**Software Requirements Specification (SRS)**

**1. Introduction**

**1.1 Objectives**

This document describes the software requirements for the **AI-Powered Chess Game** project. It provides details on the system's features, functionalities, and technical requirements to ensure efficient software development that meets user needs. The objective of this project is to create an engaging and challenging chess experience leveraging artificial intelligence. The AI component will be designed to offer varying levels of difficulty, making the game suitable for beginners and advanced players alike.

Additionally, the project aims to integrate an intuitive user interface that ensures seamless gameplay, both in offline and online modes. Players will be able to track their performance, analyze past games, and improve their skills using built-in AI-powered suggestions and move analysis. By incorporating a robust matchmaking system, the game will facilitate competitive play among users globally, further enhancing the player experience.

**1.2 Project Scope**

The AI-Powered Chess Game is a system that allows users to play chess in different modes:

* **Player vs AI**: Users can compete against an AI opponent with various skill levels.
* **Player vs Player**: Supports playing chess with friends or other opponents on the same device.
* **Online Matchmaking**: Supports finding and pairing online players.

The system consists of three main components:

* **Backend API**: Built with Spring Boot to handle game logic and user management.
* **Frontend User**: A React application for users to play the game.
* **Frontend Admin**: A React application for administrators to manage the system.

**Limitations and Constraints**

* The application requires an internet connection for online play and data updates.
* The AI system must evaluate board positions and generate reasonable moves.
* API latency must be optimized to provide a smooth user experience.

**1.3 Target Audience**

This document is intended for the following groups:

* **Software Developers**: Backend and frontend development teams.
* **Project Managers**: Oversee progress and ensure the document aligns with business requirements.
* **Other Stakeholders**: UI/UX designers, AI specialists, and QA teams.

**1.4 Reference Documents**

The following documents were referenced during the preparation of this SRS:

* Spring Boot Documentation
* React Documentation

**1.5 Summary**

This document outlines the software requirements necessary for developing the **AI-Powered Chess Game** system, including:

* **Key Features**: Playing chess against AI, human opponents, and online matchmaking.
* **Technical Requirements**: Utilization of React, Spring Boot, and a database system.
* **Limitations**: Requires an internet connection, API latency optimization.

The next section of this document will detail the specific system requirements.

**2. High-Level Requirements**

The High-Level Requirements (HLR) define the core functional and non-functional requirements for the AI-Powered Chess Game. These requirements provide a broad and abstract view of the system’s essential features and constraints.

**2.1 Functional Requirements**

These requirements describe the main functionalities that the system must support:

These requirements describe the main functionalities that the system must support:

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirement** | **Notes** | **Category** |
| 1 | The system should allow users to register and log in to their accounts. | Users must provide a valid email. | Technical |
| 2 | Users should be able to play chess against an AI opponent. | AI should support multiple difficulty levels. | Technical |
| 3 | The system should support human vs. human chess matches. | Supports both local and online multiplayer modes. | Technical |
| 4 | The system should include an online matchmaking feature. | Players can find opponents based on skill level. | Business |
| 5 | Players should have access to a game history and move analysis feature. | AI can suggest better moves. | Technical |
| 6 | The UI should be responsive and support multiple devices. | Must be optimized for mobile and desktop. | Technical |

**2.2 Non-Functional Requirements**

These requirements define system performance, security, scalability, and usability constraints:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Requirement | Notes | Category |
| 1 | The system should have a response time of less than 500ms for each move request. | Ensures smooth gameplay. | Performance |
| 2 | The AI engine should evaluate board positions within 1 second. | Optimizes AI decision-making. | Performance |
| 3 | The application should support at least 10,000 concurrent users. | Ensures scalability. | Scalability |
| 4 | The system should encrypt user data to ensure security. | Uses AES-256 encryption. | Security |
| 5 | The matchmaking algorithm should find an opponent within 10 seconds. | Improves user engagement. | Performance |
| 6 | The system should comply with GDPR data protection regulations. | Required for European users. | Compliance |

The next section will define detailed system requirements and design specifications.

**3. Security Requirements**

Security requirements outline measures to protect system integrity, user data, and prevent unauthorized access.

**3.1 Access Control**

* The system must implement role-based access control (RBAC) to define user permissions.
* Multi-factor authentication (MFA) should be required for administrative access.
* Session management should include automatic logout after a period of inactivity.

**3.2 Data Protection**

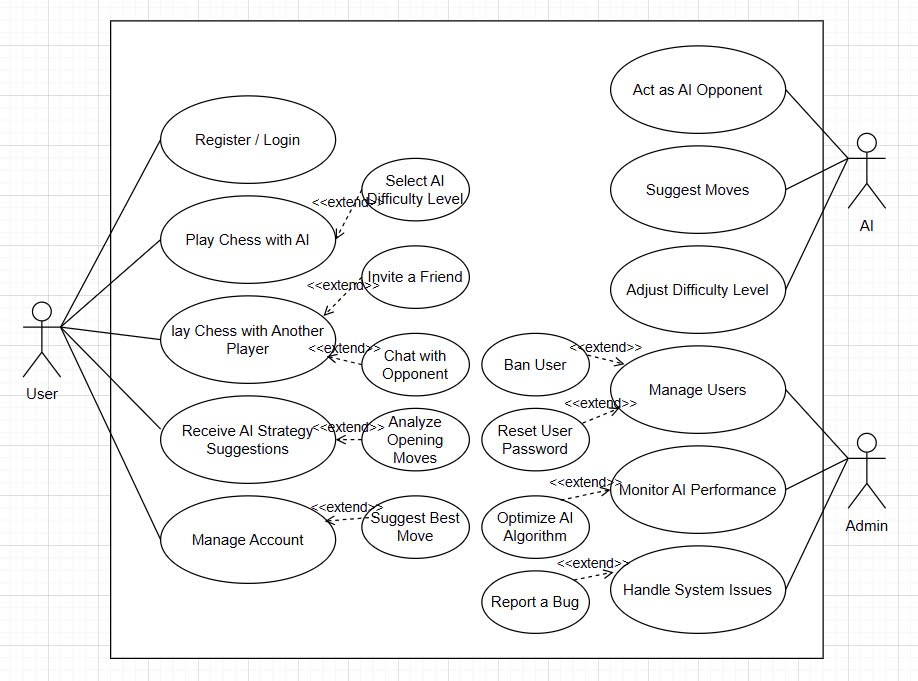
* User passwords must be hashed using a secure hashing algorithm (e.g., bcrypt or Argon2).
* Data in transit must be encrypted using TLS 1.2 or higher.
* Sensitive user data must be encrypted at rest using AES-256.

**3.3 System Security**

* The application should implement protection against common security threats (e.g., SQL injection, cross-site scripting, CSRF attacks).
* Logging and monitoring mechanisms should be in place to detect unauthorized access attempts.
* Security patches and updates must be applied regularly to mitigate vulnerabilities.

The next section will define detailed system requirements and design specifications.

**4. Use Case Specification**



**4.1 Overview**

The Chess Strategy AI application supports users in playing chess against AI or other players. Additionally, the system provides user management, AI optimization, and issue handling features.

**4.2 Use Case List**

Below are the primary use cases of the system:

**4.2.1 Register / Login**

* **ID:** UC01
* **Description:** Users can register a new account or log in to access system features.
* **Preconditions:** The user is not logged in.
* **Steps:**
  1. The user enters login credentials.
  2. The system verifies the credentials.
  3. If valid, the system grants access.
  4. If invalid, an error message is displayed.
* **Expected Outcome:** The user successfully logs into the system.

**4.2.2. Play Chess with AI**

* **ID:** UC02
* **Description:** Users can play chess against AI.
* **Preconditions:** The user is logged in.
* **Steps:**
  1. The user selects the play-against-AI mode.
  2. The system displays the chessboard and starts the game.
  3. The user makes a move.
  4. The AI responds with a move.
  5. The game continues until a winner is determined.
* **Expected Outcome:** The user completes a game against AI.

**4.2.3. Play Chess with Another Player**

* **ID:** UC03
* **Description:** Users can invite friends or be matched randomly to play chess with another human player.
* **Preconditions:** The user is logged in.
* **Steps:**
  1. The user selects the play-with-another-player mode.
  2. The system finds an opponent or accepts an invitation.
  3. Both players take turns making moves.
  4. The game concludes with a win, loss, or draw.
* **Expected Outcome:** The user successfully completes a game against another player.

**4.2.4. Receive AI Strategy Suggestions**

* **ID:** UC04
* **Description:** The system provides move and strategy suggestions.
* **Preconditions:** The user is in an ongoing chess game.
* **Steps:**
  1. The user requests strategy suggestions.
  2. The AI analyzes the board position.
  3. The AI provides the best suggested move.
* **Expected Outcome:** The user receives AI-generated strategy recommendations.

**4.2.5. Manage Account**

* **ID:** UC05
* **Description:** Users can modify their account details.
* **Preconditions:** The user is logged in.
* **Steps:**
  1. The user navigates to the account management section.
  2. The user updates personal information.
  3. The system saves the changes.
* **Expected Outcome:** The user's account details are updated successfully.

**4.2.6. User Management (Admin)**

* **ID:** UC06
* **Description:** Administrators can manage users, including banning accounts.
* **Preconditions:** The user has administrator privileges.
* **Steps:**
  1. The admin accesses the user management interface.
  2. The admin selects a user account for banning or modifying privileges.
  3. The system executes the action.
* **Expected Outcome:** The user is banned, or their information is updated successfully.

**4.2.7. Report a Bug**

* **ID:** UC07
* **Description:** Users can report system bugs.
* **Preconditions:** The user is logged in.
* **Steps:**
  1. The user navigates to the bug report section.
  2. The user describes the issue encountered.
  3. The system logs and forwards the report to the administrator.
* **Expected Outcome:** The bug is reported successfully.

**4.2.8. Monitor AI Performance (Admin)**

* **ID:** UC08
* **Description:** Administrators can monitor AI performance and optimize algorithms.
* **Preconditions:** The user has administrator privileges.
* **Steps:**
  1. The admin accesses the AI performance monitoring interface.
  2. The system displays AI performance metrics.
  3. The admin analyzes the data to improve AI algorithms.
* **Expected Outcome:** The administrator obtains insights for AI optimization.

**4.3. Detailed Use Case Descriptions**

**4.3.1. Use Case: Register / Login**

* **Actors:** User
* **Description:** This use case describes the process of user authentication.
* **Main Flow:**
  1. The user opens the application and selects the login/register option.
  2. The system presents a login form.
  3. The user enters credentials and submits them.
  4. The system validates the credentials.
  5. If valid, the user gains access to the system.
  6. If invalid, an error message is displayed.
* **Alternate Flow:**
  1. If the user forgets the password, they can request a reset link.
* **Postconditions:**
  1. The user is successfully authenticated and redirected to the main dashboard.

**4.3.2. Use Case: Play Chess with AI**

* **Actors:** User, AI
* **Description:** This use case describes the interaction between the user and AI during a chess game.
* **Main Flow:**
  1. The user selects "Play with AI" mode.
  2. The system initializes the chessboard.
  3. The user makes the first move.
  4. The AI evaluates and responds with a move.
  5. The game continues until checkmate, stalemate, or resignation.
* **Alternate Flow:**
  1. The user can request a hint from AI.
  2. The user can adjust AI difficulty.
* **Postconditions:**
  1. The game ends with a winner or a draw.

**4.3.3. Use Case: Manage Users (Admin)**

* **Actors:** Administrator
* **Description:** This use case describes how an admin manages user accounts.
* **Main Flow:**
  1. The admin logs into the system and navigates to the user management section.
  2. The system displays a list of users.
  3. The admin selects a user to ban or modify access rights.
  4. The system processes the changes.
* **Alternate Flow:**
  1. The admin can search for users based on email or username.
* **Postconditions:**
  1. The targeted user is banned or their information is updated successfully.

**Other Requirement**