# Software Design Specification (SDS) for Tic-Tac-Toe Game

## 1. Introduction

### 1.1 Purpose

This document describes the design of the Tic-Tac-Toe game application. It elaborates on the architectural design, component design, database design, and user interface design. The purpose of this SDS is to provide a detailed blueprint for the implementation of the software, ensuring that all requirements outlined in the Software Requirements Specification (SRS) are met.

### 1.2 Scope

The scope of this document covers the high-level and detailed design of the Tic-Tac-Toe game application. It includes the breakdown of the system into modules, the design of each module, the data structures used, and the interactions between different components. This document does not cover the specific implementation details at the code level, but rather the design decisions that guide the coding process.

### 1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **SDS**: Software Design Specification
- **UI**: User Interface
- **AI**: Artificial Intelligence
- **MVC**: Model-View-Controller
- **Qt**: A cross-platform application development framework.

### 1.4 References

- IEEE Std 1016-2009, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions.
- Software Requirements Specification (SRS) for Tic-Tac-Toe Game (refer to srs.md).

### 1.5 Overview

This document is structured to provide a comprehensive design overview. Section 1 introduces the purpose, scope, definitions, references, and overall structure of the SDS. Section 2 details the architectural design, including the system decomposition and design principles. Section 3 focuses on the component design, describing each major module and its functionalities. Section 4 outlines the database design, including entity-relationship diagrams and schema. Section 5 covers the user interface design, detailing the layout and interaction flows.

## 2. Architectural Design

### 2.1 System Decomposition

The Tic-Tac-Toe game application will follow a layered architectural pattern, with a clear separation of concerns to enhance maintainability, scalability, and testability. The main layers identified are:

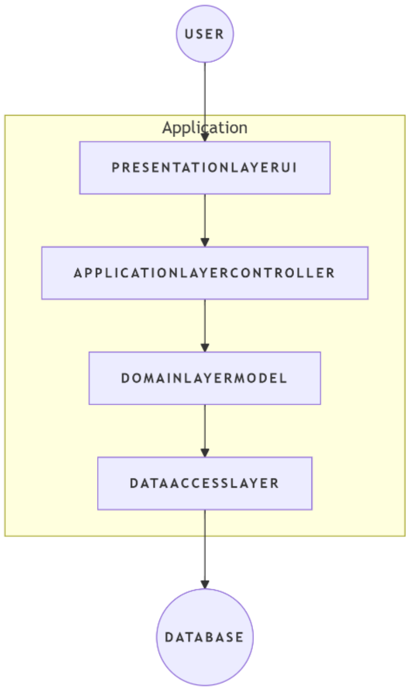
- **Presentation Layer (View)**: Responsible for displaying the user interface and handling user interactions. This layer will be implemented using Qt widgets and QML (if applicable).

- **Application Layer (Controller)**: Manages the flow of the application, processes user input, and interacts with the Model layer. This layer will contain the logic for game flow, user authentication, and navigation between different screens.
- **Domain Layer (Model)**: Encapsulates the core business logic of the Tic-Tac-Toe game, including game rules, AI logic, user data management, and game statistics. This layer is independent of the UI and data storage mechanisms.
- **Data Access Layer**: Provides an abstraction over the underlying database, handling data storage and retrieval operations for user profiles and game history. This layer will interact with the SQLite database.

### 2.2 Design Principles

- **Modularity**: The system will be decomposed into small, independent modules with well-defined interfaces to promote reusability and reduce coupling.
- **Separation of Concerns**: Each layer and module will have a single, well-defined responsibility.
- **Extensibility**: The design will allow for easy addition of new features, such as new game modes or AI algorithms, without significant modifications to existing code.
- **Testability**: The modular design will facilitate unit testing of individual components.
- **Usability**: The user interface will be designed to be intuitive and user-friendly.

### 2.3 High-Level Component Diagram



- **Presentation Layer**: Consists of `MainWindow`, `LoginPage`, `SignUpForm`, `GameModeWindow`, `PlayerSelection`, `GameplayWindow`, `ChooseDifficulty`, `GameHistory`, `StatisticsWindow`, and `UserProfile` classes, which are responsible for rendering the UI and capturing user input.
- **Application Layer**: Handles the navigation between different windows, manages user sessions, and orchestrates interactions between the Presentation and Domain layers. Classes like `main.cpp` (application entry point) and potentially `App` (if it acts as a central application manager) would reside here.
- **Domain Layer**: Contains the core game logic (`TicTacToeWidget`), AI logic, user management logic, and data structures for game state. Classes like `gameChoices`, `tictactoewidget`, and potentially parts of `userprofile` (for business logic related to user data) would be in this layer.
- **Data Access Layer**: Responsible for interacting with the `TicTacBoom.db` or `TicTacToe.db` database. This layer would contain classes or functions for database connection, querying, and updating user and game data.

## 3. Component Design

This section details the design of the major components (modules) within the Tic-Tac-Toe application. Each component is described in terms of its responsibilities, interfaces, and interactions with other components.

### 3.1 User Interface Components

The user interface components are primarily implemented using Qt widgets and UI files. These components are responsible for presenting information to the user and capturing user input.

- `MainWindow`: The main application window, serving as the entry point for the user interface. It might contain navigation to other main sections like login, game mode selection, or user profile.
  - **Responsibilities**: Application initialization, main menu display, navigation.
  - **Interfaces**: Connects to `LoginPage`, `GameModeWindow`, `UserProfile`.
- `LoginPage`: Handles user login functionality.
  - **Responsibilities**: Displaying login form, validating user credentials, authenticating users.
  - **Interfaces**: Accepts username and password input, interacts with the Data Access Layer for authentication, navigates to `MainWindow` or `SignUpForm`.
- `SignUpForm`: Manages new user registration.
  - **Responsibilities**: Displaying registration form, validating new user data, creating new user accounts.
  - **Interfaces**: Accepts new username and password, interacts with the Data Access Layer for user creation, navigates back to `LoginPage` upon successful registration.
- `GameModeWindow`: Allows the user to select between different game modes (Player vs. Player, Player vs. AI).
  - **Responsibilities**: Presenting game mode options, capturing user selection.
  - **Interfaces**: Navigates to `PlayerSelection` (for Player vs. Player) or `ChooseDifficulty` (for Player vs. AI).
- `PlayerSelection`: For Player vs. Player mode, allows input of player names.
  - **Responsibilities**: Capturing names for Player 1 and Player 2.
  - **Interfaces**: Navigates to `GameplayWindow` with selected player names.
- `ChooseDifficulty`: For Player vs. AI mode, allows the user to select the AI difficulty.
  - **Responsibilities**: Presenting AI difficulty options (Easy, Medium, Hard), capturing user selection.
  - **Interfaces**: Navigates to `GameplayWindow` with the selected AI difficulty.
- `GameplayWindow`: The core game interface where Tic-Tac-Toe is played.
  - **Responsibilities**: Displaying the 3x3 game board, handling player moves, updating game state, displaying win/draw messages, providing game reset functionality.
  - **Interfaces**: Interacts with `TicTacToeWidget` for game logic, displays game outcome, allows game reset.
- `GameHistory`: Displays a list of past games and their outcomes.
  - **Responsibilities**: Retrieving and displaying game history for the logged-in user.
  - **Interfaces**: Interacts with the Data Access Layer to fetch game history.
- `StatisticsWindow`: Shows player statistics (wins, losses, draws).
  - **Responsibilities**: Retrieving and displaying aggregated player statistics.
  - **Interfaces**: Interacts with the Data Access Layer to fetch statistics.
- `UserProfile`: Allows users to view and modify their profile information.
  - **Responsibilities**: Displaying user details, enabling password changes or other profile updates.
  - **Interfaces**: Interacts with the Data Access Layer for profile management.

### 3.2 Game Logic Components

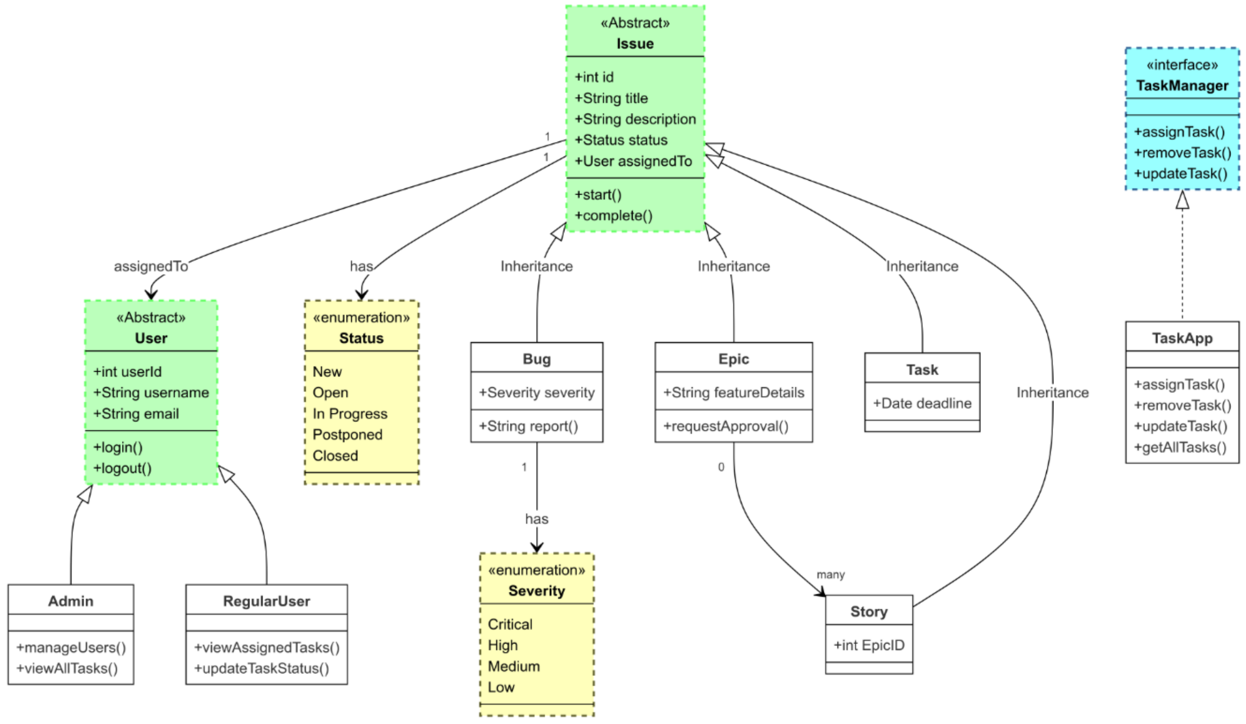
- `TicTacToeWidget`: Encapsulates the core game logic for Tic-Tac-Toe.
  - **Responsibilities**: Managing the 3x3 game board state, validating moves, detecting win conditions (rows, columns, diagonals), detecting draw conditions, managing player turns.
  - **Interfaces**: Provides methods for making moves, checking game status, resetting the board. It is likely integrated into `GameplayWindow`.

- **AI Logic (within `TicTacToeWidget` or a separate class)**: Handles the AI opponent's moves.
  - **Responsibilities**: Implementing different AI strategies (Easy, Medium, Hard).
    - **Easy**: Random valid move selection.
    - **Medium**: Basic strategy (block immediate wins, prioritize center, corners).
    - **Hard**: Minimax algorithm or similar optimal strategy.
  - **Interfaces**: Receives current board state, returns the AI's chosen move.
- **`gameChoices`**: This class appears to be related to managing game options or settings, possibly difficulty levels or player types.
  - **Responsibilities**: Storing and providing access to game configuration choices.
  - **Interfaces**: Used by `GameModeWindow` and `ChooseDifficulty`.

### 3.3 Data Management Components

- **Database Interaction Module (e.g., `DatabaseManager` or similar functions)**: This component is responsible for all interactions with the SQLite database.
  - **Responsibilities**: Establishing database connection, executing SQL queries (INSERT, SELECT, UPDATE), handling transactions, managing user data (registration, login, profile updates), storing game history, retrieving statistics.
  - **Interfaces**: Provides APIs for `LoginPage`, `SignUpForm`, `UserProfile`, `GameHistory`, `StatisticsWindow` to store ### 3.4 Class Diagram (Detailed)

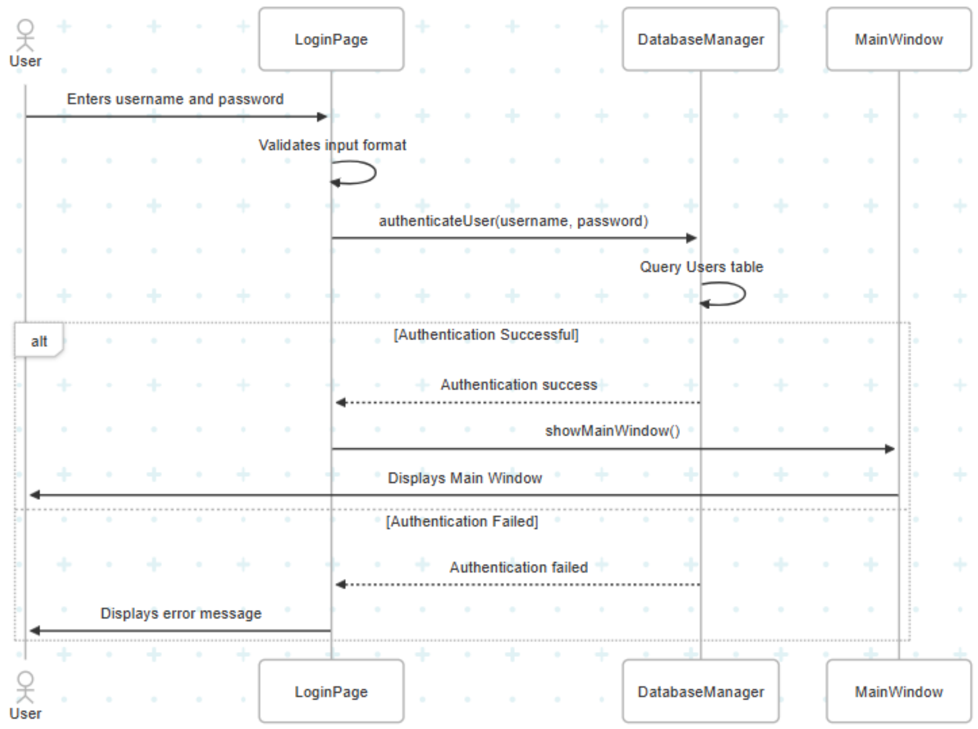
## 3.4 Class Diagram (Conceptual)



## 3.5 Sequence Diagrams

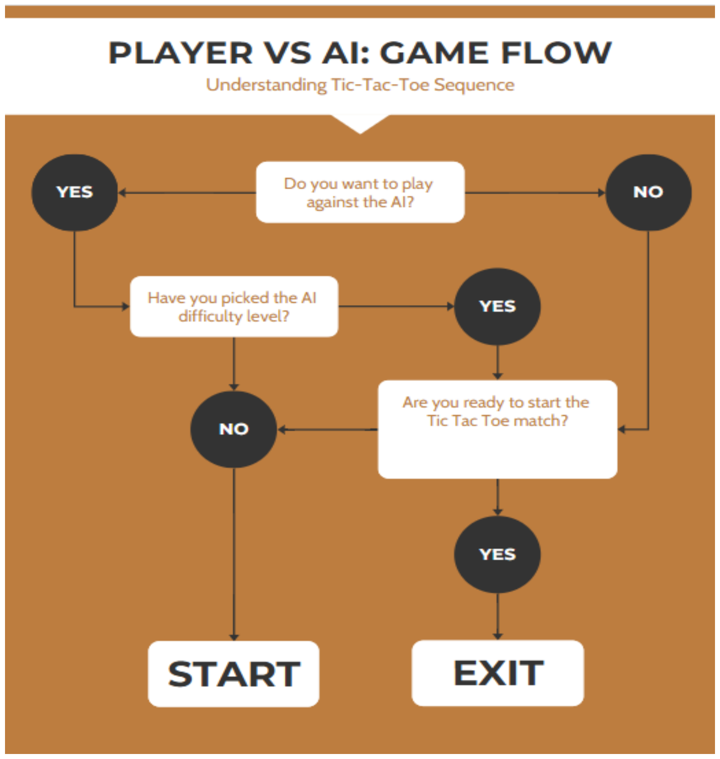
### *3.5.1 User Login Sequence*

This sequence diagram illustrates the flow of events when a user attempts to log in to the application.



### *3.5.2 Player vs. AI Game Flow Sequence*

This sequence diagram illustrates the interaction during a Player vs. AI game session.


PLAYER VS AI: GAME FLOW
Understanding Tic-Tac-Toe Sequence
YES
Do you want to play against the AI?
NO
Have you picked the AI difficulty level?
YES
NO
Are you ready to start the Tic Tac Toe match?
YES
START
EXIT

## 4.2 Database Schema

### *Table: `Users`*

This table stores information about registered users.

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| `username` | `TEXT` | `PRIMARY KEY`, `NOT NULL`, `UNIQUE` | Unique identifier for the user. |
| `password_hash` | `TEXT` | `NOT NULL` | Hashed password of the user. |
| `salt` | `TEXT` | `NOT NULL` | Salt used for password hashing. |
| `total_games_played` | `INTEGER` | `DEFAULT 0` | Total number of games played by the user. |
| `wins` | `INTEGER` | `DEFAULT 0` | Number of games won by the user. |
| `losses` | `INTEGER` | `DEFAULT 0` | Number of games lost by the user. |
| `draws` | `INTEGER` | `DEFAULT 0` | Number of games drawn by the user. |

### *Table: `GameHistory`*

This table stores records of each completed game.

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| `game_id` | `INTEGER` | `PRIMARY KEY`, `AUTOINCREMENT` | Unique identifier for each game. |
| `player1_username` | `TEXT` | `NOT NULL`, `FOREIGN KEY REFERENCES Users(username)` | Username of the first player. |
| `player2_username` | `TEXT` | `NOT NULL`, `FOREIGN KEY REFERENCES Users(username)` | Username of the second player (can be 'AI'). |
| `winner_username` | `TEXT` | `NULLABLE`, `FOREIGN KEY REFERENCES Users(username)` | Username of the winner, or NULL if draw. |
| `outcome` | `TEXT` | `NOT NULL` | Outcome of the game (e.g., 'Win', 'Loss', 'Draw'). |
| `game_date` | `DATETIME` | `NOT NULL`, `DEFAULT CURRENT_TIMESTAMP` | Timestamp when the game was played. |

## 4.3 Data Access Operations

The Data Access Layer will provide the following key operations:

- **User Management**:
  - `insertUser(username, password_hash, salt)`: Adds a new user to the `Users` table.
  - `getUser(username)`: Retrieves user details by username.
  - `updateUser(username, new_data)`: Updates user profile information.
  - `updateUserStatistics(username, outcome)`: Increments win/loss/draw counts for a user.
- **Game History Management**:
  - `insertGame(player1, player2, winner, outcome)`: Records a new game in the `GameHistory` table.

- `getGameHistory(username)`: Retrieves all game records for a specific user.

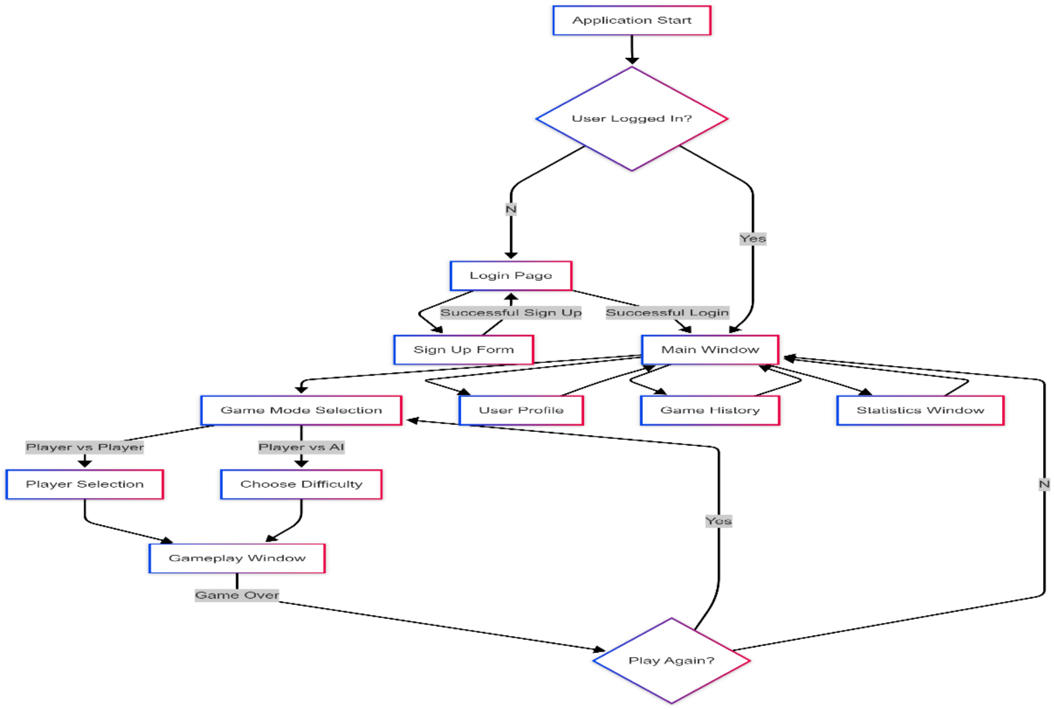
- **Statistics Retrieval**:
  - `getOverallStatistics(username)`: Retrieves aggregated statistics (total games, wins, losses, draws) for a user.

## 5. User Interface Design

The user interface (UI) of the Tic-Tac-Toe game is designed to be intuitive, visually appealing, and easy to navigate. The design principles focus on clarity, consistency, and responsiveness to user actions. The application will utilize Qt's widget-based UI system, ensuring a native look and feel across different operating systems.

### 5.1 Screen Flow

The application's screen flow is designed to guide the user through various functionalities, from initial launch to gameplay and statistics viewing. The primary screens and their transitions are as follows:



### 5.2 Wireframes/Layouts (Conceptual)

While detailed wireframes are not generated here, the conceptual layouts for key screens are described below:

#### *5.2.1 Login Page*

- **Layout**: Centered form with fields for Username and Password.
- **Elements**: Text input fields for username and password, Login button, Sign Up button.
- **Interactions**: Clicking Login attempts authentication; clicking Sign Up navigates to the Sign Up Form.

#### *5.2.2 Sign Up Form*

- **Layout**: Similar to Login Page, with additional fields for password confirmation.
- **Elements**: Text input fields for username, password, confirm password, Sign Up button, Back to Login button.

- **Interactions**: Clicking Sign Up attempts user registration; clicking Back to Login navigates to the Login Page.

### *5.2.3 Game Mode Selection*

- **Layout**: Central area with two prominent buttons.
- **Elements**: “Player vs. Player” button, “Player vs. AI” button.
- **Interactions**: Clicking a button navigates to the respective game setup screen.

### *5.2.4 Player Selection*

- **Layout**: Input fields for two player names.
- **Elements**: Text input for Player 1 Name, Text input for Player 2 Name, Start Game button.
- **Interactions**: Clicking Start Game initiates a Player vs. Player game.

### *5.2.5 Choose Difficulty*

- **Layout**: Radio buttons or a dropdown for difficulty selection.
- **Elements**: Radio buttons for “Easy”, “Medium”, “Hard” AI, Start Game button.
- **Interactions**: Selecting a difficulty and clicking Start Game initiates a Player vs. AI game.

### *5.2.6 Gameplay Window*

- **Layout**: Central 3x3 grid representing the Tic-Tac-Toe board. Areas for displaying current player, game messages (win/draw), and a reset button.
- **Elements**: 9 clickable cells for the game board, text display for current turn, text display for game status (e.g., “X wins!”, “Draw!”), Reset Game button, Back to Main Menu button.
- **Interactions**: Clicking an empty cell places a mark; game status updates dynamically; Reset Game clears the board; Back to Main Menu navigates to the main application window.

## 6. Conclusion

This Software Design Specification provides a detailed design for the Tic-Tac-Toe game application, covering its architecture, component breakdown, database schema, and user interface flow. The design aims to create a modular, maintainable, and user-friendly application that fulfills the requirements outlined in the SRS. This document serves as a guide for the development team during the implementation phase, ensuring a consistent and well-structured approach to building the software. .