# **Assignment: Architectural Decisions** **Native, Web, or hybrid app:**

**Prologue:**

In the context of a social networking app for a university, we decided to opt for a hybrid app development model. We decided this to achieve maximum cross platform compatibility accepting that performance may be affected as well as affecting development time.

**Context:**

We considered factors such as development time and performance as mentioned above. Specifically, we looked into how going hybrid would reduce code writing as React Native would heavily reduce development time as we would be able to code for both IOS and Android devices. Unfortunately, there will still be issues as not 100% of the code can be written in this way and we also wanted a web version for those who do not have access to mobile devices. We will still require native modules for certain features such as using a cloud service. We wanted to make sure whatever device, whether it was windows, Linux, mac OS, iOS or Android, every stakeholder would have the ability to use our app meaningfully. The priority is reach and making sure the app has maximum accessibility to reflect the heavily diverse university population.

**Solution:**

We are planning to use React Native to code the majority of the app due to its multi-platform accessibility. Using React native will drastically decrease development time for creating the app on iOS Android, and other devices. We will then code and create the modules of the app that require native features such as certain APIs so some knowledge of native codes like Xcode will be necessary as well. Overall, by choosing a hybrid app development approach, we are able to achieve a balance between cross-platform compatibility and native performance, with the ultimate goal of seamless user experience across different devices with maximum performance possible.

**Consequences:**

The hybrid approach will make the development process much easier as well as the continued updating and maintenance of the app after it is deployed. However, we may still run into some issues such as the iOS version requiring some extra steps and some additional modules written in Xcode for certain release-related data like provisioning profiles. It is also important to make sure we test the app across as many devices with different performance capabilities to ensure we have developed an app that can be run on all devices. We will also need to be diligent to address and fizz any issues that may arise after deployment.

**UI Framework:**

**Prologue:**

In the context of a university social networking app, facing the decision of selecting a UI framework for app development, we decided to use React Native due to its cross-platform functionality, performance and its popularity which would then make maintenance easier if we are no longer available to work on the app after deployment.

**Context:**

We knew we needed to make a visually appealing app with many user-friendly features. This app needs to be easy to understand and navigate as the university population is incredibly diverse with many people who have English as a second language or other users who will be much more “technologically challenged” and struggle with using the app. We did consider other options such as flutter but since our experience is with react native, we decided it was the better choice.

**Solution:**

React Native allows us to write the code once and deploy it to both iOS and Android platforms. With this single codebase it will be easier to prototype and test our app through development as well as update it in the future. The flexbox features will make it much easier to code for different screen sizes as well as rotating screens on mobile devices. The higher level of community support and third-party apps will also assist us with expediting our app development as well as have more thorough testing for the device performance levels and different platforms.

**Consequences:**

We should see much more rapid app development since there will be a single code base for the different devices with only a few modules requiring native code development. This will allow our team to focus more on the UI development, specifically making it more user friendly as well as making it easy to understand and use. Over the long term, we will need to continue to monitor the app and use user feedback to identify the areas of improvement and fix errors.

**Backend Language:**

**Prologue:**

In the context of developing a social networking app for a university, facing the decision of selecting a backend language for app development. We decided on node.js due to its asynchronous I/O model which will be crucial for dealing with the higher volumes of requests by all the university students using the app as well as our familiarity with it. We will accept the potential challenges with scalability that node.js could present.

**Context:**

Some of the options we considered in addition to node.js were python, ruby on Rails and Java. Factors we had to make sure we considered were the prioritizing of performance and the ability to be developed with a hybrid model which led us to node.js through React Native. The asynchronous I/O model, event driven architecture, single thread event loop and performance optimization all combine to increase performance for the app.

**Solution:**

Node.js has a vast library of third-party apps and community support which will support our development team. As mentioned above, the Asynchronous I/O will help handle multiple concurrent connections with lower resource consumption. The event-driven architecture and single-thread event loops for things such as incoming requests or data streams assist in handling the higher volume of requests without sacrificing performance. The performance optimization techniques node.js offer are crucial for maximizing performance. Node.js offers optimized code execution, caching frequently accessed data and employing efficient data structures to help with performance. The large library of community driven modules will also provide us with assistance should we face bottlenecks and accelerate development from the pre-built solutions to common tasks and functionalities.

**Consequences:**

Our decision to make node.js as our backend language will ensure optimal performance and scalability of server-side infrastructure but it may introduce challenges in CPU-bound operations and tasks. In the long run, we will have to monitor server performance to catch and address any scalability issues and ensure a seamless backend infrastructure.

**Permissions:**

**Prologue:**

In the context of developing a social networking app for a university, facing the decision of defining permissions and access control mechanisms, we decided to implement role-based access control (RBAC). We chose this method to ensure security policies and manage user permissions, we are accepting the potential downside of higher complexity in permission management.

**Context:**

We need to make sure we protect user data and ensure that each user has appropriate access to app features. Considering factors like user roles, data sensitivity and number of users, we narrowed it down to role-based access control (RBAC), attribute-based access control (ABAC) and discretionary access control (DAC). Ultimately, we chose to use role-based access control due to its ability to define user roles and grant or restrict permissions based on the roles.

**Solution:**

Using RBAC allows us to define roles for users and associate permissions with each role, this grants or restricts access to certain app features based on their role such as professors having the ability to hide grades until due dates or make classroom announcements. This method will minimize the risk of unauthorized access to sensitive data and give us control over user permissions. By adhering to the role-based system, data breaches or unauthorized modifications can be massively reduced as each role has strict privileges. RBAC has flexibility and scalability as well, so as the app develops and new features are added, the user permissions as well as the accessibility of roles can be modified by an administrative team.

**Consequences:**

The decision to implement RBAC will grant us granular control of user permissions and improve security and data protection. There is a risk of high complexity for what certain users are able to do so over the long term we must monitor access patterns and be diligent in catching unauthorized access and have a fast response to updating the app should users be lacking a permission that their role should have.

**Data Storage:**

**Prologue:**

In the context of developing a social media app for a university, fencing the decision of selecting a data storage method for app data, we decided for a combination of relational databases (MySQL) and NoSQL databases (MongoDB) to utilize the strengths that each version of databases has to offer and accepting the additional challenges of using multiple data storage models.

**Context:**

We needed to choose a data storage solution that is flexible, allows scalability and can have high performance. Important factors that we considered were the data structure, how complex the search queries will be, the options for scalability and data consistency. For relational databases we chose MySQL due to having familiarity with it on our team and for the NoSQL database we narrowed it down to Cassandra and MongoDB and chose MongoDB for its multi cloud support.

**Solution:**

Relational databases excel in handling complex search queries and managing structured data, but they fall short when dealing with unstructured data as well as having limited options for horizontal scalability which will be a problem in the future due to the high volume of data and user load. Relational databases will be used for things such as user profiles and class schedules. NoSQL databases will be used to counter the downsides of the relational database. NoSQL databases can handle unstructured and semi-structured data much better and have the ability to be scaled horizontally through distributing data through multiple nodes. As the data schemes evolve over time, NoSQL will be able to support the increasing user loads while minimizing performance issues. NoSQL databases will be used to store things such as user preferences and user generated content like posts. By combining these two models, we will be able to ensure optimal performance and have the ability to increase scale as the data structures evolve and grow in volume.

**Consequences:**

Although using both relational databases and NoSQL databases will help cover each other’s weakness and give us the ability to leverage each of their strengths. We will have to continue to monitor database performance and be diligent about maintaining data management and synchronization due the increased amount of complexity that bringing both models entails.

**Additional Frameworks:**

**Prologue:**

In the context of a social networking app for university, facing the decision of adding additional frameworks, we decided to utilize Firebase for its cloud messaging ability which will allow us to use push notifications and the React Native Accessibility API to ensure that users have enough accessibility features. Additionally, we will look to develop keyboard navigation, text to speech and high contrast mode. We are accepting the downside of increasing the development time and complexity of the app to ensure there are enough accessibility features for our users. Lastly, we need an offline mode for the app as well.

**Context:**

We need to be able to send push notifications to students and Firebase will allow us to do that as well as provide backend services like user authentication and assistance with data synchronization. Firebase can provide these services while reducing development time for these features and allow scalability in the future.

We also need to make sure we have enough accessibility features for our users like high contrast mode and text to speech for low visibility users, and keyboard navigation as well to increase inclusiveness for our app. React Native Accessibility API will help with providing some of these features and help us with the UI development with these features in mind. Since we are already using React Native to develop the app, the accessibility API will mesh nicely in the development cycle. An offline mode will use cached data in the local system and still be usable for users with the limitation of not being able to get updates or notifications until an internet connection is established.

**Solution:**

By integrating firebase into our app, we will be able to leverage all the services it provides which will include assistance with data synchronization, cloud messaging for push notifications as well as analytics for monitoring our app. Firebase will be in the backend and provide us with improved scalability in the future.

React Native Accessibility API will allow us to implement features like screen reading and text to speech, focus management and easier navigation for those with disabilities. Additionally, we would like to add a high contrast mode to make reading easier for those with low visibility and we would like keyboard navigation as well to improve accessibility. We will implement an offline mode for users without a stable internet connection, this mode of the app will utilize enough locally stored data to make the app still usable and will have the ability to update once a connection is established if the user wishes.

**Consequences:**

Integrating Firebase will streamline development efforts, improve scalability and simplify the backend development tasks like data synchronization. Of course, by using firebase we will be limiting our ability to migrate in the future, and we will have a partial dependency on google. Acknowledging this, we must minimize our dependencies where we can and maintain portability.

React Native Accessibility API will be easy to integrate into our app as we are already using React Native. We will have to accept the downside of React Native Accessibility API may lead to inconsistencies between the web, Android and iOS versions of the app. By thorough testing we can catch inconsistencies, fix them before deployment, as well as fix them as they are discovered after the app is launched. Other accessibility features like the keyboard navigation and the high contrast mode will increase development time but we believe including these features is necessary for user inclusivity. Lastly, the offline mode will be crucial as many users will not always have a stable internet connection, but they will need to have access to the app. We must monitor the offline mode to make sure the core features of the app are still usable and there is an effective local storage system for data.