Software Design Document

CSS 566 Software Management

Spring 2023, University of Washington Bothell

Team: Purple Kitty Squad

Author: Warren Liu (Lead Developer/Architecture)

Reviewers:

- Barack Liu (Product Owner)
- Haihan Jiang (QA Engineer)
- Shahruz Mannan (Sprint 1 Scrum Master)
- Amalaye Oyake (Sprint 2 Scrum Master)
- Arsheya Raj (Designer)

1 Introduction

1.1 Purpose

As a type of puzzle, word search games—also referred to as word find, word seek, word sleuth, or mystery word games—challenge players to locate words concealed within a grid of letters. Our main objective is to offer a captivating and enjoyable web-based word search gaming experience that appeals to a wide variety of players. By doing so, we hope to boost user engagement on the website hosting the game, as evidenced by the duration of visits and the number of return visits.

The primary objective of this project is to develop a straightforward Word Search game. Instead of emphasizing software development, the project seeks to provide an opportunity to experience the Scrum management framework. We intend to complete this project within 2 sprints in 4 weeks.

1.2 Scope

The scope of this document covers the design and architecture of a three-tiered client-server application that allows users to play games through a frontend user interface, which contains the game logic, a backend server managing user login/logout, user interaction, and puzzle generation, and a database for storing game data and user information. The key components included in the scope are:

- Frontend UI: A responsive web-based user interface that handles game logic and provides users with access to game selection, gameplay, score uploading, designing customized puzzles, and leaderboards.
- Backend Server: A server-side component that handles puzzle generation, user authentication, and data retrieval/storage. This component exposes a RESTful API for communication with the frontend UI.
- Database: A database system for storing and managing game data, user information, and user scores.

1.3 Roles and Responsibilities

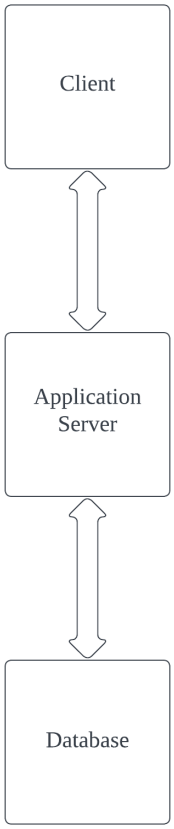
Besides the roles assigned above, the parties below are further responsible for:

- Warren Liu: Website backend development and database schema configuration
- Shahruz Mannan: Game program development
- Arsheya Raj: Website frontend UI/UX development
- Haihan Jiang: QA
- Barack Liu: Manage development progress and maintain all the supporting documentation

2 System Architecture

2.1 Overview of the Three-Tier Architecture

In this project, we will implement a three-tier client-server architecture. The first tier will be the client, where users engage in the game; the second tier will be the application server, hosting our backend functions; and the third tier will comprise the database, storing data such as user and game information. Each layer exposes an interface (API) for use by the layers above, with each layer depending solely on the facilities and services of the layer directly beneath it.



2.2 Client Tire (Frontend UI)

The client tier consists of a web-based user interface that allows users to interact with the application, select and play games, create customized puzzles, manage their user accounts, and view leaderboards. This frontend UI is built using modern web technologies such as HTML, CSS, and JavaScript, along with popular frameworks and libraries (React-native) for a responsive and user-friendly experience. The client tier communicates with the server tier through RESTful API calls to fetch data and submit user actions.

Key responsibilities of the client tier include:

- Presenting the game selection and gameplay interface to users
- Processing game logic and managing game state
- Presenting puzzle customization interface to users
- Handling user input and interactions
- Managing user authentication and authorization
- Communicating with the backend server via API calls

2.3 Server Tier (Backend Server)

The server tier is responsible for managing user sessions, generating random puzzles, and handling API requests from the client tier. This backend server is developed using a suitable server-side programming language (Python) and web application microframework Flask. It exposes a RESTful API for the client tier to interact with, allowing for data retrieval and storage, user authentication, and game-related operations.

Key responsibilities of the server tier include:

- Authenticating and authorizing user access
- Generating random puzzles
- Handling API requests from the client tier
- Interacting with the database tier for data storage and retrieval

2.4 Data Tier(Database)

The data tier is responsible for storing and managing application data, such as user information, puzzle data, and user scores. It consists of a database system, the NoSQL database (MongoDB). The server tier interacts with the data tier through a data access layer, which abstracts the database operations and provides a consistent interface for data manipulation.

Key responsibilities of the data tier include:

- Storing user information, puzzle data, and user scores
- Managing data integrity and consistency
- Ensuring data security and privacy
- Providing data access and query capabilities to the server tier

3 Frontend UI Design

3.1 User Interface Components

3.1.1 Landing Page

The Landing Page serves as the main entry point for users and provides access to key features of the application, such as logging in, viewing leaderboards, accessing play history, and selecting games to play.

- **User Authentication**
    - A `Login` button is located in the top right corner.
    - Clicking the `Login` button opens a modal for users to log in or register for an account (via POST requests).
- **Leaderboard (Data will be fetched via Ajax)**

    - A large leaderboard displays the top N scores for each of the three levels.
    - A `Refresh` button to refresh the data.
    - Clicking on a level's text opens a modal with the top 100 scores for that level.

- **Play History (Visible after user login, data will be fetched via Ajax)**

    - A list of the user's last 10 games is displayed.
    - Clicking on an item in the play history allows users to replay that game and attempt to achieve a higher score.
    - Clicking on the `Full History` opens a modal with all game histories.

- **Level Selection**

    - A button for users to choose a level to play.
    - Clicking the button opens a modal with three buttons corresponding to the three levels.
    - Clicking on a level button directs users to the game page for that level (will be specified in 3.1.3).

- **Today's Reward Game**

    - A button for users to play "Today's Reward Game" directly.
    - Clicking the button immediately directs users to the game page (will be specified in 3.1.3).

- **Puzzle Design**

    - Clicking the button immediately directs users to the "Design Puzzle" page (will be specified in 3.1.2).

3.1.2 Design Puzzle

Users are able to design their own puzzles on this page. The page has a form that contains:

- **Input Text and Level Selection**
    - A text input field for users to enter comma-separated single words (e.g. cat, dog, fish. Validation for no blank exists within commas is needed here, or in the backend **TODO**).
    - A selection menu for users to choose the desired puzzle level (1-3).
- **Confirm Button and Puzzle Generation**
    - A `Confirm` button to submit the input text and level information via Ajax.
    - The backend receives the information, generates a puzzle, and sends it back to the client via Ajax.
    - The generated puzzle is displayed to the user.
- **Puzzle Modification**
    - There are two buttions, `Confirm` and `Regenerate`. If the user is not satisfied with the generated puzzle, they can either click the `Regenerate` button again or modify the input text and level, then click the `Confirm` button again to submit their updated design. If the user is satisfied with the generated puzzle, they can directly click the `Confirm` to submit their puzzle.

3.1.3 Game UI

The Game UI allows users to play the game, view the words they need to find, see the leaderboard for the current puzzle, and submit their scores to the leaderboard.

- **Need-to-find-words List**
  - Displays all words that need to be found in the game (provided by the backend when the page loads).
  - When a word is found, it is removed from the list.
- **Puzzle Leaderboard**
  - Displays the leaderboard for the current puzzle.
  - Provided by the backend when the page loads.
- **Puzzle Grid**
  - All game logic will be implemented and be executed here.
  - A square grid in the center of the page with each cell containing a letter.
  - Users click on cells to select letters.
  - When a selected sequence matches a word in the need-to-find-words list, the cells change color and the word is removed from the list.
  - When all words are found:
    1. Notify the user.
    2. Display a button for users to save their scores.
    3. Display a button for users to upload their scores to the leaderboard (with the option to submit anonymously).
- **Clear Selection Button**
  - Allows users to clear all selected cells in the puzzle.
- **Time Counter**
  - Displays the elapsed time for the current game.
- **New Game Button**
  - Allows users to start a new game with another puzzle at the same level.
- **Switch Level Button**
  - Offers two selections for users to switch to a different level and start a new game.

3.1.4 User Account Management

INCOMPLETE WARNING

TODO

If we have time, we can work on this page. For now, users can only register an account, and use it to log in. They will not have the option to change the password and the username.

3.1.5 Error Handling

We should write some customized error pages when backend throw an error code, such as 400, 401, 403, and 500.

3.2 Technologies and Libraries

- Frontend programming language: JavaScript, HTML, CSS.
- UI frameworks and libraries: React Native.
- Additional libraries and tools: **TODO**.

3.3 Design Patterns and Principles

Lazy Loading

TODO

Input Validation

TODO

Security Policies

TODO

4 Backend Server Design

4.1 Backend Components

4.1.1 Authentication

- Implement user authentication, this will be handled by Flask built-in authentication.
- Provide functionality for user registration and login (if 3.1.4 is implemented, more functions will be added here).

4.1.2 Game Data Management

- Store game data, such as puzzles, user scores, and leaderboards.
- Implement API endpoints for frontend communication (e.g., fetching leaderboard data, submitting scores, and generating puzzles).

4.1.3 Puzzle Generation

- Implement an algorithm for generating puzzles based on user input (text and level).
- Ensure the generated puzzles meet the level requirements.
- Respond to frontend puzzle generation requests with generated puzzles.

4.2 Technologies and Libraries

- Backend programming language: Python.
- Web application frameworks: Flask.
- Database: MongoDB.

4.3 Design Patterns and Principles

Separation of Concerns

- Separate backend components (e.g., authentication, game data management, puzzle generation) to maintain a modular design and enhance maintainability.

Scalability (Only If We Have Time)

- Implement caching mechanisms to reduce database load and improve response times.

- Use horizontal scaling to accommodate increased load and ensure high availability.

Security

- Secure the backend by implementing proper authentication and access controls.
- Implement input validation and sanitization to prevent security vulnerabilities.
- Securely store sensitive data (e.g., passwords) using encryption.

Performance Optimization (Only If We Have Time)

- Optimize the puzzle generation algorithm for efficient performance.
- Monitor and fine-tune database queries to reduce response times.

Error Handling

- Implement comprehensive error handling to ensure proper error responses are sent to the frontend.
- Log errors and exceptions for easier debugging and issue resolution (Only If We Have Time).

5 Database Design

The database will store the necessary information to support the application's functionality, including user data, game data, and leaderboards. We will use MongoDB as the database management system, which is a NoSQL database optimized for scalability and flexibility.

5.1 Collections

The database will consist of the following collections:

5.1.1 Users

This collection will store user account information, including:

- User ID: use ID generated by MongoDB directly (unique identifier)
- Username (unique)
- Password (hashed)
- Registration date

5.1.2 Games

This collection will store game data, including:

- Game ID (unique identifier)
- Puzzle grid (2D array of characters)
- List of words to find
    - A list of words
    - A list of each index of each letter of each word
- Level (1, 2, or 3)
- Creator user ID (optional, for user-designed puzzles, Null means created by system)

5.1.3 Game History

This collection will store individual game history for users, including:

- Game history ID (unique identifier)
- User ID (associated user)
- Game ID (associated game)
- Timestamp (date and time)
- Completion time
- Score

5.1.4 Leaderboards

This collection will store leaderboard entries, including:

- Leaderboard entry ID (unique identifier)
- Game ID (associated game)
- User ID (associated user, optional for anonymous submissions)
- Score
- Timestamp (date and time)

5.2 Relationships

The following relationships exist among the collections:

- One-to-many relationship between Users and Game History: Each user can have multiple game history entries.
- One-to-many relationship between Games and Game History: Each game can have multiple game history entries.
- One-to-many relationship between Users and Leaderboards: Each user can have multiple leaderboard entries.
- One-to-many relationship between Games and Leaderboards: Each game can have multiple leaderboard entries.

5.3 Indexing

To optimize query performance, we will create indexes on the following fields:

- Users: Username (for quick look-up during login and registration)
- Game History: User ID and Game ID (for efficient retrieval of user game history and game-specific history)
- Leaderboards: Game ID (for quick access to game-specific leaderboards)

6 Testing

TODO Left for QA Engineer, may not have time to do all of this.

6.1 Testing Objectives

- Ensure the application meets functional requirements and specifications.
- Validate the correctness and reliability of the application.
- Identify and resolve any defects or issues before the application is released.

6.2 Testing Levels

6.2.1 Unit Testing

- Test individual components, functions, or modules in isolation.
- Use appropriate testing frameworks for the frontend (e.g., Jest) and backend (e.g., pytest).
- Automate unit tests to facilitate continuous integration and early defect detection.

6.2.2 Integration Testing

- Test the interaction and communication between different components or modules.
- Ensure that the integrated system works correctly and meets the specified requirements.
- Test API endpoints to validate correct data flow between frontend and backend.

6.2.3 System Testing

- Test the entire application as a whole, including frontend, backend, and database.
- Perform end-to-end testing to ensure the system meets functional and non-functional requirements.
- Test user scenarios and workflows to validate the application's overall functionality and usability.

6.3 Testing Methodologies

6.3.1 Manual Testing

- Test the application manually to identify defects, usability issues, and unexpected behavior.
- Perform exploratory testing to uncover issues that may not be detected by automated tests.

6.3.2 Automated Testing

- Develop automated test scripts to validate the application's functionality and performance.
- Integrate automated tests into the continuous integration and deployment pipeline.

6.3.3 Performance Testing

- Test the application's performance under various conditions, such as high load or concurrent users.
- Identify potential bottlenecks and optimize the application for better performance and scalability.

6.4 Testing Tools and Frameworks

- Frontend testing: Jest, React Testing Library, Selenium (for end-to-end testing).
- Backend testing: pytest, Postman (for API testing), locust (for performance testing).

6.5 Test Plan and Schedule

- Collaborate with the QA engineer to develop a comprehensive test plan, including test cases, test data, and expected results.
- Schedule testing phases in coordination with the project timeline and milestones.
- Regularly review test results and address any defects or issues identified during the testing process.

7 Deployment

Describe the deployment process and infrastructure for the application. Specify the hosting environment, such as a cloud provider or a local server, and any necessary steps for deploying and maintaining the application.

TODO We will build, test, and deploy locally. We can consider deploying on simple SaaS platforms such as Digital Ocean or PythonAnywhere if we have time.

8 **Security Considerations**

Elaborate on the security measures employed to protect the application and user data. Discuss authentication, authorization, encryption, and secure communication between frontend and backend.

TODO Implement this section if we have time.

9 Scalability and Performance

Discuss how the application is designed to scale and handle increasing loads. Address the potential bottlenecks and outline possible solutions to improve performance and ensure high availability.

TODO Implement this section if we have time.

10 Monitoring and Maintenance

Describe the monitoring and maintenance strategy, including any tools or frameworks used for monitoring the application's performance, uptime, and security. Explain how the development team will address issues and updates.

TODO Implement this section if we have time.

11 Future Enhancements

List any potential improvements or new features that could be added to the application in the future. This could include new game modes, social features, or additional customization options.

TODO Implement this section if we have time.