# Section 1: Introduction

1.1 Purpose:  
This document defines the requirements for a Qt-based Tic Tac Toe desktop application with secure user login, AI gameplay, and game history tracking. It serves as a reference for developers and stakeholders to ensure alignment with system functionality and performance goals.

1.2 Scope:  
The system allows users to register, log in securely, and play Tic Tac Toe in Player vs Player (PvP) or Player vs AI (PvAI) mode. It features three AI difficulty levels and saves each user's game history using a structured JSON format.

Main features include:

* Secure authentication using SHA-256
* Multiple gameplay modes
* AI logic based on Minimax algorithm
* Persistent game data and statistics
* User-friendly Qt graphical interface

## 1.3 Definitions, Acronyms, and Abbreviations:

* Qt: Cross-platform GUI toolkit used for building the application interface
* SHA-256: Secure Hash Algorithm used to hash user passwords
* PvP: Player vs Player mode
* PvAI: Player vs Artificial Intelligence mode
* JSON: JavaScript Object Notation, used for storing user and game data

## 1.4 References:

* Qt Documentation: https://doc.qt.io
* C++ ISO Standard: https://isocpp.org
* OWASP Password Storage Cheat Sheet: https://cheatsheetseries.owasp.org

1.5 Overview:  
This SRS outlines the overall system behavior, including functional and non-functional requirements, performance goals, and interface descriptions. It provides a structured overview for development and testing teams to implement the application effectively.

# Section 2: Overall Description

2.1 Product Perspective:  
This application is a standalone desktop game built using the Qt framework. It stores all data locally in JSON format and does not require any external services or internet access. The architecture follows a modular and layered approach using the Model-View-Controller (MVC) pattern.

## 2.2 Product Functions:

* User registration and login with password hashing
* Mode selection: PvP or PvAI
* AI selection: Easy, Medium, or Hard difficulty
* Real-time game interaction with win/draw detection
* Save game result and timestamp in user history
* View past game history

## 2.3 User Classes and Characteristics:

* Regular User: Can register, log in, play games, and view history
* Administrator: (Optional extension) May be added to manage data or users in future versions

## 2.4 Operating Environment:

* Platform: Windows, Linux (Desktop)
* Language: C++
* Framework: Qt 6+
* Data Storage: JSON files

## 2.5 Design and Implementation Constraints:

* Application must operate offline
* All passwords must be securely hashed (SHA-256)
* JSON file must be structured and human-readable

## 2.6 User Documentation:

* A user manual or help section may be provided within the GUI in future updates.

## 2.7 Assumptions and Dependencies

* User will have a compatible operating system (Windows/Linux)
* Application assumes local file permissions are granted for reading/writing JSON files
* No network connectivity is required

# Section 3: Specific Requirements

3.1 Functional Requirements:  
FR1. User Registration:

* The system shall allow new users to register using a unique username and password.
* The password must be hashed using SHA-256 before storage.

#### FR2. User Authentication:

* The system shall verify user credentials during login.
* The system shall deny access if authentication fails.

#### FR3. Game Mode Selection:

* After login, the user shall choose between PvP and PvAI modes.

#### FR4. AI Difficulty Selection:

* If PvAI is selected, the system shall allow the user to choose the difficulty level: Easy, Medium, or Hard.

#### FR5. Gameplay:

* The system shall display a 3x3 grid for gameplay.
* The system shall alternate turns and update the board accordingly.
* The system shall detect win, loss, or draw conditions.
* The system shall prevent moves on already played cells.

#### FR6. Game History Tracking:

* The system shall store the result of each game (win/loss/draw), timestamp, mode, and difficulty in the JSON file.
* The system shall allow the user to view their game history.

#### FR7. Navigation:

* The system shall allow users to navigate between login, home, game, and history windows.
* The user shall be able to log out or quit the application at any point.

#### FR8. Error Handling:

* The system shall display appropriate error messages for invalid inputs or failed actions.
* The system shall prevent the application from crashing under unexpected user behavior.

#### FR9. Game Rules:

* The game shall follow classic Tic Tac Toe rules: 3 symbols in a row (horizontal, vertical, or diagonal) results in a win.
* If the grid is full and no player wins, the game is a draw.

#### FR10. Data Structure Format:

* User and game data shall be stored in a JSON format with clearly defined keys for username, password hash, history, and game logs.

#### FR11. UI Behavior:

* The system shall disable buttons for clicked cells.
* The current player turn shall be displayed and updated after each move.
* The restart button shall reset the game board without restarting the application.

3.2 Non-Functional Requirements:  
NFR1. Performance:

* The application shall respond to user actions within 0.5 seconds.
* AI calculations (even Hard mode) shall complete within 1 second.

#### NFR2. Security:

* All user passwords shall be hashed before storage using SHA-256.
* No plain text passwords shall be stored.
* Input validation shall be enforced to prevent injection or invalid data.

#### NFR3. Usability:

* The application shall provide a clear and user-friendly interface.
* Buttons and labels shall be clearly named and intuitive.

#### NFR4. Portability:

* The application shall run on Windows and Linux desktop environments.

#### NFR5. Reliability:

* The application shall not crash under normal usage.
* JSON data shall be backed up automatically on changes (optional extension).

#### NFR6. Maintainability:

* Code shall be modular and well-documented.
* Components shall follow object-oriented best practices.

## 3.3 Game Rules:

* The game is played on a 3x3 grid.
* Two players take turns marking cells with X or O.
* A player wins by placing three of their marks in a horizontal, vertical, or diagonal row.
* If all cells are filled without a winning combination, the game ends in a draw.
* AI logic varies by difficulty level and simulates strategic gameplay.

## 3.4 Performance Requirements:

* Game interface should respond instantly to user interactions.
* AI response time shall be:

1. Easy: ≤ 0.2 seconds
2. Medium: ≤ 0.5 seconds
3. Hard: ≤ 1 second

* Application launch time: ≤ 2 seconds

## 3.5 Software System Attributes:

* Security: Passwords are hashed with SHA-256. No sensitive data is stored in plain text.
* Reliability: System handles incorrect input and prevents crashes.
* Usability: Clear layout, responsive design, and error messages guide user behavior.
* Maintainability: Modular structure (e.g., separate UserManager, GameWindow classes) makes future changes simple.
* Scalability: New features (e.g., additional game modes) can be added without major redesign.
* Portability: Cross-platform support for Windows and Linux.

# Section 4: External Interface Requirements

## 4.1 User Interfaces:

* MainWindow: Login and registration screen with form fields and buttons.
* RegisterWindow: Interface for creating a new account.
* HomeWindow: Mode selection and navigation hub.
* GameWindow: Interactive Tic Tac Toe grid, restart/back buttons, and turn indicator.
* HistoryWindow: Displays list of past game results with timestamps.

## 4.2 Hardware Interfaces:

* Not applicable. The application does not depend on any external hardware.

## 4.3 Software Interfaces:

* Qt Framework: GUI, event handling, and window management.
* JSON Files: Local storage of user credentials and game history.
* QCryptographicHash: Used for hashing passwords.

## 4.4 Communication Interfaces:

* Not applicable. The system operates entirely offline and does not use network communication.

# Section 5: Other Requirements

## 5.1 Data Security:

* Passwords are encrypted using SHA-256 hashing before storage.
* Input fields are sanitized to prevent invalid entries or crashes.
* All user and game data is saved locally in structured JSON format with human-readable keys.
* No sensitive data is transmitted or stored in plain text.

## 5.2 Data Integrity and Backup:

* Game results and user data are saved immediately after relevant actions (e.g., end of game, registration).
* The system ensures data persistence by safely writing to the JSON file upon every update.
* An optional extension may include automatic backup of the user data file with timestamps.

## 5.3 Error Recovery and Stability:

* The system is designed to gracefully handle incorrect or unexpected input.
* If a file read/write error occurs, the user is notified without application crash.
* All interactions are protected using checks to ensure stability and prevent memory issues.

## 5.4 Compliance and Best Practices:

* The system follows best practices in secure user authentication and GUI design.
* Password storage follows OWASP standards for desktop applications.
* Codebase adheres to object-oriented principles and Qt framework guidelines.

**[Section 1: Introduction 1](#_Toc11769)**

[1.1 Purpose: 1](#_Toc13532)

[1.2 Scope: 1](#_Toc32190)

[1.3 Definitions, Acronyms, and Abbreviations: 1](#_Toc3617)

[1.4 References: 1](#_Toc20687)

[1.5 Overview: 1](#_Toc20083)

**[Section 2: Overall Description 2](#_Toc24817)**

[2.1 Product Perspective: 2](#_Toc12773)

[2.2 Product Functions: 2](#_Toc26028)

[2.3 User Classes and Characteristics: 2](#_Toc22176)

[2.4 Operating Environment: 2](#_Toc7768)

[2.5 Design and Implementation Constraints: 2](#_Toc31011)

[2.6 User Documentation: 3](#_Toc3547)

[2.7 Assumptions and Dependencies 3](#_Toc28566)

**[Section 3: Specific Requirements 4](#_Toc32046)**

[3.1 Functional Requirements: 4](#_Toc17591)

[3.2 Non-Functional Requirements: 5](#_Toc17179)

[3.3 Game Rules: 6](#_Toc5926)

[3.4 Performance Requirements: 6](#_Toc720)

[3.5 Software System Attributes: 7](#_Toc18461)

**[Section 4: External Interface Requirements 8](#_Toc24645)**

[4.1 User Interfaces: 8](#_Toc7381)

[4.2 Hardware Interfaces: 8](#_Toc24701)

[4.3 Software Interfaces: 8](#_Toc20521)

[4.4 Communication Interfaces: 8](#_Toc7470)

**[Section 5: Other Requirements 9](#_Toc15922)**

[5.1 Data Security: 9](#_Toc28250)

[5.2 Data Integrity and Backup: 9](#_Toc25080)

[5.3 Error Recovery and Stability: 9](#_Toc463)

[5.4 Compliance and Best Practices: 9](#_Toc23911)