Project Architecture

VocabVersus

Thomas van der Molen

|  |  |
| --- | --- |
| **Project Information** | |
| Project members | Thomas van der Molen |
| Project Name | VocabVersus |
| Version | 1.0 |

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Change** |
| 1.0 | 05-04-2023 | Moved architecture documentation from Project Plan |

Table of Contents

[Introdocution 3](#_Toc131600762)

[Considerations 3](#_Toc131600763)

[Architectural Exploration 4](#_Toc131600764)

[Game 4](#_Toc131600765)

[Data 4](#_Toc131600766)

[User 4](#_Toc131600767)

[Architecture 5](#_Toc131600768)

[Diagrams 5](#_Toc131600769)

[Front-end 6](#_Toc131600770)

[Back-end 7](#_Toc131600771)

# Introduction

This architecture document is used as a broad explanation of the architecture and related technologies that have been chosen for the VocabVersus application.

This document is a continuation of the Project Plan, which is recommended to read first.

# Considerations

As shortly discussed in the [Target Audience second of the Project Plan](Project%20Plan.docx#Target_Audience), there are several considerations that will be made to create an optimal application, these considerations will be explored as architectural decisions.

After the considerations based on architectural decisions have been made, an architecture for the application can be created, with technologies aligning with the needs formed from these decisions.

Below is a list of subjects that are of interest during this project and will have to be taken into consideration when creating the application architecture.

|  |  |
| --- | --- |
| **Considerations** | |
| **CG** | **Game** |
| CG-1 | Game-User connection |
| CG-2 | Game latency |
| CG-3 | Game events |
| CG-4 | Game instance load |
| **CD** | **Data** |
| CD-1 | Data management |
| CD-2 | Data privacy |
| CD-3 | Game data aggregation |
| CD-4 | Game event persistence |
| **CU** | **User** |
| CU-1 | Webpage design |
| CU-2 | Player engagement |

## Architectural Exploration

### Game

Players will generally only be playing a single instance of a game at once with other players, this allows player connection to be linked directly with the single game instance that is relevant to them (CG-1). By grouping player connections, instance specific events can much more easily be distributed to the relevant players (CG-2), and it allows the game load to be distributed among smaller game instances, allowing for game instances to be scaled up based on the amount of impact a game instance might have (e.g. a 4 player game vs a 100 player game) (CG-4).

During the game, several players will be playing against one-another in real-time, this makes it extremely important to reduce the latency for actions taken by players that will affect the game state of other players.

Thankfully, players will only know of other player’s actions via events inside the game, due to this a larger connection delay margin can be acceptable as long as it is not disruptive to the game’s flow, for example; slight delay margins can be added to event handling allowing player actions to still be evaluated even if it is technically outside of the relevant game state time (e.g. answers submitted at the end of a round will still be evaluated if it is within the connection delay margin) (CG-4).

### Data

A lot of persistent data from different domains should be stored for the web-application, this includes account information, game instance information and general data used for game instances. Game instance data will often have to be stored temporarily and deleted later, while aggregated game data will be mostly static after being added, with frequent evaluations being made based on the stored data (CD-1).

The data used by the game will be aggregated either by system administrators and users alike, allowing for new data sets to be created for different games, extra care will have to be taken by allowing users to input their own data (CD-3).

Some data, such as the inputs made by users during a game, or personal information such as passwords, have to be stored securely (e.g. properly hashing passwords) (CD-2).

Situations might come up where a game instance might be removed, causing any non-persistent data to be lost (e.g. connection loss causing all players to disconnect), to ensure that players are able to continue their games, important game state data should be stored in a short-term persistent manner and be able to happen asynchronously from the progress of the game (CD-4).

### User

The game is expected to be played on many different devices, such as phones, tablets and computers, this makes it important that the user interface for the game can support these different screens well to improve user experience (CU-1).

Furthermore, to help keep player engagement high, it should be made as easy as possible for users to access core features of the web-application such as joining or creating a game, and to help keep players engaged in playing a more competitive mode should be created with a public leaderboard, allowing players who prefer more challenge to also be engaged (CU-2).

# Architecture

## Graphical user interface, text, application Description automatically generatedDiagrams

Graphical user interface, diagram

Description automatically generated

### Front-end

The front-end will be contain the visual interfaces the user will interact with, these have been separated into two separate parts, the dashboard that serves as the landing page and the game creation screen, and the game itself.

#### Dashboard

The dashboard will mostly contain static information such as buttons to join/create a game and a leaderboard, the dashboard will be the entry point for most users and thus will be important to have fast load times and good SEO.

For the dashboard, the [Astro](https://astro.build/) framework will be used, this framework is purpose built to deliver fast page load times, and allows for dynamically loading in javascript components from the server side renderer, this allows the pages to have fast load times and be very SSO friendly. Astro is a relatively new framework, and I would like to learn more about it as the concepts it provides seem to be very useful.

#### Game Interface

The game interface will be a graphical interface used to inform the user on the current game state, as well as allowing the user to make the necessary commands for the game, this game interface will be rapidly changing based on the game state and will need to communicate in real-time with the game’s logic.

Due to the requirement for the game interface to be rapidly changing, a good choice for technology will be [React](https://reactjs.org/), as it supports fast document changes with the “reactive” virtual DOM, and has been proven as a professional good javascript framework, as seen with it’s popularity ratings on [the 2022 stackoverflow survey](https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe)

### Back-end

The initial scope for the project will contain a game service where the game instances are created and handled and a data aggregation service, which will contain data used in a game (such as word sets) that will be evaluated very quickly, alongside a more generic Game Storage where gameplay behavior such as word use is stored after a game, and is able to be obtained later for use in game balancing.

#### Game Engine

The game engine will be a back-end server that handles the game logic and the connections made for specific game instances, this game engine will have to be able to maintain real-time connections with several users and support fast event processing (taking user input and processing it into a game action).

Due to the real-time nature of the connection required between the game engine and the game interface, [Signal-R](https://dotnet.microsoft.com/en-us/apps/aspnet/signalr) will be used, as this allows for fast and reliable real-time connections between server and client and allows for [native grouping and scaling of connections](https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-7.0) into game instances, more research into low latency Client-Server communication has previously been done and can be found [here](https://github.com/Thomas-Molen/WebAdventure/blob/main/Documentation/Low-LatencyClient-ServerCommunication.pdf).

As Signal-R is chosen for the server-client communication technology, the ASP.NET framework will be used, as Signal-R is part of this framework as well as having a large professional user base and support, for example in [this survey for frameworks](https://survey.stackoverflow.co/2022/#section-most-popular-technologies-other-frameworks-and-libraries).

#### Data Aggregation

For data aggregation a data storage will be used, that is able to store large amounts of small size records of datasets and is able to be interacted with frequently by several connections and is able to perform basic evaluations.

The primary evaluation that has to be done on the data will be record matching, here it is only important for the data storage to be able to quickly evaluate if a given record value is already in the database, as this will be used to check user input against a words list.

For handling these text searches, a standard database might not be sufficient, as most SQL/NoSQL evaluations can be relatively slow when having to search through many documents, for this [Apache Lucene](https://lucene.apache.org/) will be used as it can handle large amounts of data and is specialized for text search.

Lucene supports many different version for specific environments, the version that will be used during this project will be [Lucene.net](https://lucenenet.apache.org/), as it allows for the Data Aggregation service to be using a similar framework as the Game Engine, which should allow for easier integration and the .NET framework has straight-forward and robust systems for creating web API endpoints with ASP.NET or possibly using the real time Signal-R technology already researched for the [Game Engine](#_Game_Engine).

#### Game Storage

The game storage will be a large database that contains many records for the behavior of users, here one large database insert will be done at the end of game where all events are stored (e.g. all words used to gain points), these records can later be re-obtained to use for game balancing (common words will become worth less points).

As this database will contain many records a [MySQL database](https://www.mysql.com/) will be used, as this is a relatively affordable database technology (for cloud storage) and is proven to be scalable for rapid data entry/requesting for example with [twitter](https://scaleyourapp.com/what-database-does-twitter-use-a-deep-dive/#:~:text=MySQL%2C%20PostgreSQL%2C%20Vertica-,MySQL,millions%20of%20queries%20per%20second.).