Comparison of Tools and Languages

The tools and languages being compared were all suggested by the client for the software. In cases where this isn’t true, a special note will be added.

**Desktop (Java vs** [**Node.js**](https://nodejs.org/en/)**)**

**Java**

**Pros**

* Faster than Node.js
* Platform agnostic
* Stable documentation and features
* Much of the dev teams already knows Java

**Cons**

* UI tools are not great (JavaFX, Java AWT, Java Swing, etc)
* Rapid development could be more difficult than JavaScript
* Integration of future, functional code with any UI made before the mid-term could pose issues

**Node.js**

**Pros**

* Faster coding time than Java
  + JavaScript is typically easier to create initial prototypes in compared to Java
* Better UI tools than Java leveraging HTML and CSS
  + Using HTML and CSS will allow us to reduce cross-platform UI problems and create UI which will work on any device with relative ease.
* Lighter weight / more built-in tools which are relevant to the project than Java
* Access to a large amount of pre-built libraries using npm

**Cons**

* Features and documentation change frequently
* Dev team would need to learn the Node and JavaScript platform if they do not already know it

[Qt](https://www.qt.io/) is also a potential option, being cross-platform and using C++, but is likely not worth the hassle of learning a new eco-system in a relatively difficult and quirky language like C++.

**Mobile (React Native vs React native Expo)**

React Native and React Native Expo are both frameworks for building mobile applications using React.js. Here are some key differences between the two:

1. Development workflow: React Native requires developers to set up a development environment for each platform (iOS and Android) separately. React Native Expo provides a development environment that is already set up, making it easier to get started with development.
2. Package management: React Native uses npm as its package manager, while React Native Expo uses the Expo CLI, which is a command line interface for managing the packages and dependencies of a React Native project.
3. Access to Native features: React Native allows developers to have full access to all the native features and APIs of the target platform, while React Native Expo provides a more limited set of APIs and features, but also provides a more consistent development experience across platforms.
4. Deployment: React Native requires you to create a standalone app and submit it to the app stores, React Native Expo uses an "over the air" deployment system where you can update your app without going through the app store.

In summary, React Native is a more flexible framework that allows for greater control over the development process and access to native features, but requires more setup and configuration. React Native Expo provides a more streamlined development experience, with a simplified development environment and package management, but with a more limited set of features and APIs.

React Native has native features (features specific to the hardware and OS of the target platform) which will help with performance of the application whilst Expo forgoes the native features to foster a simpler development experience which will cost performance in the long run.

However, we only have a short time to create this software and it is advised that a senior native developer offer guidance to take full advantage of the native features while Expo has a faster onboarding process, which is helpful because we are just beginning.

React native requires Mac OS, whether that be a Mac computer or a vitual machine on windows, One other solution is to use a program called Codemagic to build the applications. Expo allows us to run the code by downloading an application on our phones be it iOS or Android.

Most of the tutorials I found for map api usage on youtube seem to be using expo even though I was searching for react native specifically

**Backend (SQL vs Mongo DB)**

SQL (Structured Query Language) and MongoDB are both popular database management systems, but they have several key differences:

1. Data model: SQL uses a relational data model, in which data is organized into tables with rows and columns, and relationships are defined between tables using foreign keys. MongoDB uses a document-oriented data model, in which data is stored in semi-structured documents in the form of key-value pairs.
2. Query language: SQL uses a declarative query language that allows users to specify what data they want, and the database management system figures out how to retrieve it. MongoDB uses a more flexible and expressive query language that allows users to specify the exact location of the data within the documents.
3. Scalability: SQL databases are typically vertically scalable, meaning they can handle more load by adding more resources (e.g. CPU, memory, disk) to a single server. MongoDB is horizontally scalable, meaning it can handle more load by adding more servers to a cluster.
4. Use case: SQL databases are typically used for transactional workloads where data consistency and integrity are of paramount importance. MongoDB is more suitable for non-relational data and high-performance, high-availability systems.

In summary, while SQL and MongoDB are both powerful database management systems, they are designed for different use cases and have different strengths and weaknesses. SQL is best suited for transactional workloads with strong data consistency and integrity requirements, while MongoDB is better for high-performance, high-availability systems with non-relational data.

Organizations that require a more generic solution and have relatively small database needs often choose MySQL and other full-scale databases. MySQL lacks speed, and developers face difficulties when the data volume is large or when the schema is unstructured. MySQL is widely accepted for its performance, reliable data protection and high availability case.

When your data is unstructured and complex or when you cannot accurately specify schemas, MongoDB is the best choice. MongoDB will help you overcome all your challenges with its document-based data modeling.

**Desktop Framework (**[**Quarkus**](https://quarkus.io/) **vs** [**Spring Boot**](https://spring.io/projects/spring-boot)**)**

**Spring Boot (Java Framework)**

**Pros**

* Good for quick development
* Opinionated approach (Has specific ways of doing things that focus on ease of use and easier to find documentation.
* For Java language
* Handles boilerplate code
* Code runs immediately without configuring XML
* Includes embedded servers

**Cons**

* Opiniated approach (Locked into a certain way of doing things and can’t experiment with additional options)
* Some bloat packaged in (larger file sizes)
* Experimental native imaging using other software

**Qwarkus (Java Framework)**

* Provides Native images (platform specific executables)
* Qwarkus supports all of Spring Boot and can import Spring Boot projects
* Lighter in weight than Spring Boot

**Node.js (JavaScript backend)**

**Pros**

* Faster coding time than Java
  + JavaScript is typically easier to create initial prototypes in compared to Java
* Better UI tools than Java leveraging HTML and CSS
  + Using HTML and CSS will allow us to reduce cross-platform UI problems and create UI which will work on any device with relative ease.
* Lighter weight / more built-in tools which are relevant to the project than Java
* Access to a large amount of pre-built libraries using npm

**Cons**

* Features and documentation change frequently
* Dev team would need to learn the Node and JavaScript eco-systems if they do not already know it

Quarkus is a more modern and lightweight framework that is optimized for cloud-native deployment, while Spring Boot is a more established and feature-rich framework that is more geared towards traditional Java development. Node.js would allow for the backend to be written in JavaScript, which if chosen with React Native Expo would allow for the entire project to be written in JavaScript.

**Chosen Desktop Backend: NodeJS**

NodeJS will be our chosen backend due to it also being in the JavaScript language, this will allow the entire project to be in JavaScript and allowing for any knowledge learn to apply to the whole project. One potential downside of NodeJS is the lack of developer-facing asynchronous threading. In a traditional language such as C++, threads would be controlled by the developer. In NodeJS, there are hidden multi-threaded parts of the virtual machine, such as IO operations which the developer can "await" to be finished. However, unlike in a language such as C++, developers are more or less safe from many of the common pitfalls of lower-level languages, such as thread-safety and common memory leaks. Safety and reliability for the backend is crucial, as a loss of the backend will result in the entire project going offline and unusable.

Another strength of NodeJS is the high amount of backend-specific frameworks and libraries. Relevant ones to the project include the MariaDB Node.js Connector, which allows for the connection of NodeJS to MariaDB, our chosen database for the project. Have easy, official, and well-documented connection to our database will be crucial to the projects functionality. Utilizing the Node Package Manager, third-party open-source libraries can be easy utilized and built upon for the functionality of the project. Java, another option considered for this role, has similar support for third-party libraries, typically through build systems such as Gradle and Maven, which can be complicated to setup, maintain, and rapidly prototype, a benefit for this project as many of the team members will need to learn these technologies. While Python is quite comparable to JavaScript in terms of capability, having the project in one, common language will benefit from many learned techniques from the mobile app being applicable to the backend.

While the team will need to learn this technology, which wouldn't be a requirement for a language such as Java, the primary advantages of an easier build system compared to Java, a unified language for the entire project, easy third-party library support compared to Java with explicit official support for MariaDB, and better memory / thread safety compared to C++ make JavaScript our technology of choice for the backend.