Software Design Specification (SDS) for Tic-Tac-Toe:

1. Overview

The SDS provides a detailed design description for the tic-tac-toe game, outlining the architecture, components, interfaces, and interactions necessary for its implementation.

2. Architectural Design

2.1 System Architecture

The tic-tac-toe game will follow a client-server architecture for both local and online multiplayer modes. The client side will handle the graphical user interface (GUI), user inputs, and game logic. The server side will manage multiplayer sessions, game state synchronization, and AI opponent interactions (if applicable).

2.2 Component Diagram

-Client Components:

-GUI Module:

Responsible for displaying the game board, player turns, and game status.

-Game Logic Module:

Implements the tic-tac-toe rules, validates moves, checks win conditions, and manages game state.

- Network Module (for online play):

Facilitates communication with the server, handles multiplayer sessions, and manages data synchronization.

- Server Components (for online play):

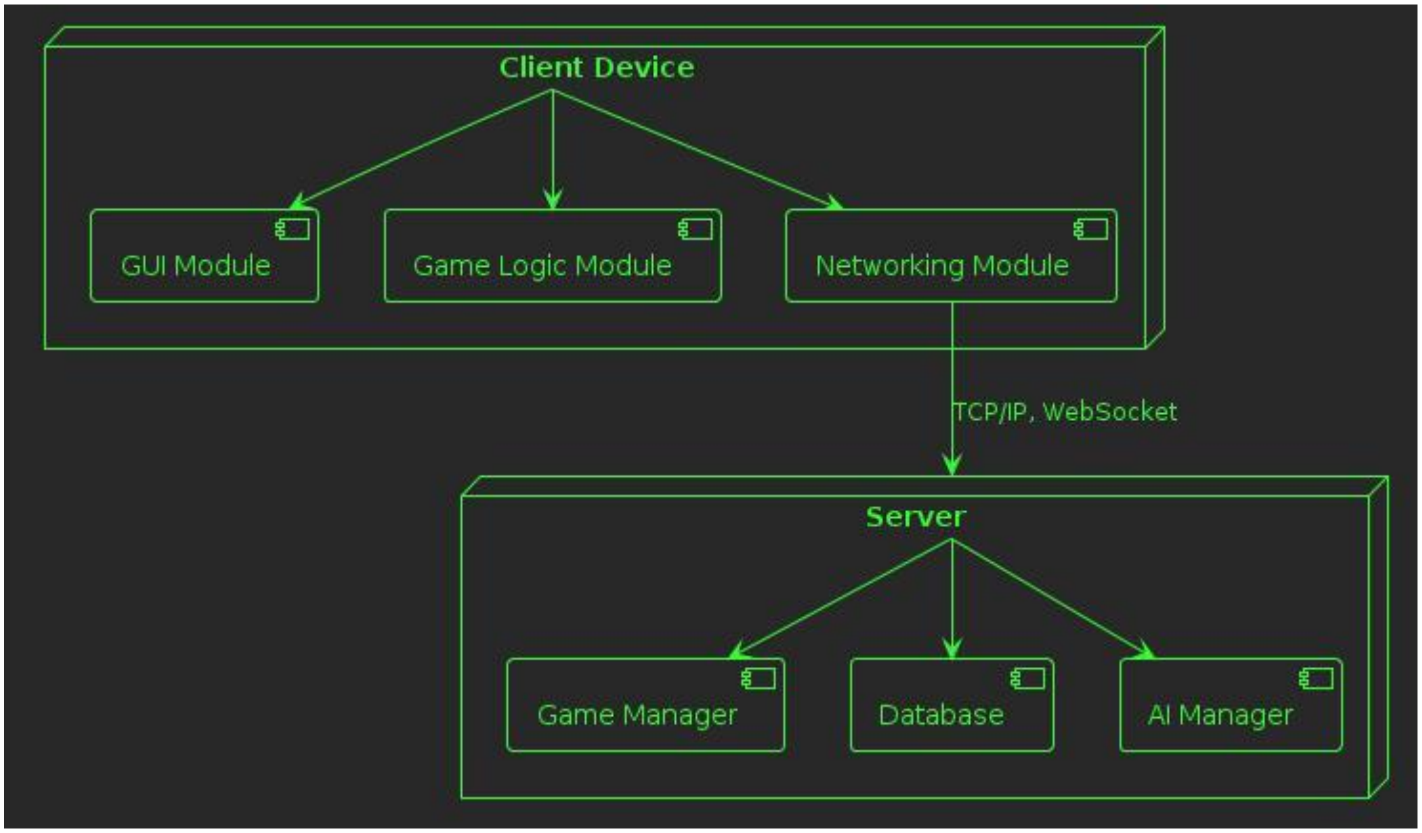
-Game Manager:

Manages game sessions, player connections, and game state synchronization.

-Database Interface:

Stores player profiles, game history, and session data.

2.3 Deployment Diagram:



-Client Side:

Runs on desktop and mobile platforms (Windows, macOS, iOS, Android) with appropriate GUI components.

- Server Side (for online play):

Hosted on a cloud platform (AWS, Azure) with scalable server instances.

3. Detailed Design

3.1 GUI Design

-Game Board:

Grid layout for tic-tac-toe cells, interactive buttons for player moves.

- Player Turn Indicator:

Displays current player turn (X or O).

-Game Status Display:

Shows game results (win, lose, draw) and prompts for new games or quitting.

3.2 Game Logic Design

-Game Initialization:

Sets up the initial game state (empty board, starting player).

-Move Validation:

Checks if a move is valid (cell is empty).

-Win Condition Check:

Verifies if a player has achieved a winning pattern (horizontal, vertical, diagonal).

- Game State Management: Handles game progression, switching turns, and determining game outcomes.

3.3 Network Communication (for online play)

- Client-Server Interaction:

Utilizes TCP/IP or WebSocket protocols for real-time game synchronization.

-Session Management:

Establishes and maintains connections between players and the server.

- Data Serialization:

Formats game state data for transmission between client and server.

4. Implementation Constraints

- Platform Compatibility:

Ensures compatibility with multiple platforms (desktop, web, mobile).

5. Testing and Validation

-Unit Testing:

Tests individual components (GUI, game logic, AI) for correctness and reliability.

-Integration Testing:

Verifies interactions between modules (GUI with game logic, client-server communication).

-User Acceptance Testing:

Involves end-users to validate usability, functionality, and adherence to requirements.

6. Documentation

-User Manual:

Provides instructions on how to play the game, controls, and options.

-Developer Documentation:

Includes API references, system architecture diagrams, and design rationale.

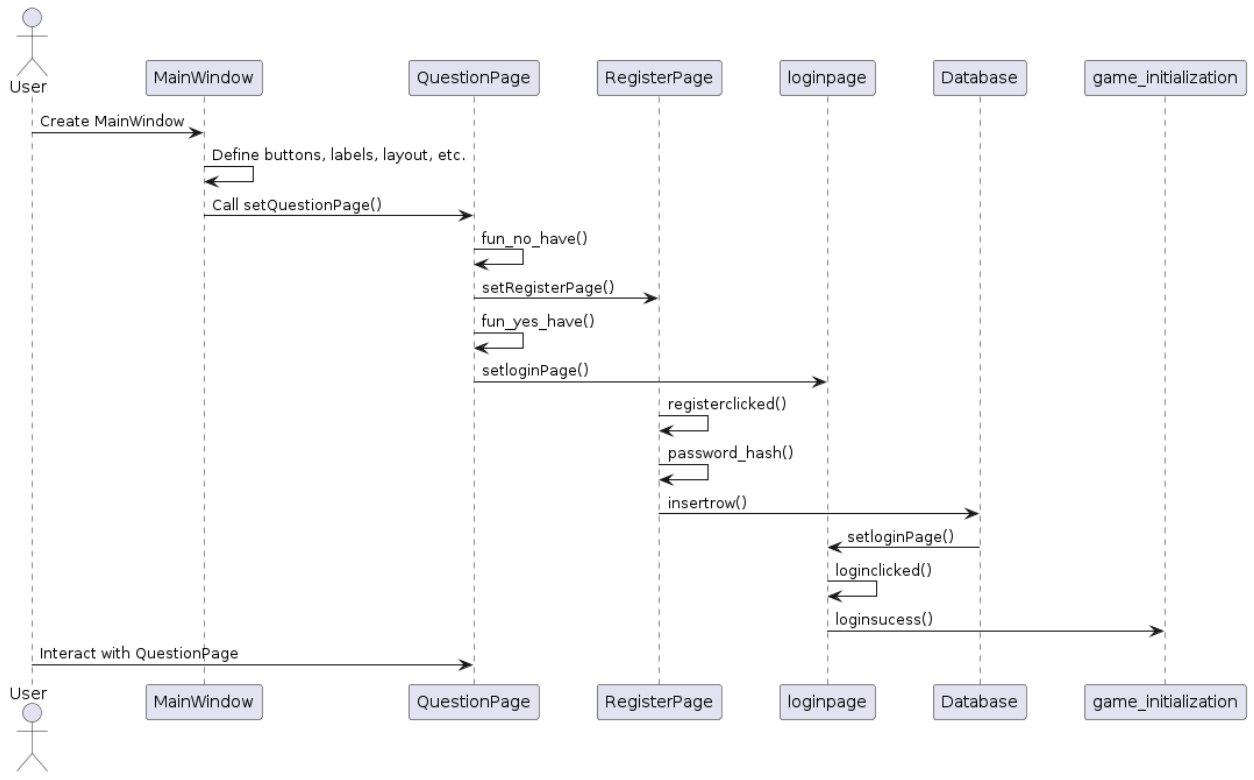
7. Assumptions and Dependencies

-Assumptions:

Users are familiar with basic tic-tac-toe rules.

-Dependencies:

Relies on stable internet connectivity for online play; utilizes standard GUI components for consistent user experience.


User
MainWindow
QuestionPage
RegisterPage
loginpage
Database
game_initialization
Create MainWindow
Define buttons, labels, layout, etc.
Call setQuestionPage()
fun_no_have()
setRegisterPage()
fun_yes_have()
setloginPage()
registerclicked()
password_hash()
insertrow()
setloginPage()
loginclicked()
loginsucess()
Interact with QuestionPage