1. Analysis of alternatives

1.1. Summary

This Analysis of Alternatives is a description of what this team has chosen for the following:

- Backend Programming Language
- Frontend Framework
- Database Management System (DBMS)

And the team's choices for each were:

Decisions	Choices
Backend Programming Language	Python, JS, TypeScript, PHP
Frontend Style Framework	Bootstrap, Tailwind, Bulma, Foundation
Frontend Framework (Structure)	React JS, Svelte, Angular, Flutter
Database Management System (DBMS)	Firebase Cloud, MySQL, MongoDB, Redis

Each and every choice was analysed with a set of factors, and graded them from 0 - 5 (unless the factor is talking about a type -like SQL or NoSQL for example-). The factors at play were:

Backend Languages:

- Teams Expertise of the language
- Compatibility with Frontend Framework
- Compatibility with Database Management Systems

Frontend Frameworks (Presentation):

- Difficulty of Implementation
- Difficulty of Integration

Frontend Frameworks (Structure):

- Difficulty of Implementation
- Difficulty of learning the language

Database Management System:

- SQL or NoSQL
- Ease of Use and Integration

After comparing all our choices with the factors shown, the decision was to pick the following:

Decisions	Choices
Backend Programming Language	TS
Frontend Library(Presentation)	Bootstrap
Frontend Framework (Structure)	React JS
Database Management System (DBMS)	Firebase

We opted for TypeScript as our backend programming language because it offers the power of JavaScript with added features such as type casting and extensive support. While Python was considered, we found it unsuitable for web development, and PHP was not preferred due to lack of team experience. Therefore, TypeScript emerged as our ultimate choice.

Bootstrap was picked over the presentation libraries due to it being overall simplicity yet powerful where as other libraries required a more complex setup

ReactJS was chosen due to…[Aditti complete this pls]

Firebase was chosen due to…[Chua complete this aight?]

1.2. Analysis of possible options

1.2.1. Backend Programming/Framework

All of the factors are graded on a scale of 0 - 5, where 0 is least fulfilling of the factor and 5 is most.

	Teams expertise with the syntax	**Compatibility with Front-End Framework**	**Compatibility with Database Management System**
Python (Flask)	5/5	4/5	3/5
JavaScript (Node.JS)	4/5	5/5	4/5
TypeScript (NestJS)	4/5	5/5	4/5
PHP (Laravel)	1/5	5/5	1.5/5

Teams expertise with the syntax

Python:

- The team is quite familiar with Python due to its involvement in many units throughout our degree, such as FIT2004, ENG1013, etc.

Javascript:

- The team has a good understanding of JavaScript syntax due to their use of it throughout their personal projects

TypeScript:

- Even though the team has barely any experience with using it, they have a good amount of knowledge on TypeScript syntax as it is not so different from Javascript with the main difference being the addition of type casting

Php:

- This team doesn't have much experience with Php, due to it being less popular compared to other options

Compatibility with Front-End Framework

Python(Flask):

- Flask simplifies the creation of RESTful APIs, facilitating communication between front-end and back-end systems through HTTP requests.
- Developers can dynamically generate HTML templates with Flask, enabling front-end frameworks to enhance user interfaces while maintaining separation of concerns.

JavaScript(Node.JS):

- Node.js and front-end frameworks both use JavaScript, enabling code sharing and reducing context switching between the front end and back end.
- Its asynchronous programming model and real-time communication support make Node.js ideal for building scalable and interactive web applications.

TypeScript(Nest.JS):

- Nest.js follows a modular structure, facilitating code organisation and reusability, which aligns well with the component-based nature of front-end frameworks.
- Nest.js provides robust support for building RESTful APIs and WebSocket communication, allowing seamless integration with front-end applications for data exchange and real-time interactions.

Php(Laravel):

- Laravel offers strong support for building RESTful APIs, allowing easy consumption by front-end frameworks for seamless integration.

- Its Blade templating engine facilitates dynamic and reusable views, complementing front-end frameworks' handling of dynamic rendering and user interactions.

Compatibility with Database Management System (DBMS)

Python(Flask):

- Flask's flexibility allows seamless integration with ORMs like SQLAlchemy and direct database libraries, enabling compatibility with various database systems.
- Flask's compatibility with databases in large-scale apps can be challenging due to complexity, ORM overhead, limited access to DB features, and integration learning curve.

JavaScript(Node.JS):

- Node.js boasts extensive compatibility with DBMS, leveraging its asynchronous architecture and diverse ecosystem of database drivers and libraries.
- Challenges may include handling asynchronous code, scalability concerns, potential performance bottlenecks with connection management, and compatibility limitations with less common databases.

TypeScript(NestJS):

- NestJS offers strong compatibility with DBMS, seamlessly integrating with various databases like MySQL, PostgreSQL, MongoDB, etc., through its support for TypeORM, Sequelize, or Mongoose.
- NestJS's compatibility with DBMS may face challenges such as configuration complexity, limitations in certain features, and dependencies on external libraries for specific integrations, requiring careful consideration and management by developers.

Php(Laravel):

- Laravel seamlessly integrates with various DBMS including MySQL, PostgreSQL, SQLite, SQL Server, and MongoDB through its Eloquent ORM system.
- Potential performance issues and scalability limitations may arise due to ORM overhead and the framework's overhead, especially in large-scale applications with complex database interactions.

1.2.2. Front - End Style Library

	Difficulty of Implementation (1 = easy; 5 = hard)	Difficulty of Integration (1 = easy; 5 = hard)
Bootstrap	2/5	1/5
Tailwind	4/5	3/5
Bulma	3/5	3/5
Foundation	4/5	4/5

Difficulty of implementation:

Bootstrap:

- Bootstrap is known for its relatively easy implementation due to its extensive documentation and large community support. Its predefined CSS classes and components make it straightforward to use, especially for beginners.
- However, customising Bootstrap beyond its default styles might require some CSS knowledge.

Tailwind:

- Tailwind CSS requires a different approach compared to traditional CSS frameworks like Bootstrap. It's more verbose and relies heavily on utility classes, which can make implementation initially seem complex.
- However, once you understand the utility classes and configuration, implementing designs with Tailwind can become efficient.

Bulma:

- Bulma offers a straightforward implementation process with a simple and intuitive syntax.
- It provides a flexible grid system and predefined components, making it easy to get started without extensive CSS knowledge.
- Customization is relatively easy with Bulma due to its modular architecture.

Foundation:

- Foundation's implementation can be a bit more involved compared to some other frameworks, especially for beginners.
- It offers a comprehensive set of components and customization options, which may require more time to grasp fully.
- Foundation's documentation is robust but may require more effort to navigate for those new to the framework.

Ease of Integration:

Bootstrap:

- Bootstrap integrates well with various front-end frameworks and libraries.

- It has extensive compatibility with popular JavaScript libraries and frameworks like jQuery and Angular, making integration relatively seamless.
- Many third-party tools and plugins are specifically developed to work with Bootstrap, enhancing its integration capabilities.

Tailwind:

- Tailwind can be integrated into different projects easily due to its utility-first approach.
- It doesn't impose any particular JavaScript framework, allowing it to be used with virtually any Front - End stack.
- However, integrating Tailwind may require some adjustment in workflow and build processes compared to traditional CSS frameworks.

Bulma:

- Bulma integrates smoothly with various Front - End libraries and frameworks.
- Its modular structure allows for selective inclusion of components, reducing bloat and facilitating integration.
- While not as widely supported as Bootstrap, Bulma still enjoys good compatibility with many Front - End tools and libraries.

Foundation:

- Foundation offers decent integration options but may not be as extensively supported as Bootstrap or Bulma.
- It provides integration guides for popular JavaScript frameworks like React and Angular, facilitating smoother integration with these technologies.
- However, compared to Bootstrap, Foundation may require more effort to integrate seamlessly with certain Front - End stacks.

1.2.3. Front - End Framework

	Difficulty of implementation	**Difficulty of learning**
ReactJS	3/5	3/5
Svelte	3.5/5	3/5
Flutter	4/5	3/5
Angular.js	4/5	4/5

Difficulty of Implementation:

ReactJS:

- ReactJS tends to have a moderate difficulty of implementation. It provides a component-based architecture, which can streamline development, especially for large-scale applications. However, it might require setting up additional tools, which can add complexity for beginners.

Svelte:

- Svelte is known for its simplicity in implementation. Its compiler-based approach generates highly optimised code, resulting in a smaller bundle size and better performance. This streamlined approach often makes implementation relatively easy compared to other frameworks.

Flutter:

- Flutter offers a moderate difficulty of implementation. While it provides a comprehensive set of widgets and tools for building cross-platform mobile applications, developers need to learn Dart, the programming language used with Flutter. Once acquainted with Dart, Flutter's reactive framework can simplify the implementation of UI components.

Angular.js:

- Angular.js, being an older version of Angular, can have a higher difficulty of implementation compared to modern frameworks like React or Vue. Its opinionated structure and steep learning curve might pose challenges, especially for beginners. However, once understood, Angular.js can offer robust features for building complex web applications.

1.3.3.2. Difficulty of Learning:

ReactJS:

- ReactJS typically has a moderate difficulty of learning. Its component-based architecture and JSX syntax might be unfamiliar to developers coming from traditional HTML and JavaScript backgrounds. However, React's extensive documentation, large community, and abundance of learning resources make the learning curve manageable.

Svelte:

- Svelte is often considered relatively easy to learn compared to other frameworks. Its intuitive syntax, compiler-based approach, and minimal boilerplate code make it accessible to developers with varying levels of experience. Additionally, Svelte’s documentation and tutorials provide comprehensive guidance for newcomers.

Flutter:

- Flutter may have a moderate difficulty of learning, primarily due to the need to learn Dart alongside the framework. However, Dart's syntax is familiar to developers with experience in languages like JavaScript or Java. Flutter's comprehensive documentation, official tutorials, and active community contribute to easing the learning process.

Angular.js:

- Angular.js can have a higher difficulty of learning compared to some other options. Its extensive feature set requires developers to grasp complex concepts. Additionally, the framework's reliance on TypeScript and its opinionated structure might pose challenges for beginners. However, thorough study of Angular.js documentation and tutorials can help developers overcome these hurdles.

1.2.4. Database Management System

For the factor of SQL or NoSQL will only have the possible answer of “SQL” or “NoSQL’

Options	SQL or NoSQL	Ease of use and integration
MySQL	SQL	2/5
Oracle	SQL	1/5
Firebase Cloud	NoSQL	4/5
Redis	NoSQL	3/5
MongoDB	NoSQL	4/5

Strengths and weaknesses of each solution:

MySQL (SQL):

Strength:

- Reliable and ACID compliance, can ensure data integrity which will play a crucial role for managing credentials.
- MySQL is quite straightforward to set up and use, especially for those who are familiar with SQL already.
- Offers comprehensive documentation and community support which can be found online making it easier to implement.
- Suitable for storing game scores
- Excellent at supporting complex queries for detailed leaderboard statistics.

Weakness:

- Cannot be scales horizontally as it is really challenging and costly compared to noSQL options.

Oracle (SQL):

Strength:

- Security, backup and data recovery which are very important for managing the credentials of the user.

- Performs well for complex queries and transactions.
- Scalable both vertically and horizontally.

Weakness:

- More costly.
- Hard to pick up how to use, manage and optimise the database using oracle.

Firebase Cloud (noSQL):

Strength:

- Provides real time updates to data which will be beneficial for creating the leaderboard.
- Very developer friendly and it is relatively easy to be picked up.
- Provides built in authentication making it easier to manage user credentials and game sessions

Weakness:

- Costly if the game scales.

Redis (noSQL):

Strength:

- Really easy to implement as it just holds key-value which makes it really easy to implement for the leaderboard scoring.
- Fastest at reading and write operations and is perfect for real time leaderboard.
- Can be used alongside another database to manage user credentials more securely but this will make it harder to implement.

Weakness:

- Not ideal for any credential storage as it is primarily designed for high-performance and in memory storage.

MongoDB (noSQL):

Strength:

- Offers a developer friendly environment.
- Straightforward to store complex game data and user credentials.
- Highly scalable which makes it a really good choice when handling large volumes of data and traffic.
- Allows the data structure to evolve over time without significant downtime but it is not as efficient as the other choices.

Weakness:

- Can't be as efficient as SQL databases when executing complex queries.

After much consideration to either use SQL database or noSQL database, we have decided to not use SQL database because although it is reliable and has consistent data integrity and it also supports complex queries which we will not necessarily need in this project as we will only need to store the credentials and the leaderboard for now. So we decided to use a noSQL database as we'll

only need to store the scores of the players and output it into a leaderboard which does not require any complex queries.

As for the choices for noSQL databases, both mongoDB and Firebase Cloud Firestore are excellent for their ease of use, scalability and real time capabilities. As for redis, although it is way faster than the other two databases for reading and writing data, it does not have any built-in authentication. Hence, we are left with either mongoDB or Firebase Cloud which are both easy to learn and use. We decided to choose Firebase Cloud over mongoDB because everyone has more experience using Firebase Cloud.