ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY (AAiT)

School of Information Technology and Engineering(SiTE)

Distributed Systems
Mini design documentation

Group Members	Id
1. Hiwot Addis	UGR/3763/14
2. Kalkidan T/haimanot	UGR/4332/14
3. Sifen Beshada	UGR/0667/14

Stream: Software

Submitted to: Instructor Wondimagegn D.

Submission Date: 05/02/2025

Scrambled Words - Design Document

Table Of content

1. Introduction

1.1 Project Overview

Scrambled Words is an interactive multiplayer word puzzle game designed to challenge players in a competitive environment. Unlike traditional word games with time limits, this game determines the winner based on who unscrambles the words first and gets to three points. When one player finishes, their score will increase and the next word is presented. The game also incorporates multiple servers that act as backups to ensure uninterrupted gameplay in case of failures.

1.2 Objectives

- ❑ Develop an engaging multiplayer word game.
- ❑ Implement secure authentication and user management.
- ❑ Enable real-time gameplay using WebSockets.
- ❑ Provide a responsive and intuitive UI.
- ❑ Ensure high availability with multiple backup servers.

2. System Architecture

2.1 Overview

The game follows a distributed client-server architecture, where multiple servers operate in parallel to handle game logic and backup redundancy. The frontend communicates with a Go-based backend to process authentication, game logic, and real-time updates. MongoDB is used for persistent storage, and Redis is utilized for caching leaderboards and game states.

2.2 Components

Frontend (Client-Side)

- Built using *HTML, CSS,* and *JavaScript.*
- Manages user interactions and game UI.
- Establishes WebSocket connections for real-time updates.

Backend (Server-Side)

- Developed in *Go*, handling authentication, game logic, and WebSocket communication
- The main server handles authentication, authorization and redirecting traffic to the game servers.
- If one of the game servers is down, the main server detects the failure and redirects traffic to the working server.
- The game state is replicated on both the second and third server to ensure fault tolerance.
- Interfaces with MongoDB for user data and game history.
- Uses Redis to store real-time game states and leaderboards.

Database

- *MongoDB*: Stores user accounts, game sessions, and leaderboards.
- *Redis*: Caches active game states and frequently accessed data for performance optimization.

3. Functional Specifications

3.1 User Authentication

- Users can sign up, log in, and log out securely.
- Passwords are hashed and stored securely in MongoDB.

3.2 Game Mechanics

- Players are presented with scrambled words and must unscramble them
- Points are awarded based on completion order.
- Game states are maintained using WebSockets for a real-time experience.

3.3 Multiplayer Support

- Players compete in real-time matches.
- WebSockets handle synchronized game state updates.
- Scores are updated dynamically and stored in the database.

3.4 Leaderboard System

- The leaderboard reflects the number of matches a user has won.
- Redis caches recent scores to optimize performance.
- Displays real-time rankings to players.

3.5 UI and User Experience

- Responsive design for various screen sizes.
- Interactive design and real-time feedback for a seamless experience.
- Intuitive navigation for easy access to different game settings.

4. Technology Stack

4.1 Programming Languages & Frameworks

- Go: Backend services and WebSocket handling.
- JavaScript: Frontend interactivity and WebSocket communication.
- HTML/CSS: User interface design.

4.2 Databases

- MongoDB: Persistent user data and leaderboard storage.
- Redis: Caching system for quick access to frequently requested data.

4.3 Web Technologies

- WebSockets: Enables real-time game updates.

5. Implementation Details

5.1 Backend Structure

- Controllers: Manage authentication, game state, leaderboard operations and WebSocket communication.
- Routes: Define HTTP and WebSocket endpoints for game interactions.
- Models: Contain Go structs for managing user data and game states.
- Shared: Includes helper functions and reusable utilities.
- Server: Manages multiple backup instances to ensure continuity in case of failure.

- Multiple servers handle game logic and serve as backups, ensuring fault tolerance and uninterrupted operation.

5.2 Frontend Structure

- HTML Templates: Define game screens (login, game, leaderboard, etc.).
- CSS Styling: Enhances user experience with visually appealing elements.
- JavaScript: Manages game logic, user interactions, and WebSocket communication.

6. Challenges and Future Enhancements

6.1 Challenges Faced

- Ensuring real-time synchronization between players.
- Managing server failover mechanisms to handle backup instances.
- Maintaining security in user authentication and session management.

6.2 Future Improvements

- Adding different game modes (e.g., timed challenges, multiplayer tournaments).
- Enhancing UI/UX with animations and accessibility features.
- Integrating AI-powered hints for players needing assistance.

7. Conclusion

Scrambled Words is a feature-rich multiplayer word game combining real-time interactivity, competitive gameplay, and an intuitive user experience. By leveraging multiple backup servers, the game ensures high availability and fault tolerance. Future improvements will focus on expanding game modes, improving UI, and enhancing game mechanics to create a more engaging experience for players worldwide.