# Prompt Party: AI Art Game Project

**System Documentation**

**Brian Meder**

**Professor Jackowitz**

**2 May 2025**

**Submitted in partial fulfillment**

**of the requirements of**

**CMPS (IT) 490 -- Capstone Project**

# Table of Contents

Table of Contents 1

Extended Abstract 7

Justification and Feasibility 10

Requirements Specification 12

Requirements Abstraction 12

Requirements Main Body (w/ subsections) 12

1. Introduction. 12

2. Features and Functionalities. 12

2.1 Accessibility: 12

2.2 Login: 13

2.3 Hosting/Joining Games: 13

2.4 Word Bank Entries: 13

2.5 Prompt/Image Generation: 13

2.6 Prompt/Image Guessing: 13

2.7 Scoring: 14

3. User Assumptions and Expectations. 14

3.1 Webpage Familiarization: 14

3.2 AI-Art Familiarization: 14

4. System Requirements. 14
4.1 Hardware: 14
4.2 Software: 15
System Design 16
System Design Abstraction/Introduction 16
***Structure and Relation Diagram/Graph*** 16
Levels 16
1. Frontend 16
1.1 Frontend Architectural Design 16
A.1 Subpart - App.js: 17
A.2 Abstract Specification for App.js: 17
A.3 Interface Design for App.js: 17
A.4 Component Design for App.js 17
B.1 Subpart - index.js 17
B.2 Abstract Specification for index.js 17
B.3 Interface Design for index.js 18
B.4 Component Design for index.js 18
C.1 Subpart - firebase.js 18
C.2 Abstract Specification for firebase.js 18
C.3 Interface Design for firebase.js 18

C.4 Component Design for firebase.js 19
D.1 Subpart - socket.js 19
D.2 Abstract Specification for socket.js 19
D.3 Interface Design for socket.js 19
D.4 Component Design for socket.js 19
E.1 Subpart - Component Files 19
E.2 Abstract Specification for component files 19
E.3 Interface Design for component files 20
E.4 Component Design for Component Files 20
F.1 Subpart - Miscellaneous/Various Libraries 20
F.2 Abstract Specification for Miscellaneous/Various Libraries 20
F.3 Interface Design for Miscellaneous/Various Libraries 20
F.4 Component Design for Miscellaneous/Various Libraries 21
2. Backend 21
2.1 Backend Architectural Design 21
A.1 Subpart - server.js: 21
A.2 Abstract Specification for server.js: 21
A.3 Interface Design for server.js: 21
A.4 Component Design for server.js 22
B.1 Subpart - firebaseAdmin.js 22

B.2 Abstract Specification for firebaseAdmin.js 22

B.3 Interface Design for firebaseAdmin.js 22

B.4 Component Design for firebaseAdmin.js 23

C.1 Subpart - Handler Function Files 23

C.2 Abstract Specification for Handler Function Files 23

C.3 Interface Design for Handler function Files 23

C.4 Component Design for Handler function Files 24

D.1 Subpart - Miscellaneous/Various Libraries 24

D.2 Abstract Specification for Miscellaneous/Various Libraries 24

D.3 Interface Design for Miscellaneous/Various Libraries 24

D.4 Component Design for Miscellaneous/Various Libraries 24

3. User Interface Design 25

3.1 Design Principles: 25

3.2 Examples: 25

4. Help System Design 25

4.1 Implementation: 25

Testing Design 27

Abstract 27

Introduction 27

A. Unit Testing 27

B. Module Testing 28

C. Subsystem Testing 29

D. System Testing, Validate Requirements 30

E. Acceptance Testing 31

Testing 31

A. List tested items organized by phases. 31

I. Front-end Testing 32

II. Back-end Testing 32

B. Requirements Traceability. 33

C. Testing Schedule 35

D. Test Recording Procedures 35

E. Hardware and software requirements 36

User Manual 38

Abstract 38

Introduction 38

Login Functionality/Reason 38

Matchmaking Functionality/Reason 39

Host Functionality/Reason 39

Word Bank Functionality/Reason 39

Prompt Creation Functionality/Reason 40

Guessing Functionality/Reason 40

Score Screen Rank Functionality/Reason 41

Global Leaderboard Functionality/Reason 41

Introductory Manual 41

a. How to Use the System 41

b. Information on the Help System 43

System Reference Manual 44

a. Listing of Services 44

b. Error Recovery 44

User Manual Glossary 46

Project Narrative Section 47

System Maintenance Guide 50

Running the App 50

Known Problems 50

Hardware and Software Dependencies 51

Product Evolution 51

Index 52

# Extended Abstract

## Introduction:

The entertaining nature of party games are the main inspiration for this project. Some main inspirations include the games, Kahoot and Jackbox, respectively. In games like these, ease of access, replayability and adaptability are the biggest components. As such, the project draws inspiration from these sources to ensure that it provides users with a player experience that can be easily approached, meaning both easy to learn and easy to access (free to play through choice internet browsers). The project also looks to offer replayability through allowing users to have direct control over key game aspects. This ensures each game will always play differently, as different players will always employ different tactics when playing.

## Context:

This project is a multiplayer game that players will interact with through an internet browser of their choice. Matches of this game will provide a competitive atmosphere where players fight for the most points by continuously answering questions correctly. The player with the most points at the conclusion of a match will be crowned as the winner. These winners will have their usernames and scores saved to a leaderboard thanks to an implemented database.

This game will utilize an AI image generating API in which users will attempt to guess the keywords used to generate an image. At the beginning of a match, players will input a given amount of words in the time allotted. Each player's chosen words will be mixed together and given to a chosen player whose current turn it is. Alongside these user words, other randomly provided nouns and art styles from a preset list are provided. This chosen player will then use the words given to them to create an image generation prompt before time runs out. Once this chosen player submits this prompt, an image is generated by the AI api using the newly created prompt. This AI generated image will then be shown to the other players in the lobby. These other players will now have the opportunity to guess the words that were used to create the image being presented to them. They will do so by utilizing the complete list of words that were available to the chosen player who generated the image. Guessing players will receive points for getting keywords correctly. Players will also receive bonus points for guessing the prompt’s keywords in the correct order they were placed in the generative prompt.

A game like this not only provides a competitive atmosphere for users, but also proves to be a fun and enjoyable experience due to the outlandish and randomized nature of AI generated images. However, to ensure the fun atmosphere is maintained and in order to prevent abuse of this system, a special character remover API and profanity censor API are to be implemented in order to ensure user input words are not inappropriate or nonsensical. The profanity API will also be applied to the chatroom available to players.

The UI of the webpage will be colorful and exciting to retain player attention. Players will have their usernames, and points presented to the whole lobby. This provides users with the incentive to do well, as they may show off their high scores and rewards to others.

## Overview:

A JavaScript browser game that supports many users playing at the same time. It will use a database to keep track of player scores, their usernames and their passwords. It will contain an implemented chat room so players may speak to each other during the games. The project involves front-end and back-end development, concurrency programming, using a database and some basic level of encryption for holding the user passwords. HTML, CSS, JavaScript, React, Socket.io library for JavaScript and MySQL for the database. The project will utilize multiple

APIs, such as an AI art generating API, a spell check API and a profanity censor API. In this case, the artificial intelligence will be pre-trained and therefore training the AI is not a necessary step. The project requires a server to host multiple game lobbies at once. This server is also needed to host the database storing player information.

# Justification and Feasibility

## **Justification:**

Browser-based party games are some of the most fun you can have online with friends. Games like Kahoot, JackboxTV, Gartic Phone and Skribbl, are all examples of these fun browser-based party games. These games maintain their popularity even though they may lack state-of-the-art graphics and may even be a decade old, and this is for several reasons. They do not require expensive computer hardware, they are usually free and they are easily accessible by players. All that is needed to play is a device that can connect to the internet and may access any of the common internet browsers compatible with Javascript. These include the most popular browsers on devices like Chrome, Safari, Firefox and Edge. In our contemporary times, the chance a person has a device capable of accessing these browsers is extremely high. Therefore, the barrier to entry for these games is very low and subsequently very accessible.

These browser games are also very social and replayable. In each of them, players are given creative control of how the games are played, or what content they are presented with in each game. This dependency on player creativity and player choice makes each match completely unique. These games may be played often due the variability and uniqueness of each match which keeps the game fresh and exciting for players.

The justification for another browser-based party game such as these is to supply an easily accessible and fun experience to players all over the world. The usage of the AI art program within the project necessitates further justification, as it aims to reduce AI art stigma. Not necessarily to normalize the replacement of real human artists with AI generative art, but to take a source of negativity and create something positive from it. It is no lie that AI art will be an ever-increasing factor in our lives going forward, so it is better to have a way to view its positive impact rather than focusing on the negative.

…

The game outlined in this project, *Prompt Party,* aims to deliver a highly accessible, fun, replayable and memorable experience to players everywhere while simultaneously creating a source of positive reception to generative AI art programs.

…

## **Feasibility:**

The game outlined, *Prompt Party*, will utilize a robust tech stack that involves many of the most popular app development technologies in the industry today. With a realistic goal and with the use of tech stacks that have been utilized in similarly successful projects, it is known that the deliverable is indeed possible. However, there are many moving parts required to get this game in its optimal state or ready as a minimum viable product. The complexity of the project, combined with its necessity to undergo further research into topics unknown ensures the game will be of a needed complexity for a capstone project. Due to previous work with similar tech stacks, it is likely the game will reach the minimum viable product before the end of the Spring 2025 semester. If time does not permit, certain game features may have to be cut, but the game will be in a playable state by May 2025, even if it lacks some certain features unnecessary to the core gameplay.

# Requirements Specification

## Requirements Abstraction

The following chapter outlines the requirement specifications for the project titled, Prompt Party. This project is an AI art game in which users compete to guess user generative prompts supplied to an AI art model. The following sections will cover a brief introduction of the project, features and functionalities, user assumptions and expectations, and lastly, system requirements.

## Requirements Main Body (w/ subsections)

### 1. Introduction.

Prompt Party is a competitive online party game that supports up to four players per match. As such, users can expect a fun atmosphere in which they, along with their friends, create lasting memories through a free and easily accessible videogame. The game revolves around users guessing the prompts their competitors used to create an AI generated image.

The game begins with all players populating a single unified word bank at the beginning of the match. Following this step, each player will get a turn in which they are allowed to create a generative prompt using this created word bank. These prompts get supplied to an AI model that will generate a corresponding image unique to each prompt. Users will then attempt to guess the prompts used to generate those images using the preexisting word bank they created.

### 2. Features and Functionalities.

#### 2.1 Accessibility:

Prompt Party will be easily accessible via the most popular web browsers on the internet. Developed specifically for, but not limited to Chrome, Firefox and Edge, users can expect to access the game through any or all of these browsers. Upon navigation to the page, users will be greeted with a friendly user-interface that is both intuitive and in no way impedes player enjoyment or execution of the game.

### 2.2 Login:

The webpage provides users with the ability to securely login with their Google account (...@gmail.com) or to play as a guest. Logging in with Google saves player data, like a chosen username/gamertag that will remain consistent amongst all matches. Logging in with Google also provides users with a safe two-factor authentication provided by Google services that keeps account login secure. On the other hand, when choosing to play as a guest, users forgo the option of having their player data saved in order to gain access to the game without having to login to their emails.

### 2.3 Hosting/Joining Games:

After logging in, users can host their own lobby or join a preexisting lobby. When hosting a lobby, a special six character lobby ID code is generated. Other players will utilize this code to join the host's game. Lobbies support up to four players. The host can choose to start the game whenever, so long as there's at least two players in the lobby.

### 2.4 Word Bank Entries:

Within the lobby, players will get a minute to input single words into a word bank that is shared amongst all players. There is slight enforcement on the input of these words, ensuring the censoring of profanity and ensuring the input of a single word per entry, not a sentence.

### 2.5 Prompt/Image Generation:

Within the game, when it is a player’s turn, they will be presented with a screen of the previously created word bank. The chosen player will now be able to create a prompt by handpicking up to ten words from the bank. This newly created prompt is then run through a special artificial intelligence model and the user will be presented with the image result. If they like the image, they may submit it to be used in the game. If they dislike it, they may have the option to generate the image again or choose a new prompt.

### 2.6 Prompt/Image Guessing:

Other players will be presented by the newly generated image without the prompt used to generate said-image. These other players will be presented with the word bank, as well as ten empty spaces along with a submit button. These players will click single words from the wordbank to populate the ten empty spaces, this is their guess of what prompt has been utilized. The prompt may be less than ten words and never more than ten words. When these players have finalized their guesses, they may hit the submit button to further the game. Players have 30 seconds to make these guesses.

### 2.7 Scoring:

After the player's guesses are gathered, they are then scored for accuracy. Players who correctly guess words that preside somewhere within the prompt will receive 10 points per word. Players who correctly guess words in the correct order corresponding to where the words preside within the prompt will receive 15 points per word. Users who guess the entire prompt correctly (meaning both word choice and in the correct order) will receive 15 points per word, plus a bonus 50 points. The player with the highest score after each player has had a chance to generate at least one image will be crowned the winner.

## 3. User Assumptions and Expectations.

### 3.1 Webpage Familiarization:

It will be expected of users of Prompt Party to have some similarity with accessing websites through a web browser. They will also be expected to understand a simple login process (commonly used Google login process used on many websites) and to understand the ramifications of choosing to play as a guest rather than logging in.

### 3.2 AI-Art Familiarization:

Users will also be expected to understand the somewhat outlandish and confusing results created by AI image creation software, which contributes to the fun and difficult aspect of the game. If users are too unfamiliar with this concept, they may be too caught up in the fact that the generated image is not a one-to-one realistic recreation of what the prompt had specified. *This is supposed to be the fun part!*

## 4. System Requirements.

### 4.1 Hardware:

Users require a computer that can access the internet. They also require a stable internet connection to maintain connection to the game lobby and other players. They will require computer device peripherals like a keyboard (to input words, lobby IDs, etc.) and a mouse (to select words, navigate the web page, etc.).

## 4.2 Software:

Users will need their computers to be able to access the most popular, well-supported for internet game browsers. These browsers include, but are not limited to, Google Chrome, Microsoft Edge and Mozilla Firefox.

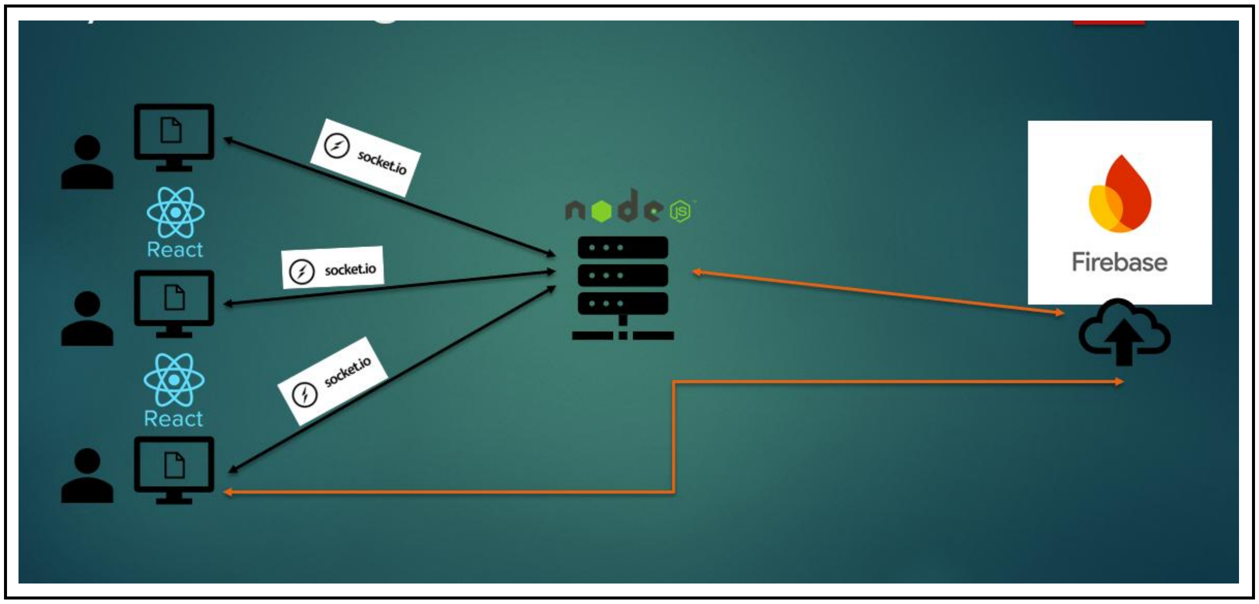
**A Logo as Imagined by ChatGPT 4.0**

# System Design

## System Design Abstraction/Introduction

The following chapter outlines the system design for the project titled, Prompt Party. This project is a browser-based AI art game that utilizes a React frontend written in JSX connected to a Node backend written in JavaScript built on the Express framework. Rather than an integrated self-hosted database, the application utilizes the Google Firebase services to provide data storage and retrieval. The following sections will cover the project's architectural design, its component's designs, data structure designs, algorithm designs, and lastly, its user interface and help system design.

***__Structure and Relation Diagram/Graph__***



This diagram displays a visual representation of the system design outlined within this document. (Brian Meder)

## Levels

### 1. Frontend

#### 1.1 Frontend Architectural Design

On the frontend, the architectural design is quite similar to other web applications. There is a client-end application accessed via web browser. This client application is a React application that utilizes a few different subparts to perform its function. These subparts are configuration files, Javascript and JSX files, CSS files, and various component files written in JSX.

### A.1 Subpart - App.js:

#### *A.2 Abstract Specification for App.js:*

The main application accessed by users is the subpart known as App.js. This is a JSX file that contains the markup for the user interface, as well as the socket emitting functions that allow users to interact with the backend. This means that this subpart is responsible for both communication and relaying information to the user. It is the most important subpart of the client view. It must operate under strict constraints that enforce proper socket emitting and receiving. If a message is not received by the backend or is not received by the frontend, the application may fall into an error state, where important information is withheld from the user or backend.

#### *A.3 Interface Design for App.js:*

This subpart interfaces with other subparts by utilizing import statements and React. When starting a React application in the command line, or by accessing it in a web browser, the file known as index.js is actually run. The App.js file is imported to this index.js file. The index.js file runs the App.js file as a component. App.js interacts with other subparts via import statements such as the use of importing the Material UI library, importing the Firebase library through the firebase.js file and various React features like hooks through the importing of the React library.

#### *A.4 Component Design for App.js*

***On application startup…***

- Begins running thanks to index.js.
- Socket.js connects the app to the backend.
- Socket utilizing functions communicate with the backend on user request.
- Firebase API functions allow users to authenticate themselves .
- Webpage dynamically displays depending on the status of React hooks.

### B.1 Subpart - index.js

#### *B.2 Abstract Specification for index.js*

Index.js is a file that all initialized React apps are given. By default, it acts as the entry point into a React application by running the components outlined within the file. In this application, Prompt Party, all the application's code lies in the App.js file. This is the only component listed to run when index.js is called at application runtime. It also contains necessary libraries such as React DOM which configure the display of the webpage. This file can also be utilized as a place for initializing global configurations, such as APIs, that you wish your whole application to be able to access.

### *B.3 Interface Design for index.js*

This file interfaces with the rest of the application via import statements. It contains an import of App.js, which it then subsequently uses to run the App.js component for the application. This is the only interaction between index.js and other documents of the frontend.

### *B.4 Component Design for index.js*

***On application startup…***

- Default imported React libraries configure the DOM.
- File acts as the root for the application App.js.
- Begin the running of the App.js file (the main application) in the root.render() function.
    - Root.render is from the default imported ReactDOM library.

## C.1 Subpart - firebase.js

### *C.2 Abstract Specification for firebase.js*

Firebase.js is an initialization file for the firebase API, specifically for the frontend users. It initializes the instance of the firebase API utilizing the special API key and other confidential information. Upon uploading to the internet, this file will be obscured by a GIT ignore file. The import statements necessary for certain Firebase functions and features needed are also imported into this file directly. These features include gaining API authorization, initializing the database, initializing the Google login provider and more.

### *C.3 Interface Design for firebase.js*

This file interfaces with the rest of the application by being imported into App.js, the main user frontend file. From here, authorization to the API is already gained. It also hides confidential authorization information from plain user view. This allows App.js to utilize the API features without having to also contain the 50 lines of code needed to initialize the API. The main reasons this file needs to exist and interface with App.js in the first place is for code cleanliness and to obscure private information.

#### *C.4 Component Design for firebase.js*

***On application startup…***

- Utilize imported Firebase libraries to initialize the Firebase API.
- Gain authorization for use in App.js by using the provided API key.
- Configure Google authorization provider for sign in features in App.js.

### D.1 Subpart - socket.js

#### *D.2 Abstract Specification for socket.js*

In a similar vein to the last subpart, socket.js exists as a separate file to initialize the client socket library in a clean way. Since this is a library rather than an API, no hidden credentials need to be obscured, this plainly keeps the initialization of the client sockets to communicate to the backend in one centralized location. This design also allows for multiple files to utilize the same socket.js file which is pre-configured to connect to the backend's address. This saves time if more files needed to access the backend must be written.

#### *D.3 Interface Design for socket.js*

This subpart interfaces with the App.js file only via import statement. The file is utilized to save space in the App.js file and to initialize the connection to the backend as previously stated. Socket.js interacts with no other subparts, however, if there were to be more subparts created, they could gain access to this file and therefore the backend via simple import statements of "import socket from ./socket.js".

#### *D.4 Component Design for socket.js*

***On application startup…***

- Initialize socket connection to the backend using the imported socket.io library.
- Export the socket so it may be imported to App.js where clients will be able to create unique socket connections to the backend.

### E.1 Subpart - Component Files

#### *E.2 Abstract Specification for component files*

In React, component files are collections of JSX code that offer a feature one may utilize on their webpage. Examples of web page components include search bars, headers, home pages and more. Within this application, there is a components folder. Within this folder is a collection of these component files. One such of these components is the wordSubmitForm component. This component is utilized by players of Prompt Party to submit words to the communal wordbank at the beginning of a game. These various components are utilized within the App.js file to aid in easy readability. They could very well be hard coded into the App.js file, but then the components would lose reusability and make the code of App.js much longer and more complex.

#### *E.3 Interface Design for component files*

The component files within the component folder interface with App.js via import statements. App.js is the only frontend file to utilize the components specifically built for this application. After the import of the components into App.js, they can then be used like HTML element references within the JSX markup section of the Application.

#### *E.4 Component Design for Component Files*

***On application startup…***

- Reside in a components folder specifically made for these files.
- Compile and export the React JSX component using React libraries.
- Utilize the imported Material UI library to offer visually pleasing components.
- Get imported into App.js for use in the web page like HTML elements.

### F.1 Subpart - Miscellaneous/Various Libraries

#### *F.2 Abstract Specification for Miscellaneous/Various Libraries*

This subpart is the conglomeration of various utilized libraries. Since they act as part of the frontend application it would be remiss not to touch upon them in some fashion. They are utilized in many of the frontend Javascript and JSX files. One example includes the Material UI library. This is used in various component files through an import statement to offer users consistent and visually pleasing user interfaces.

#### *F.3 Interface Design for Miscellaneous/Various Libraries*

Simply put, these libraries interface with the application by being imported into the App.js and component JSX files. These libraries are downloaded locally on the machine hosting the page, and are able to be accessed in their downloaded locations in the node_modules folder. From here, the application may access the large amount of functions that are offered by the libraries.

#### *F.4 Component Design for Miscellaneous/Various Libraries*

***On application startup…***

- Be downloaded to the machine hosting the webpage and visible in the node_modules folder.
- Get imported where needed to be used as needed.

## 2. Backend

### 2.1 Backend Architectural Design

The backend is designed as a Node server run by the light Express framework. It responds to user requests via socket emitting and socket receiving. Within the backend, there exist many handler functions that guide users through the sign in processes, matchmaking processes, and gaming processes. Without the backend, the frontend would be unable to offer communication between players and their games. The backend also provides access to the Firebase database, which requires admin authentication. All users will have their information processed and saved by the backend after being sent through a socket emission to the backend.

### A.1 Subpart - server.js:

#### *A.2 Abstract Specification for server.js:*

The server.js file is the main backend file that controls all the game logic, matchmaking logic and database access by the application. This is a Node server file that is prepped for hosting by the Express framework. Upon running, it stands by and listens for incoming messages from client sockets. It then processes the messages and operates accordingly. Often, processing these messages either results in the saving of data to the Firebase database, the sending of data back to the client, or both! The server file is able to operate this way by utilizing the socket backend library rather than the socket client library.

#### *A.3 Interface Design for server.js:*

This server.js file interfaces with a few different subparts of the entire application. Firstly, it interacts with the frontend App.js and communicates to it via sockets. This is done through the socket client and socket backend libraries respectively. The server.js file also interfaces with the firebaseAdmin.js file which will be touched upon later. This Firebase file is used through an import statement, and makes the utilization of the

Firebase admin API much cleaner and easier. The server.js file also interacts with many handler functions. These handler functions are also brought into the server file through import statements. By placing these helper files outside of the main server file, we are really just saving space and making the code easier to read and more scalable.

### *A.4 Component Design for server.js*

***On server startup…***

- Offer a centralized location where users will send requests through sockets.
- Handler functions written in separate files are imported and will perform work on the user's behalf, keeping the central file clean.
- FirebaseAdmin.js initialized API is imported and used to process user requests needing access to the database (read or write).
- Express, Cors, Socket and HTTP used in tandem to prep the server for hosting.

## B.1 Subpart - firebaseAdmin.js

### *B.2 Abstract Specification for firebaseAdmin.js*

The file known as firebaseAdmin.js is a Javascript file that initializes the Firebase Admin API. In action, this subpart is very similar to the previously mentioned frontend subpart, firebase.js. The main difference is that firebaseAdmin.js is exclusively used as a backend subpart that initializes the Firebase Admin API, which offers different features. One of these important features is higher level database access. Using the firebaseAdmin.js file also allows for sensitive information like API keys and service account information to be obscured from regular vision. The main purpose of the firebaseAdmin.js file is to initialize the Firebase Admin API utilizing sensitive credentials, then exporting this instance of authorization to other files (in this case server.js).

### *B.3 Interface Design for firebaseAdmin.js*

This file interfaces with server.js through an import statement. Once the API is imported into server.js, then the server can begin performing sensitive operations like storage and retrieval of player and lobby information from the database. Without this file, server.js would not be able to access the database or Firebase at all. This is because users on the front end gain authorization to use firebase by logging into Google, on the backend, there is no such login. This is why we must abstract the initialization of the admin API using proper credentials in another file. Without this interaction between the firebaseAdmin and server files, there would be no persistent data offered by the application.

### *B.4 Component Design for firebaseAdmin.js*

***On server startup…***

- Offer a centralized location where the Firebase Admin API is initialized.
- Use a saved json file to autocomplete the certification process without having all the sensitive data hard coded within the firebaseAdmin file.
- Export needed packages from this authorized API to be used by the server.

## C.1 Subpart - Handler Function Files

### *C.2 Abstract Specification for Handler Function Files*

The backend utilizes handler function files written in Javascript. These are small functions that the server will offload work to. The reason these functions exist outside of the server file itself is for cleanliness and ease of reading. These functions are minor and more are being created as development goes on. Due to this, they will be considered under the same subpart, since they all operate similarly, are stored in the same backend handler function folder and are used by the server file through import statements. For example, the main handler function developed so far, wordEnforcer.js, takes user input words and makes sure they meet specific criteria. These criteria include containing no special characters, being only one word, being properly spelled, and not being a bad word. This function is utilized by the server file after being imported and then subsequently called as needed in the server file.

### *C.3 Interface Design for Handler function Files*

This collective subpart of handler functions interfaces directly with the server.js file as much work needs to be offloaded to functions written in separate files to maintain readability. After these files are imported to the server file, they can be called as though those functions were written in the main server file natively. This also aids in scalability for the handler functions. For example, if it is decided to add more or less word altering criteria to the word enforcement stage of computation, developers can easily access the file containing only the word enforcement function. From here, it will be simple to add, remove or alter any code as needed.

#### *C.4 Component Design for Handler function Files*

***On server startup…***

- Reside in a specific folder called Handler functions on the backend.
- Utilize whatever manner of libraries are necessary to accomplish the function’s goal (like “profanity-util” library for wordEnforcer.js)
- Get imported into server.js and be used as needed.

### D.1 Subpart - Miscellaneous/Various Libraries

#### *D.2 Abstract Specification for Miscellaneous/Various Libraries*

Similarly to the frontend subpart with the same name, this subpart encapsulates many of the libraries utilized by the backend. These libraries include Express, http, socket.io, cors and more. These libraries all aid in the web routing and hosting of the server file itself. They are utilized by import statements into the main server file. From within the server file, their helpful functions are then employed as needed. These are all downloaded locally and accessed through the node_modules folder.

#### *D.3 Interface Design for Miscellaneous/Various Libraries*

The libraries interface with all Javascript subparts found on the backend, as each utilizes at least one library. Express, http and cors are utilized by the server file after being imported in order to facilitate the hosting of the server. Express creates the server application, http allows the server application to be of type HTTP server, then cors configures various server permissions like what methods users may use on the server. It is important to note that while socket.io on the frontend is authenticated in a separate file, it is not on the backend. This is so that during the configuration of the socket, it may access the Express application turned HTTP server object directly. It may be possible to offload this into a separate file, but as for now, this is the implementation.

Handler functions utilize libraries like the wordEnforcer.js file, which imports an npm library called “profanity-util” to offer a function that sifts through a string and turns bad word instances to “****”. This saves a lot of time, since the library provides the bad word library, leaving time for development elsewhere.

#### *D.4 Component Design for Miscellaneous/Various Libraries*

***On server startup…***

- Be downloaded to the machine server and visible in the node_modules folder.
- Get imported where needed to be used as needed.

## 3. User Interface Design

### 3.1 Design Principles:

The design principles for the user interface design ultimately need to provide a friendly user-interface that is both intuitive and in no way impedes player enjoyment or execution of the game. Principles that will contribute to the accomplishment of this goal include keeping the design easy to understand, easy to navigate, easy to access, maintaining fast UI response time (or display loading bars if slow), and keeping the visuals bright and exciting.

### 3.2 Examples:

The UI of the application currently revolves around the idea of simplicity. Users are greeted by buttons on the page, only two to three at a time. They have no other options to click. Each button has a word or words displayed upon them, giving context to what they do.

When users are in a lobby, the host gets an option to start the game in bright green. This is the only bright green button on the website. It offers a visual cue to press it for a good effect. *Green means go!*

When users are in a lobby, the host of the lobby gets a crown displayed next to their names. This crown is visible to all players in that lobby. This lets all players know who has the power to start the game. This also lets the players know who they have to yell at if they have been waiting around forever for the game to start.

## 4. Help System Design

### 4.1 Implementation:

Currently the application offers no help system. This is not to say it will not be appended in the feature. If it were to be, the help system design would be as follows. A static webpage, not needed to connect to the backend, that displays simple FAQs (frequently asked questions) and answers about what the game is, how to play and how to navigate the website. By being static, this means the page would not require heavy programming to implement. This means it would be very easy to implement, at least much easier than creating the game page, which requires constant communication with the backend server.

This page would become a subpart of the frontend, but would likely not utilize anything but React components and Material UI for pleasing visuals and UI. It would not need socket.io or anything of the sort in order to be effective in its limited FAQ helping role. It would be accessed as a static webpage with no menus, only displaying lined up FAQs with answers.

**A Logo as Imagined by ChatGPT 4.0**

# Testing Design

## Abstract

The following chapter outlines the testing design for the project titled, Prompt Party. This project is a browser-based AI art game that utilizes a React frontend written in JSX connected to a Node backend written in JavaScript built on the Express framework. Rather than an integrated self-hosted database, the application utilizes the Google Firebase services to provide data storage and retrieval. The following sections will cover the project's testing design in terms of how developers will test different aspects of the project, then lastly, how the project has been tested thus far in development. It should be noted that some aspects of the testing to be covered, such as acceptance testing, are conceptual of a potential implementation for this project. This is due to the fact that this project is not planned to truly be released publicly as a consumer product!

## Introduction

The following introduction section will overview the major phases of the testing process…

### A. Unit Testing

**Testing individual functions of the system can be done through automation or developer verification assuming proper safeguards are in place.**

- ❖ Some of the functions within this project require a socket to receive a message to operate.
- ❖ Others do not require this.
- ❖ As such, when performing unit testing there are two kinds of testing utilization.
- ❖ Firstly, you may utilize the frontend console to emit socket messages to the backend to fire off functions.

- These functions should contain debug states to ensure the correct information is relayed.
- These functions should also contain thrown exception states in case the expected information is not received, not processed correctly or the function operates in some undefined way.
- The other method of unit testing is for functions that do not require sockets.
- Testing for these functions (primarily existing on the backend), can be called by themselves given arbitrary test data.
- These functions will then process the arbitrary data, which then can be compared to what is the expected output.
- If they match, the test succeeds.
- One example of this is the convert prompt to AI image function which resides on the backend.
- This function does not require a socket message to function; therefore, it may be called in an isolated development environment, then the image generated may be verified by the developer visually to ensure the function operated as intended!

## B. Module Testing

**Testing modules, a collection of units composing some small component of the overall system, can be done by utilizing the component in a minimal environment.**

- Having a module with some function, such as simply displaying the game page, can be employed in an isolated environment away from the main application.
- This means that in isolation, the developers can more easily figure out if the module works as intended before pushing it to the larger system.

- ❖ Many of the modules within this project must be visually analyzed, which leaves automated testing as less useful.
- ❖ Using these module display components in a separate environment, then receiving recognition from an actual developer’s view will be better suited for module testing in this case.
- ❖ That is not to say that we cannot also use automated testing in the form of debug statements and thrown exceptions if something goes awry.
- ❖ In cases like these, automated testing may be able to catch errors and better instruct the development team of what went wrong with the module.

## C. Subsystem Testing

**Entire subsystems are able to be tested by specifically paying attention to or targeting subsections on the webpage and ensuring they operate as intended.**

- ❖ The developers, or the internal testing team, can ensure large sections of the system operate as intended by trying to use them specifically.
- ❖ For example, the login feature prominently featured on the landing page of the website is a subsystem, or, a section of the overarching project containing many different modules.
- ❖ This subsystem is able to be tested by developers repeatedly logging in and logging out of the webpage.
- ❖ It is assumed that any undefined or incorrect behavior of the subsystem would be caught here by the repetitious nature of repeatedly using the subsystem.
- ❖ This can be further aided by helpful debug statements to prove that the data structures or objects being mutated are doing so correctly.

- This goes hand in hand with throwing exceptions if results received do not match expected results.
- Developers can then fix bugs as needed reading these exceptions, which give a good idea of where the subsystem went wrong.
- Often a value will be returned from the client to the server or vice versa, and when this happens, the value may be printed in a console log to ensure it is correct and expected.

## D. System Testing, Validate Requirements

**To test the entire system, the development team may play games split between multiple monitors to test the functionality of the system as a whole, as well as validate the requirements.**

- Running through entire games split between one developer or several, bugs can be found, and unintended behavior can be noticed.
- It is better that the internal testing team finds these bugs before pushing the project to acceptance testing.
- Validating requirements comes down to verifying that the system is able to perform the many tasks outlined early in the project's development.
- These requirements include being able to login, play a game, make a prompt, guess a prompt, and more.
- Since these requirements are crucial to the system, if something were to be missing, it would be noticed quickly!
- The same idea applies to the system, as if a core component of the system is missing, the game will not perform, and this will be noticed very quickly!

### E. Acceptance Testing

**As the project is not intended for public release at this time, the following outline of acceptance testing for Prompt Party will be a potential possible implementation.**

- ❖ The game would be made available in a beta or alpha build to select users who would be the target audience for this project. This is known as a closed beta, or closed alpha testing.
- ❖ The users will play the game and submit feedback to a feedback portal.
- ❖ This feedback could be related to the system design or aspects of the game design.
- ❖ For example, equally valid user reports submitted to this portal could be either the login system does not work well or the time of rounds is too fast.
- ❖ After a brief period of time, the closed beta or alpha test would cease and the submitted feedback would be implemented on an as needed basis.
- ❖ An example of a supposed feedback portal could simply be aggregating results of various user submitted Google forms into a Google sheets spreadsheet.

## Testing

### A. List tested items organized by phases.

**Note:** The project utilized the module testing process most often, meaning, testing a complete module to ensure a single function works as intended. This was necessary as the webpage is a live communicating service, in which special situations must be true for a given function to fire off in its intended way. As such, the following shall list modules tested, then list the various functions, react hooks, or other code that comprise that module.

## I. Front-end Testing

Following the system design structure outlined in previous reports, testing begins at the frontend level. Modules tested here begin all stem from important user requirements as needed by the client side.

- **Login and Logout Module (as displayed on entry to the webpage).**
  - Login React Hook
  - Logout React Hook
- **Lobby Manager Module.**
  - Join Lobby Function
  - Leave Lobby Function
  - Create Lobby Function
  - Lobby Live Listener React Hook
- **Game Manager Module.**
  - Handle Player Turn Function
  - Handle Player Prompt Function
- **Page State Manager Module.**
  - Various React Hooks change value depending on whether certain conditions are met such as…
    - isHost?
    - inLobby?

## II. Back-end Testing

Follows previously mentioned listing principal.

- **Lobby State Manager**

- ➢ **Join Game Function**
- ➢ **Leave Game Function**
- ➢ **Transfer Lobby Host Function**

❖ **Game Logic Manager**

- ➢ **Player Turn Controller Function**
- ➢ **Player Score Controller Function**
- ➢ **Retrieve Word Bank Function**

❖ **AI Image Generation Manager**

- ➢ **Stable Diffusion Generate Image Function**
- ➢ **Display Image Elements Function**

## B. Requirements Traceability.

In accordance with the requirement specification document, here are how different requirements are specifically tested to exist and function as intended…

❖ **Accessibility**

- ➢ Tested visually by developers, utilizing library Material UI, React code.

❖ **Login**

- ➢ ***Login and Logout Manager****,* as previously mentioned, encompassed fully by functions on the front-end.

❖ **Hosting/Joining Games**

- ➢ ***Lobby Manager Module****,* as previously mentioned, encompassed by both functions on the front-end and back-end.

❖ **Word Bank Entries**

- ➢ ***Game Manager Module****,* as previously mentioned, encompassed by both functions on the front-end and back-end. React Components to display the submit box and the completed bank. Back-end functions to retrieve and store these words to the database.

- ❖ **Prompt/Image Generation**
    - ➢ ***Stable Diffusion Generate Image Function**, as previously mentioned,* encompassed by functions on the back-end. However, it does require front-end functions that supply the backend with the prompt to be used, BUT, it may be called from the backend without this and still function!
- ❖ **Prompt/Image Guessing**
    - ➢ ***Game Manager Module**,* as previously mentioned, encompassed by both functions on the front-end and back-end. React Components to display the guess submission box which will be processed on the backend. Before being guessed, a back-end function will send the image to users when prompted by the front-end.
- ❖ **Scoring**
    - ➢ ***Game Manager Module,*** in which scoring is a standalone function on the backend that tasks the guess and compares its accuracy, scoring accordingly. It will store the score to the database and also send it back to the front-end to be displayed in the game. This display in the game on the front-end would be part of the **Page State Manager Module.**
- ❖ **Hardware**
    - ➢ "*computer that can access the internet. They also require a stable internet connection to maintain connection to the game lobby and other players. They will require computer device peripherals like a keyboard*" - **Requirements Specification Document**
    - ➢ The project was **tested on all the mentioned hardware.**

- **Software**
  - “*Users will need their computers to be able to access the most popular, well-supported internet game browsers. These browsers include, but are not limited to, Google Chrome, Microsoft Edge and Mozilla Firefox.*” - **Requirements Specification Document**
  - The project was **tested on the types of browsers mentioned above.**

## C. Testing Schedule

Test scheduling was performed on a as needed basis. For example, when developing the project the multi-socketed nature of the programs necessitated that the frontend and backend components to any given module must be completed at the same time. A developer would not, for example, write front-end code for a join lobby function, then not write the back-end response to that command in succession. This meant that the testing schedule really went as follows…

- **(Phase 1 - Login Module → Front-end tested only)**
- **(Phase 1 - Page State Manager → Front-end and Back-end tested)**
- **(Phase 2 - Lobby Management Modules → Front-end and Back-end tested)**
- **(Phase 3 - Game State Management Front-end and Back-end tested)**

## D. Test Recording Procedures

While no rigorous test recording procedures like saving all tests made to a massive spreadsheet, there are saved data entries into the database. Whenever the program runs, much of the data it processes or utilizes ends up saved in the database under either the player’s stored document containing their information or the lobby’s stored document containing game information. All tests, meaning run throughs of games, can be seen here. Developers are able to view the data saved here for undefined behavior or unexpected entries. In a way, this is the testing recording procedure utilized.

Test recording procedures also exist in the many console logs both on the server console and the front end console. These print out much needed information which can be cross referenced with what was intended to be sent. For example, if you know a front-end function needs to send a lobby ID to a given back-end function, try to console log the received data. If the back-end received a “NULL”, it is known the test has failed! If it received a lobby ID as needed, it is known to have succeeded and may proceed.

## E. Hardware and software requirements

- Tools required and hardware utilization needed for the testing performed thus far have been on a Windows machine running Windows 11. **The technical specs are as follows…**
  - AMD Ryzen 7 5800H 3.20 GHz Processor
  - NVIDIA GeForce GTX 3060 GPU
  - 16 GB of RAM
  - 477 GB of storage
- The specifications above are overkill, but it should be noted that running the Stable Diffusion model as used by the generate AI image function requires a hefty graphics card to generate images quickly. Developer’s attempting to self host the server as has been done so far will require a beefy graphics card such as the one outlined above to utilize the model successfully.

- A tool utilized was localhost in which the project was hosted locally on the machine to simulate the web server/client model environment.
- Another tool was the IDE, Visual Studio Code.
- Yet another tool(s) was the usage of internet browsers (specifically Chrome and Edge) to access the locally hosted project on localhost.

PROMPIN
PRRY
P

**A Logo as Imagined by Stable Diffusion v1-5**

# User Manual

## Abstract

The following manual describes the functionality and reason for the system. Topics covered include how to use the system, information on the help system, listing of services, and error recovery methods. Lastly, there is a glossary to explain technical terms that may be unknown or confusing to users. After reading this document, users can expect a good wealth of knowledge regarding the Prompt Party product, how to use it, how to operate when potential errors arise, and somewhat on how the product works on a technical level.

## Introduction

***The system, Prompt Party, has many functionalities and reasons for these functionalities to exist…***

#### Login Functionality/Reason

Firstly, the system offers the functionality to login. The reason for this is to have a way to identify users and have a way to save their information. Particularly saving their scores for the leaderboard. This feature functions as many other popular apps users may be familiar with, this is due to the fact it utilizes a Google product known as Firebase. Firebase allows the application to let user's log in using a screen commonly seen in Google products like YouTube and Gmail.

Subsequently related, the application allows a user to log out, much the same as these other Google applications. This is helpful and reasoned to exist in case a user wishes to login to a new account, or remove their credentials from a shared computer so other users do not have access to their account.

### Matchmaking Functionality/Reason

Secondly, the application allows the user to create a lobby or join a lobby. The reason for these functionalities are to connect players together in order to allow them to play. A lobby can be thought of as an exclusive room, of which each user inside the room gets to play the game together. The application offers the functionality of safeguarding a lobby with a special lobby ID. This lobby ID is akin to a password to enter the special exclusive "room" as previously mentioned. Without a lobby ID, any unknown users would be able to come into the room freely, which for some products may be fine, but it is prohibited in this application.

### Host Functionality/Reason

Thirdly, within a game the system offers the functionality of offering a host position for the creator of a lobby. The reason for this is to have a user that can dictate the flow of the game. This user gets to click the "begin game" button, as well as click the "go to next turn" button during gameplay. The creator gets sole control, so that many users cannot fight over these game dictating privileges. The host has these special privileges unless they click the "leave lobby" button. If this happens, the first player that joined the lobby after the host becomes the newly appointed host, gaining the special privileges. This behavior continues if the newly appointed host leaves as well. If the host leaves the game and there are no other users within the lobby, the lobby will be closed and will disappear. If this happens, the lobby ID previously used will no longer function.

### Word Bank Functionality/Reason

Now that the game has started, all users are exposed to the system functionality of entering words into the word bank. This is offered by a large submission box displayed to all users within the same game in the same lobby for 30 seconds before disappearing. The reason for this functionality to exist is that it creates what may be considered a “universe” of words of which users may utilize. The custom “universe” idea allows users to tailor their games to their own humor which adds infinite replayability! This word bank/universe is stored within a database, which is stored on the server, which can be thought of as stored on a secure file on a computer stored elsewhere that is accessed when a user connects to the webpage.

### Prompt Creation Functionality/Reason

The system offers the functionality for the user to choose a prompt from the word bank created in the previously mentioned functionality. The reason for this is to further the game and generate an image that players will guess the prompt for. Without this functionality, there would be no core gameplay loop, it is essential to the design of the system. Users will be presented with all the words from the word bank in a table and choose accordingly. The UI elements display and give a good indication of what users will press to choose words or to delete words. The prompt being developed is displayed prominently in the center of the screen. No other users except the sole user creating the prompt (happens once per player turn) can see this step. The user can submit when they feel that their prompt is finished.

### Guessing Functionality/Reason

The system offers the functionality of guessing the prompt of the image displayed. After the previous step, the corresponding generated image is displayed prominently to all users in the same game. User’s will then be presented with a table of words, the same design as the prompt creation table of words. User’s will then choose words from this table to submit a guess. This guess is then graded by a grading algorithm on how accurate it is to the true prompt. The reason for this functionality is to create a competitive atmosphere where players have to compete for points through accurate guesses!

#### Score Screen Rank Functionality/Reason

After each player has gone for their turn, the game will end and the system will offer the functionality to display a ranking table of that game. It shows which player won and subsequent rankings of the other players within that lobby. This is important for users so that they may know if they won the game which contributes to the competitive atmosphere. The UI is intuitive in that winners are displayed in gold with a golden trophy displayed next to their name.

#### Global Leaderboard Functionality/Reason

The system offers the functionality to access a global leaderboard which contains the lifetime scores of all registered users in the database. Users can imagine this as the accumulation of all points they received over all games played on a single registered account. For example, if Player 1 has played four games on his account total, and scored 100 points in each of those games, Player 1 will be displayed in the lifetime scoring leaderboard with 400 lifetime score displayed. This further contributes to the competitive atmosphere, so that not only are users competing in a single game but also vying for the top spot on the lifetime scoring leaderboard.

## Introductory Manual

### a. How to Use the System

To use the system, a user must first navigate to the webpage. Upon accessing the webpage, a user must login. A login is required to generate an authentication token that allows the user to communicate with the server. The server can be thought of as a separate computer running elsewhere running the necessary technology for the application. A user must communicate with this computer to access the application's aforementioned functionalities. Upon logging in, users must choose to create a lobby or join a lobby. It can be easily thought of as, create a lobby if you want to play a new game, join a lobby if someone already has a lobby and you wish to join them. Users should also remember that if they choose to create a lobby, they will have special host privileges.

Within a game, user's will be shown a word bank submission box. The design here is intuitive enough such that user's will see they need to simply write a word in the box and hit enter on their keyboard or hit the submit button. These words go into a shared word bank to be used by all players within the same lobby. Next, the game will alert users when it is their turn. If it is their turn, they will be displayed with another intuitive designed table to create a prompt from the previously created word bank. After this is submitted, by hitting the submit button on the page, another element is displayed. All users will now guess the image prompt using a table that looks just like the prompt creation table. Users have to click the words displayed there. Each word may be used only once.

In between turns, the host must click a button that says something along the lines of "Go to the next turn", so that the game may continue. The host is given this privilege so that the score screen that is displayed after all guesses are received may be looked at for however long as needed. It also allows the space for a break, such that if a user needed to use the bathroom they could and it would not impact their placing in the game.

Once a game is completed. The score screen will show the final rankings of all players. At this point, users must all click the "leave game" button as the game has entered a state in which no gameplay will further happen. The application does not support restarting the game from within this game over state. Users must leave the game and create a new one.

Users will want to see the global standings of the scoring leaderboard. To do so, simply click the scoreboard icon on the header of the webpage located at the top right. It will display an intuitive menu showing all top player lifetime scores.

Lastly, users may want to communicate with each other over text chat while playing the game. To do so, users must click the expand chat button which will appear once the game has started and be positioned on the bottom right of the screen. This opens a group chat style screen that looks much like text messages on a cell phone. Users must input their text into the text box and click submit to send the message to other users.

## b. Information on the Help System

The help system offered by this product is a FAQ or “frequently asked question”, pop up menu that is accessible via a button in the header. This FAQ icon once clicked will open a table to display commonly asked questions and how to remedy potential errors that may arise with the system. It will also display the rules of the game in case users get confused about the core gameplay loop or some decisions made about scoring/host privileges, etc.

# System Reference Manual

## a. Listing of Services

- **C.** Chat Room Services
- **F.** FAQ services
- **G.** Global leaderboard Services
  Guessing Prompt Services
- **H.** Host Privileges Services
- **L.** Login & Logout Services
- **M.** Matchmaking Services
- **P.** Prompt Creation Services
- **S.** Score Screen Services
- **W.** Word Bank Submission Services

## b. Error Recovery

Most commonly, errors with the system can be fixed by refreshing the webpage. This will refresh a user's socket identifier. The browser may refresh a user's socket identifier without letting the system know, in a case like this, the system would then be communicating with a socket that no longer exists. This would cease communication between the system's backend and the user. When refreshing, the socket identifier will be regenerated, alerting the system and thus re-engaging communication with the user.

During a game, the gamestate may reach a locked status in which progress will no longer continue. This may occur if player's leave at inopportune times, such as right before it becomes their turn, during their prompt generation turn, or during guessing. There are safeguards in place to prevent this, but in the chance that it does occur, users are advised to leave the game and start another one. On a related note, users are disallowed from joining a game in progress so that the chances of these locked states within a game are minimized.

To help combat this locked state, as part of the host privileges, the host may hit the “Stuck” button to skip over a player's turn and bypass whatever message is being waited on but will never be received. This feature has the ability to be abused by the host, as they may skip competing player’s turns, so it is the honor system by which this feature will operate. It is a necessary evil to ensure the game does not freeze and become stuck forever. If you do not trust other hosts to not abuse this feature, then consider hosting the game yourself to properly utilize it! This may break your game if spammed, try to press it once then see what happens. If you are still stuck, please press again and wait. Repeat as needed.

## User Manual Glossary

| **Term** | **Definition** |
|---|---|
| **Application** | A program or group of programs designed for users to perform specific tasks. |
| **Authentication Token** | A digital key that verifies a user's identity to access secure systems. |
| **Backend** | The part of a software system that runs behind the scenes, handling logic, databases, and servers. |
| **Database** | An organized collection of data that can be easily accessed, managed, and updated. |
| **FAQ (frequently asked questions)** | A list of common questions and answers to help users understand a product or service. |
| **File** | A collection of data stored as a single unit on a computer. |
| **Firebase (software)** | A platform developed by Google for creating mobile and web applications with backend services like databases and authentication. |
| **Frontend** | The part of a software system that users directly interact with, such as a web page or application interface. |
| **Functionality (software)** | The specific features or tasks that a software program is designed to perform. |
| **Prompt (AI image generation)** | A text instruction given to an AI model to generate a specific image based on the description. |
| **Server** | A computer or system that provides resources, services, or data to other computers over a network. |
| **Socket** | A communication endpoint used to send and receive data between computers over a network. |
| **Socket Identifier** | A unique label used to recognize and manage individual socket connections. |
| **System** | A set of connected components or programs working together to perform tasks. |
| **UI (user interface)** | The visual and interactive elements through which a user engages with a software application. |
| **Webpage** | A document on the internet that is viewed through a web browser. |

## Project Narrative Section

The development process for this project began as early as November 2024. This is where my original idea of creating a browser-based party game began, however, I wanted to make a trivia game at first. This idea was later replaced sometime around early January 2025, with the idea to make an AI-music hybrid project. However, this music idea did not stick as I had read that Spotify’s APIs were going to be changed in the coming months. Around late January 2025, I returned to the idea of a browser-based party game. This time, I had the idea to add AI image generative models to the game. This idea stemmed from the JackBox.tv game called Tee K.O., in which players compete over drawing the best t-shirt designs. I had played this sometime in mid-January, and it really got me thinking about how much I enjoyed visual elements and the idea of a communal word bank element to add to replayability.

The programming of this application began in early February, in which the very first steps were deciding on a tech stack. I had gone with the React and Node.js stack due to my familiarity with them, but I would soon find out that I was not as familiar as I would have thought. It took a while before these languages became intuitive to me. I also decided to use Socket.io, the premier JavaScript socket library, since I knew I would need communication between the frontend and backend. I also took a chance on using Firebase, the Google product. I was unaware of Firebase’ existence until early February when I was researching ways in which I could seamlessly integrate a database and login features to a web app. Firebase really seemed to be the best way for me to accomplish these features, since it was free, it was a Google product (which I have always had good experiences with, especially their documentation), and it natively accomplished all these features through an easy to use API. It even has the feature of allowing the API key to be publicly visible, since it only corresponds to a single project, hence Prompt Party, rather than connected to a credit card.

I coded and assembled the project within VSCode. It is the IDE I have become the most familiar with. As for documentation, I wrote everything in this document using Google Docs. I find that Google Docs is much more user friendly than Microsoft Word. The only time I used Microsoft Word is at the completion of a document, in which I convert the Google Doc file into a Microsoft Windows DOCX file. I also use Microsoft Windows at this stage to add the index to the document. This is because natively, Google Docs does not support indexes. At first, I made the indexes manually in Google Docs, but this was time consuming and not worth the effort because it always looked a bit ugly.

Another tool I used was the Stabled Diffusion Web UI, accessible on GitHub. I downloaded this and a Stable Diffusion model off the internet, I ran these on my laptop which acts as my server. When this web UI was running, I could use it like an API and take user prompts and turn them into images. This was really helpful, as native JavaScript AI image APIs like OpenAI's ChatGPT image generator cost money and have so many guardrails. For example, these APIs cannot depict any licensed characters. Running my own model locally allowed me to get rid of some of these guard rails and make images for free.

The project was written and assembled over the course of the Spring 2025 semester, written almost entirely at my desk in my bedroom in my off-campus apartment. I did not work with any peers on this project. I used the internet, YouTube and help from AI when I got stuck. Admittedly, this worked but at first it was very hard. I quickly learned that a lot of the first steps I made, such as file organization of the project, were not scalable and became too messy. I later went back and had to fix these, which was not a problem, but if I had organized the files correctly from the beginning, I could have saved myself time and effort.

The project reached its final and current form around late April 2025. In total, I think I met the mark with what I was trying to accomplish. However, I could definitely see points of improvement. For example, the AI image generation is not as accurate to prompts as I would like, however, this is really a limitation of both hardware and software. More advanced models are locked behind paywalls, and even if I did have access, my computer's graphic card may not be powerful enough to run said models. If this project were going to be released to the public, this is definitely something I would change. I could rent a more powerful server and pay for more powerful AI models, but I view this project more as a proof of concept for an AI image game rather than a fully realized product.

Working on this project taught me a lot, not only about this tech stack, but time management, overcoming coding burn out, and more. I would like to extend my thanks to my advisor Professor Jackowitz whose weekly meetings kept me on my toes making steady progress throughout the semester, and whose questions of why I chose to implement certain features over others always had me thinking about and improving my system design.

- Brian Meder

# System Maintenance Guide

## Running the App

## Known Problems

- Sometimes the web page becomes unresponsive. This is more than likely due to the fact that the socket ID of the client has been refreshed without the server's knowledge. Refresh the page to fix this.
    - Also, if you attempt to log in and the page freezes, refresh.
- Sometimes if a player leaves a lobby mid-game, it causes the game to lock up. This is due to the fact that a socket listener on the server is waiting for a message that will never be received. If this is the case, all players must leave the game and make a new one. The host may also use the special "stuck" button to bypass the turn of the player who has left and the system is waiting on.
- There is a spacing issue on the webpage, in which the viewport width has grown slightly larger than it should be. This does not impact performance or function; therefore, it is a simple visual bug.
- If you create a prompt and submit it and nothing happens, it is likely the server is not running the Stable Diffusion Web UI. Ensure this is running before any games occur. This must be done manually, the model does not start with the server.

## Hardware and Software Dependencies

- The frontend software relies on the backend software. Both must be running for the project/application to function as intended.
- The front end and backend both rely on the running of the Stable Diffusion Web UI software. Without this, there will be no images displayed, and the game will lock up during the display image step of gameplay.
- The Stable Diffusion Web UI relies on graphics card hardware. These models require a hefty graphics card to create images at a fast speed. Therefore, the server running the Node server file and the Stable Diffusion Web UI depends on a good graphics card. This project was tested on an NVIDIA GeForce GTX 3060 GPU.

## Product Evolution

- Thanks to the intuitiveness of React, anytime a developer wishes to add a new feature they may add a new React component. This React component can then correspond to a backend handler JavaScript file which can perform some tasks.
- Together this makes the product able to evolve with new features that can be appended quite easily!
- Thanks to this documentation, developers can get a better idea of where to begin, how the project works, and how the system architecture is set up before they begin writing code.

# Index

abuse, 8, 45
Acceptance Testing, 31
accomplishment, 25
advisor, 49
AI, 0, 7, 8, 9, 10, 11, 12, 14, 16, 27, 28, 33, 36, 46, 47, 48, 49
AI image, 7, 14, 28, 36, 46, 47, 48, 49
API, 7, 8, 9, 17, 18, 19, 22, 23, 47, 48
API key, 18, 19, 48
API keys, 22
application, 16, 17, 18, 19, 20, 21, 23, 24, 25, 27, 28, 38, 39, 42, 43, 46, 47, 50, 51
**Application**, 20, 46
architecture, 51
**Authentication Token**, 46
backend, 16, 17, 19, 21, 22, 23, 24, 25, 27, 28, 34, 35, 44, 46, 47, 50, 51
**Backend**, 21, 46
bugs, 30
code, 13, 18, 19, 22, 24, 31, 33, 35, 51
communication, 17, 21, 26, 44, 46, 47
configuration, 17, 24
connection, 14, 19, 34
console, 27, 30, 36
credentials, 19, 22, 23, 38
data, 13, 16, 21, 23, 27, 28, 29, 35, 36, 46
database, 7, 8, 16, 18, 21, 22, 27, 34, 35, 40, 41, 47
**Database**, 46
debug, 28, 29
design, 16, 19, 25, 27, 31, 32, 40, 42, 49
developers, 23, 27, 28, 29, 33, 51
environment, 28, 29, 37
exception, 28
**FAQ**, 26, 43, 44, 46
features, 11, 12, 17, 18, 19, 22, 46, 47, 49, 51
feedback, 31
file, 17, 18, 19, 20, 21, 22, 23, 24, 40, 48, 51
**File**, 18, 46
**Firebase**, 16, 17, 18, 19, 21, 22, 23, 27, 38, 46, 47
Firebase admin, 22
folder, 20, 21, 23, 24, 25, 50
frontend, 16, 17, 18, 20, 21, 22, 24, 26, 27, 32, 35, 47, 51
**Frontend**, 16, 46
function, 17, 18, 23, 24, 28, 31, 33, 34, 35, 36, 39, 50, 51
functionalities, 12, 38, 39, 42
functionality, 30, 38, 39, 40, 41
**Functionality**, 38, 39, 40, 41, 46
game, 7, 8, 10, 11, 12, 13, 14, 15, 16, 20, 21, 25, 27, 28, 30, 31, 34, 35, 36, 39, 40, 41, 42, 43, 44, 45, 47, 49, 50, 51
Google, 13, 14, 15, 16, 18, 19, 23, 27, 31, 35, 38, 46, 47, 48
graphics card, 36, 51
handler, 21, 22, 23, 51
handler functions, 21, 22, 23
hardware, 10, 34, 36, 49, 51
host privileges, 42, 43, 45
HTML, 8, 20
initialize, 19, 22
input words, 8, 14, 23
installation, 50
internet, 7, 10, 12, 14, 15, 18, 34, 35, 37, 46, 48
JavaScript, 8, 16, 27, 47, 48, 51
JSX, 16, 17, 19, 20, 27
leaderboard, 7, 38, 41, 43, 44
libraries, 18, 19, 20, 21, 24
lifetime scores, 41, 43
lobby, 8, 13, 14, 22, 25, 34, 35, 36, 39, 40, 41, 42, 50
lobby ID, 13, 36, 39
login, 13, 14, 18, 23, 29, 30, 31, 38, 41, 47
manual, 38

Manual, 38, 41, 44, 46
Material UI, 17, 20, 26, 33
menu, 43
model, 12, 13, 36, 37, 46, 48, 50
module, 28, 29, 31, 35
Node, 16, 21, 27, 47, 50, 51
NVIDIA, 36, 51
party game, 10, 12, 47
party games, 7, 10
player, 7, 8, 10, 12, 13, 14, 22, 25, 36, 39, 40, 41, 43, 44, 45, 50
programming, 8, 25, 47
project, 7, 8, 10, 11, 12, 16, 27, 29, 30, 31, 34, 35, 37, 47, 48, 49, 50, 51
prompt, 7, 12, 13, 14, 28, 30, 34, 40, 42, 44, 50
Prompt Party, 0, 11, 12, 14, 16, 18, 20, 27, 31, 38, 48, 50
python, 50
React, 8, 16, 17, 18, 19, 20, 26, 27, 32, 33, 34, 47, 50, 51
react hooks, 31
requirements, 0, 12, 30, 32, 33, 36
scoreboard, 43
screen, 13, 38, 40, 42, 43
sensitive information, 22
server, 9, 21, 22, 23, 24, 25, 26, 30, 36, 37, 40, 41, 48, 49, 50, 51
**Server**, 46
socket, 17, 19, 21, 24, 26, 27, 28, 44, 46, 47, 50
**Socket**, 8, 17, 19, 22, 46, 47
Stable Diffusion, 33, 34, 36, 37, 48, 50, 51
Stable Diffusion Web UI, 51
static, 25, 26
stuck, 45, 48, 50
system, 8, 12, 16, 25, 27, 28, 29, 30, 31, 32, 38, 39, 40, 41, 43, 44, 45, 46, 49, 50, 51
**System**, 0, 14, 16, 25, 30, 41, 43, 44, 46, 50
System Testing, 30
testing, 27, 28, 29, 30, 31, 32, 35, 36
throwing exceptions, 30
**UI**, 8, 25, 26, 40, 41, 46, 48, 50, 51
unintended behavior, 30
Unit Testing, 27
username, 13
webpage, 8, 13, 18, 19, 21, 25, 26, 29, 31, 32, 40, 41, 43, 44, 50
**Webpage**, 14, 17, 46
word bank, 12, 13, 39, 40, 42, 47