**SOFTWARE REQUIREMENTS SPECIFICATIONS**

A video game controller with colorful buttons

Description automatically generated

**Abdulmohaimin Bashir**

**Nusaiba Mekkaoui**

**Muhammad Bilal**

**Mohammed Islam Haji**

**19th May 2024**

**Table of Contents**

1. **Introduction**
   * 1.1 Purpose
   * 1.2 Scope
   * 1.3 Definitions, Acronyms and Abbreviations
   * 1.4 References
   * 1.5 Overview
2. **Overall Description**
   * 2.1 Product Perspective
   * 2.2 Product Functions
   * 2.3 User Classes and Characteristics
   * 2.4 Operating Environment
   * 2.5 Design and Implementation Constraints
   * 2.6 Assumptions and Dependencies
3. **Requirements Specification**
   * 3.1 Functional Requirements
   * 3.2 Non-functional Requirements
   * 3.3 External Interface Requirements
   * 3.4 System Features
4. **Appendices**
   * 4.1 Glossary
   * 4.2 System Models
   * 4.3 Other Supporting Information

**1. Introduction**

**1.1 Purpose**

The purpose of this Software Requirements Specification document is to provide a comprehensive description of the requirements for the Gimmick Games platform. This document will describe the project's purpose, scope, functional and non-functional requirements, and system models to ensure a clear understanding among stakeholders.

**1.2 Scope**

Gimmick Games is an online platform designed to offer two mini games that cater to both casual and experienced gamers. The platform will include essential features such as user authentication, profile management, game selection, score saving, and leaderboards.

**1.4 Definitions, Acronyms and Abbreviations**

* **UML (Unified Modelling Language):** A standardised modelling language used to visualise the design of a system.
* **UI** **(User Interface):** The bridge between users and computers.
* **MongoDB:** A NoSQL document-oriented database program that uses JSON-like documents with optional schemas.
* **NoSQL (Not only SQL):** Is a non-relational database management system that employing flexible data models such as key-value, document, etc, to handle large volumes of unstructured and semi-structured data.
* **MVC** **(Model-View-Controller):** A software architectural pattern used to implement user interfaces, data and controlling logic by emphasizing a separation between the software’s business logic and display.
* **API (Application Programming Interface)**: A set of rules or protocols that let software applications exchange data, features, and functionality.
* **ODM (Object-Document Mapper):** Maps data stored in documents from the database to objects in the application code.
* **JSON (JavaScript Object Notation):** Is a lightweight format for storing and transporting data.
* **EJS (Embedded JavaScript):**  A templating language used to generate HTML markup with plain JavaScript.
* **Node.js:** A JavaScript runtime environment.
* **Express.js:** A web application framework for Node.js.
* **HTML (HyperText Markup Language):** Markup language for creating web applications and pages.
* **CSS (Cascading Style Sheets):** Used to describe the presentation of a document written in HTML or XML.
* **JavaScript:** A scripting or programming language that allows you to implement complex features on web pages.
* **HTTPS (HyperText Transfer Protocol Secure):** Uses encryption for secure communication over a computer network.

**1.5 References**

* Joudeh, K., Riyad, R., & Juady, M. (n.d.). *Software Requirements Specification version 1.0*. Retrieved May 21, 2024, from <https://repository.najah.edu/server/api/core/bitstreams/614de4f7-f3ff-4384-b50f-d7a532b8d3fd/content>
* ‌ <https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database>
* <https://snappify.com/blog/software-requirements-specification-sample>

**1.6 Overview**

In this document there are two more sections. The next section, Overall Description contains the overview of the functionality of the web application and how the users will interact with it. Further, the section also describes the environment in which the system will operate and the constraints on the system implementation and design. Some assumptions and dependencies are also suggested.

The third section of this document, Requirement Specifications is written for both users and system developers in multiple level detail. It also addresses in detail, the functional and Non-functional requirements, external interface requirements and the system features.

**2. Overall Description**

**2.1 Product Perspective**

Gimmick Games is an independent online application that functions as a gaming platform with two unique mini games. The platform has a client-server architecture, with Node.js and Express.js used to build the backend and MongoDB being used for data storage.  
  
HTML, CSS, and JavaScript are used in the development of the frontend, with the Bootstrap framework enhancing the user interface to ensure a responsive and visually appealing experience. EJS (Embedded JavaScript) is also used for rendering dynamic content on the client side.  
  
The application is accessible through modern web browsers, providing a responsive and interactive user experience. The platform allows users to play games directly, and registered users can take advantage of features like score-saving and access to leaderboards that display the best players. To maintain a competitive and compelling environment, user authentication is integrated to guarantee that only registered users can save their scores and view leaderboards.

**2.2 Product Functions**

The following are the main features included in this web app:

* **User Authentication**: The system allows users to securely create accounts in the system, log in, and reset their passwords, ensuring that only authorized users can access the personalized features and data.
* **Profile Management**: Users can view and edit their profile information, ensuring their personal data is accurate and up-to-date.
* **Game Selection**: The system offers users to choose from two mini-games, Snake game and Space invaders game and play them directly on the platform providing a fun and interactive gaming experience.
* **Game Play**:
  + **Snake Game:**
    - **Objective:** The objective of the snake game is to control a snake to eat the food items (the apple) that appear on the screen which causes the snake to grow in length. The game ends if the snake runs into itself or the screen edges.
    - **Mechanics:** The player uses the arrow keys (up, down, left, and right) to control the snake's direction. The snake gains points and grows longer each time it consumes food. As the snake gets longer and takes up less space to move, the game gets harder and harder.
  + **Space Invaders:** 
    - **Objective:** The objective is to control a spaceship at the bottom of the screen to shoot down waves of approaching aliens. The game ends if the aliens reach the bottom of the screen.
    - **Mechanics:** The player uses arrow keys to manoeuvre the spaceship left and right and uses spacebar to shoot. The aliens move horizontally and gradually descend and reverse in movement towards the player.
* **Score Saving and Leaderboards**: The system allows registered users to save their scores and view the leaderboards for each game.

**2.3 User Classes and Characteristics**

There are only three types of users that can interact with this system each with their own distinct purposes. The users are as follows:

* **Guests**: Can play games; however, their score won’t be saved even though they can see the games’ leaderboards.
* **Registered Users**: Can play the games and save their scores. They can also view leaderboards and manage their profiles.
* **Administrators**: The system allows admins to directly access and manage the database, as well as manage the overall platform’s functionality and content.

**2.4 Operating Environment**

The operating environment for Gimmick Games consists of the following components:

* **Client-side**: Modern web browsers e.g. Chrome, Firefox, Safari, Edge.
* **Server-side**: Node.js and Express.js running on a Linux or Windows server.
* **Database**: MongoDB for data storage
* **Frameworks/Libraries**: Bootstrap for frontend UI components

**2.5 Design and Implementation Constraints**

The design and implementation of Gimmick Games platform adheres to the following constraints:

* The platform must stick to the MVC design pattern which will help in organising the code and separate concerns effectively.
* MongoDB will be used as the primary database for storing user and game data. This NoSQL database provides scalability and flexibility appropriate for the requirements of the application.
* Security features are essential and must include passports for authentication and bcrypt for password hashing. These measures will ensure protection of user data and secure access to the platform.
* The platform will integrate with third-party email services to facilitate the password reset functionality, ensuring reliable and efficient email delivery.
* The frontend of the application will be developed using HTML, CSS and JavaScript along with Bootstrap framework to create a responsive and aesthetically pleasing UI.
* The backend of the application will be built using Node.js and Express.js exploiting its event-driven, non-blocking I/O style to effectively manage several concurrent connections.
* The application will communicate with the server and client via RESTful APIs, ensuring a standardised and flexible approach to transfer of data. Integration with the Postman API platform will be implemented to make API development, testing, and documentation easier.

**2.6 Assumptions and Dependencies**

* The application will be deployed and hosted on a local server environment which should be correctly configured and maintained to ensure optimal performance and availability.
* The software requirements, such as Node.js, Express.js, and MongoDB, as well as the relevant versions and configurations needed for the application to run properly, will be installed on the local server environment.
* Users will need to have access to a reliable internet connection to access and engage with the system since the program is a web-based platform and needs internet connection.
* Users will be using modern web browsers that support the latest web standards to ensure a consistent and ideal user experience on different devices.

**3. Requirements Specifications**

**3.1 Functional Requirements**

**3.1.1 User Authentication**

* **Sign Up**: Users must be able to register for an account with an email and password. The system will validate the email format to ensure it is correctly structured and password length.
* **Login**: Registered users can log in with their registered email and password. The system will authenticate the credentials and if the credentials are incorrect, it should display an error message. Users will be directed back to main dashboard once logged in successfully.
* **Password Reset**: Users must be able to reset their password if forgotten by entering their registered email address to request password rest.
* **Unique Username:** The system must check in the database during the registration process that the username selected by the user is unique. If the username already exists, the user will be prompted to choose a different username.

**3.1.2 Profile Management**

* **View Profile**: Registered users can view their profile information, including username, email, and game scores.
* **Edit Profile**: Users can update their profile information in ‘my profile’ page, such as changing their username and password. Users can access this page from the main dashboard.
* **Delete Account**: Users can delete their account from ‘my profile’ page upon a confirmation step.

**3.1.3 Game Selection and Play**

* **Game Selection**: The main page should display two games with their image.
* **Game Play**: Users should be able to select and play any of the two games displayed. The game should load quickly and provide an engaging experience.

**3.1.4 Score Saving and Leaderboards**

* **Save Score**: After completing a game, registered users can save their scores. The system should ensure the score is associated with the correct user.
* **View user scores and Leaderboard**: Each game should have a leaderboard displaying the username, score associated to that user and the date saved. Users should also be able to view their own scores for both games in their profile information page.

**3.1.5 Administrator Functions**

* **Manage Users**: Administrators will have access to the database and can view, edit, and delete user accounts.
* **Manage Games**: Administrators can add, update, or remove games from the platform.

**3.2 Non-functional Requirements**

**3.2.1 Security**

* **Authentication**: The program handles user authentication securely and manages secure session through Passports and Express Sessions.
* **Password Storage**: The system uses bcrypt to hash passwords and store them securely in the database in order to prevent unauthorised access.
* **Data Protection**: HTTPS is used for all data transmissions to protect data integrity and privacy.
* **Safety Requirements**: Ensure data integrity and availability by performing daily backups of the database to prevent data loss and test backup restoration processes regularly to ensure data can be recovered quickly in case of failure.

**3.2.2 Performance**

* **Scalability**: Ensure the platform can handle growth increase of user base by implementing scalable architecture. The system should handle up to 1000 concurrent users without performance degradation.
* **Response Time**: Ensure platform is responsive by optimizing front-end and back-end to minimize latency and shorten load times. Game pages should load within 2 seconds on a stable internet connection.
* **Availability**: Ensure the platform is reliable and consistently accessible. The platform should be available 99.9% of the time.

**3.2.3 Maintainability**

* **Code Structure**: Follow the MVC pattern to separate concerns and improve maintainability.
* **Documentation**: Provide comprehensive documentation for both users and developers by creating detailed documentation and include API documentation, user guides and system architecture overviews to facilitate easier and onboarding system maintenance
* **Testing**: Implement automated testing for critical parts of the system, including unit tests, integration tests and end-to-end tests to ensure the reliability of the system.

**3.2.4 Usability**

* **Interface Design**: The UI should be user-friendly (intuitive and easy to navigate).
* **Accessibility**: Ensure the platform is accessible to users with disabilities by following WCAG 2.1 guidelines.
* **Feedback**: Provide users with clear feedback on their actions, such as form submissions and game interactions.

**3.3 External Interface Requirements**

**3.3.1 User Interfaces**

* **Login Page**: Form for users to enter their email and password to log in.
* **Signup Page**: Form for users to create an account with email, password, and username.
* **Main Page**: Displays the two games with options to select and play.
* **Game Page**: Interface for playing games.
* **My Profile Page**: Interface for users to view and edit their profile information and view their scores for each game.

**3.3.2 Hardware Interfaces**

* Operating system such as Windows, macOS and Linux.
* A browser that supports HTML, CSS & JavaScript.

**3.3.3 Software Interfaces**

* **Client Interface:** Modern web browsers such as Chrome, Firefox, Safar, Edge.
* **Database**: MongoDB will store user data, game data, and scores.
* **Application Runtime**: Node.js runtime environment which provides necessary libraries and environment to run JS on the server side.
* **API Development and Testing**: Postman will be used for developing, documenting and testing the API endpoints.
* **Third-party Services**: Integrate with email services for sending password reset links.

**3.4 System Features**

**3.4.1 User Authentication**

Users can securely log in, sign up, and reset passwords.

**3.4.2 Game Interaction**

Users can select and play games with a responsive and engaging interface.

**3.4.3 Data Management**

Efficiently handle and store user profiles and game scores.

**4. Appendices**

**4.1 Glossaries**

|  |  |
| --- | --- |
| Term | Definition |
| Bcrypt | A password hashing function used to secure passwords before storing them in the database. |
| Database | A database is an organized collection of structured information, or data, typically stored electronically in a computer system. |
| Entities | Are objects that exist in which data can be stored about. |
| Relationships | Define how entities are related to each other. |
| User | A person who interacts with a computer, software application or any other system. |
| An Actor | An entity that interacts with the system, typically a user. |
| Middleware | A software that lies between an operating system and the applications running on it. |
| Stakeholder | Is a person, group or organization with a vested interest, or stake, in the decision-making and activities of a business, organization or project. |
| Bootstrap | Is an open-source CSS framework used for developing responsive and mobile-first websites and web applications. |
| Postman | Is an API platform for building and using APIs. |

**4.2 System Models**

**4.2.1 Use Case Diagram**

* **Actors:** 
  + **Player:** two kinds of players, registered and unregistered player. Both interact with the system to play games and view leaderboards, but only registered players can save scores.
  + **Admin:** Manages user accounts and game content.
* **Use Cases:**
  + **Sign Up:** Allows users to create accounts.
  + **Play Game:** Enables users to play games.
  + **View Score:** Allows users to view their scores in their profile page.
  + **View Profile:** Enables users to view their profile information.

A diagram of a game

Description automatically generated

**4.2.2 Class Diagram**

* **User Class:** Attributes include username, email, password. Methods include registerUser(), loginUser(), resetPassword(), updateUser(), deleteUser().
* **Game Class:** Attributes include gameId, gameName and methods include renderSnakeGame() which starts the game, renderSnakeIndex() which ends the game.
* **Score Class:** Attributes include scoreId, playerId, gameId and methods include saveScore(), getLeaderboard().

A diagram of a computer

Description automatically generated

**4.2.3 Entity-Relationship Diagram**

* **Entities:** Three entities, player, scores and games and their attributes mentioned in the previous class diagram explanation.
* **Relationships:**
  + **Player-Score:** The relationship between these two entities is One-to-many meaning one player can have multiple scores.
  + **Game-Score:** The relationship between these two entities is also One-to-many meaning one game can have multiple scores.

A screenshot of a computer

Description automatically generated

**4.3 Other Supporting Information**

**4.3.1 Software Quality Attributes**

* **Reliability**: The system should have minimal downtime and handle errors gracefully.
* **Scalability**: The architecture should support easy scaling to accommodate more users and games.
* **Maintainability**: The codebase should be well-documented and modular to facilitate easy updates and bug fixes.
* **Usability**: The interface should be intuitive and accessible, providing a positive user experience.