless onboarding experience when using React Native for production, but dealing with bugs is inevitable. It may face difficulties when developing due to inconsistent development production. React Native is still native and does not fully abstract away the need to understand how things work technically when building and debugging; this is a painful experience where you don’t even know the root cause. Take an example of file system management in React Native Expo on Android. It is extremely hard to bypass its strict permission policy because you lack root access. Thus, you can’t manipulate solely on Expo; it requires you to dive into React Native CLI, which requires access to native files. This is from personal experience when working with files like images. Deciding which platform depends on the factors and other variables is key.

An honorable mention is the new Capacitor developed by an ionic team that replaces the Cordova, previously known as PhoneGap, and is also used to bridge communication between

platforms when approaching a hybrid development. At a glance, the Capacitor still acts as an intermediary between native code and the Ionic framework. This is also clearly stated in Ionic developing native web apps and creating cross-platform applications using web technologies like JavaScript, HTML, and CSS. The runtime environment is on Node.js, and to develop natively, you must utilize Web View. (*Getting Started | Capacitor Documentation*, n.d.)

After a thorough discussion with our production team, we agreed to opt for React Native as the framework for our production. The team's previous background in React will make the production environment slightly faster and more productive. Although Flutter seems highly advantageous over React Native, the risk is too high to adopt new technology into production; besides, none of the production team has experience with Flutter. Both Flutter and React Native have their strengths and limitations. React Native has been around for a while, and naturally, with a larger community, it contributes to a more mature and stable platform. While Flutter is relatively new with a smaller community in comparison, the limitations are less well known and might affect developers' ability.

Hybrid applications are challenging to develop; there are many things to assert, and you need to gain extensive knowledge in every field to avoid nontrivial challenges and problems. Using the right tools for the use case is the optimal approach. The main idea behind hybrid development is to ship out production fast and time to market. A unified, standardized approach to continuous integration and delivery is more optimized and efficient to maintain. From a common user perspective, they can't be able to distinguish applications made by a hybrid approach or those made natively; they expect them to function properly. (Malavolta et al., 2015, p. 25). When deciding to publish an application on Google Play, the application can either be produced native or hybrid. An empirical study was done by splitting applications made native and hybrid. The

result shows that most negative reviews from applications made by a hybrid approach can be categorized as non-data intensive, but require more low-level hardware interaction with platform-based operations like games. The positive can be categorized as data-intensive, such as social applications. They are focused more on content delivery, navigation, and standard read-write operations. (Malavolta et al., 2015, p. 28).

We can conclude that, if an application made hybrid requires extensive access to the hardware, it heavily depends on plugins or APIs, which are more or limited to an extent. Another study has also shown that a hybrid application is more likely prone to bug and security issues. (*The Impact of Cross-Platform Development Approaches for Mobile Applications from the User's Perspective | Proceedings of the International Workshop on App Market Analytics*, n.d., p. 46). Google also promotes Progressive Web Applications in contrast to the mobile-first principle in Responsive Web Design, where you have to take into account the coverage of all device sizes. In contrast, every browser has its limitations of modality. The idea behind PWA is to unify the web platform-based application on a more native feeling that delivers a seamless user experience using a mix of web technology and native components. With good design and interaction, this architecture can offer a cost-effective production while targeting a broader audience. (*Analyzing User Experience in Mobile Web, Native and Progressive Web Applications | Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems*, n.d., p. 1).

Another important aspect of developing a social app using a hybrid approach is handling privacy issues. Security poses a big threat, especially on Android devices. This is because the JavaScript bridge communicates with Java code over sinks, an access point for code execution. The ability to inject JavaScript code to perform an Intent on Android that can perform system-level

operations exposes vulnerabilities and, especially when code injection is not sanitized, may execute cross-side scripting even from a web-based technology. (Bai et al., 2019, p. 681).

The decision to choose Firebase as the backend aligns with the production objective and plan. Firebase provides a range of beneficial services for applications like TravelSnap, which requires a database to preserve user data. It offers synchronized real-time data, authentication, and fast development. Firebase offers real-time updates across all clients, which is critical for achieving a smooth development experience and user experience. It even works offline in case of network disruption. Without diving too deep into how it works technically, the concept is that a database is cached locally and uses something called service workers to read the cached database instead of from the network. Then, the reconciliation happens when you are back online. This simulates the experience as if you were online all the time. (*Architecting Mobile Web Apps(Google i/o'19)*, n.d., n. 29:57). Firebase utilizes a concept they call 'stealing', which involves splitting up script modules into parts and preloading them when necessary. This saves traffic and contributes to a greener development. (*Architecting Mobile Web Apps(Google i/o'19)*, n.d., n. 15:00). For TravelSnap, Firebase offers a comprehensive, efficient, scalable backend solution that is easy to integrate and aligned with the objective. This is an ideal choice for a user-focused app like TravelSnap.

The landscape of developing mobile applications using cross-platform is complex and challenging. On the one hand, it offers numerous options for developers and is a cost- and time-saving solution. On the other hand, it involves many technologies, which can overwhelm some developers. The fragmented ecosystem also contributes to complexity, making it difficult to navigate. ("A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development | Acm Computing Surveys," n.d., p. 108:28)

Reference

A good start in react native. (n.d.). Retrieved December 14, 2023, from https://www.netguru.com/blog/a-good-start-in-react-native

Analyzing user experience in mobile web, native and progressive web applications | proceedings of the 17th brazilian symposium on human factors in computing systems. (n.d.). ACM Other Conferences. Retrieved December 14, 2023, from https://dlnext.acm.org/doi/10.1145/3274192.3274201

Architecting mobile web apps(Google i/o'19). (n.d.). Retrieved December 14, 2023, from https://www.youtube.com/watch?v=NwY6jkohseg

Bai, J., Wang, W., Qin, Y., Zhang, S., Wang, J., & Pan, Y. (2019). Bridgetaint: A bi-directional dynamic taint tracking method for javascript bridges in android hybrid applications. *IEEE Transactions on Information Forensics and Security*, *14*(3), 677–692. https://doi.org/10.1109/TIFS.2018.2855650

Cross platform implementation · react native. (2022, March 10). https://reactnative.dev/architecture/xplat-implementation

Dart overview. (n.d.). Retrieved December 14, 2023, from https://dart.dev/overview.html

Getting started | capacitor documentation. (n.d.). Retrieved December 14, 2023, from https://capacitorjs.com/docs/ios

Iphone market share by country 2023. (n.d.). Retrieved December 13, 2023, from https://worldpopulationreview.com/country-rankings/iphone-market-share-by-country

Malavolta, I., Ruberto, S., Soru, T., & Terragni, V. (2015). End users' perception of hybrid mobile apps in the google play store. *2015 IEEE International Conference on Mobile Services*, 25–32. https://doi.org/10.1109/MobServ.2015.14

Osetskyi, V. (2020, October 9). *Best hybrid app framework in 2022—Existek blog*. https://existek.com/blog/best-hybrid-app-framework/

Peal, G. (2023a, July 5). React native at airbnb: The technology. *The Airbnb Tech Blog*. https://medium.com/airbnb-engineering/react-native-at-airbnb-the-technology-dafd0b43838

Peal, G. (2023b, July 5). Sunsetting react native. *The Airbnb Tech Blog*. https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a

Publish online. (n.d.). Retrieved December 13, 2023, from https://indd.adobe.com/view/a1b88e4a-cb86-446a-bbb2-265829b540a8

The impact of cross-platform development approaches for mobile applications from the user's perspective | Proceedings of the International Workshop on App Market Analytics. (n.d.). ACM Conferences. Retrieved December 14, 2023, from https://dlnext.acm.org/doi/10.1145/2993259.2993268

Yoga Layout | A cross-platform layout engine. (n.d.). Retrieved December 14, 2023, from https://yogalayout.dev/