## **Functional Requirements Document (FRD)**

### **1. Introduction**

This document outlines the functional requirements for a PHP-based Tic Tac Toe game. The game should support two players, display the game board, detect winners, handle invalid moves, and store game results.

### **2. Overall Description**

**2.1 Product Perspective** The Tic Tac Toe game is a console-based application for two players.

**2.2 Product Functions**

* Initialize a new game
* Allow players to take turns
* Display the game board after each move
* Determine the winner or a draw
* Handle invalid moves
* Store game results

**2.3 User Characteristics** Casual gamers familiar with Tic Tac Toe.

### **3. General Description**

**3.1 User Functions**

* Start a new game
* Make a move
* View game results

**3.2 Hardware Interfaces**

* Standard input/output

**3.3 Software Interfaces**

* PHP programming language

**3.4 Communications Interfaces** N/A

**3.5 Controls, Inputs, Outputs**

* User inputs: Move coordinates (row, column)
* System outputs: Game board, win/loss/draw message, game results

### **4. Specific Requirements**

**4.1 Functional Requirements**

* The game board should be a 3x3 grid.
* Players alternate turns placing their mark (X or O) on an empty cell.
* The game ends when one player has three marks in a row, column, or diagonal, or when the board is full (draw).
* Invalid moves (occupied cell, out-of-bounds) should be handled with an error message.
* Game results should be stored, including player names, moves, and outcome.

**4.2 User Interface**

* Text-based interface displaying the game board.
* Clear prompts for player input.
* Informative messages for game status, winner, or draw.

**4.3 Performance**

* The game should respond promptly to user input.

**4.4 Design Constraints**

* PHP programming language
* Console-based application

### **5. Other Non-functional Requirements (NFRs)**

* **Usability:** The game should be easy to understand and play.
* **Reliability:** The game should function correctly without errors.
* **Maintainability:** The code should be well-structured and easy to modify.
* **Portability:** The game should be compatible with different PHP environments.

## **Non-Functional Requirements (NFR) Document**

### **1. Introduction**

This document outlines the non-functional requirements (NFRs) for the PHP-based Tic Tac Toe game.

### **2. Usability**

* The game interface should be intuitive and easy to navigate.
* Clear and concise instructions should be provided.
* Error messages should be informative and helpful.

### **3. Reliability**

* The game should function correctly without crashes or unexpected behavior.
* Data integrity should be maintained.

### **4. Maintainability**

* The code should be well-structured, commented, and easy to understand.
* The code should be modularized for easy modification.
* Appropriate coding standards should be followed.

### **5. Portability**

* The game should be compatible with different PHP versions and environments.
* The code should avoid platform-specific dependencies.

## **Technical Design Document (TDD)**

### **1. Introduction**

This document outlines the technical design for the PHP-based Tic Tac Toe game.

### **2. Overall Design**

* **Game Board:** A 3x3 array to represent the game board.
* **Players:** Two player objects with names and marks (X or O).
* **Game Logic:** Functions to handle player turns, check for win/draw conditions, and validate moves.
* **Data Storage:** A file or database to store game results.

### **3. Data Design**

* **Game Result:** A data structure to store player names, moves, and outcome.

### **4. Architectural Design**

* **Modular structure:** Separate modules for game board, players, game logic, and data storage.
* **Object-oriented design:** Use classes for game objects and data structures.

### **5. Interface Design**

* **Text-based interface:** Use PHP's output functions to display the game board and messages.
* **User input:** Use PHP's input functions to get user moves.

### **6. Component Design**

* **Game Board:** A class representing the game board with methods to display, update, and check for win/draw conditions.
* **Player:** A class representing a player with properties for name and mark.
* **Game Logic:** Functions for handling player turns, checking for win/draw conditions, and validating moves.
* **Data Storage:** Functions for saving and retrieving game results.

### **7. Algorithms and Data Structures**

* **Array:** To represent the game board.
* **String:** To store player names and marks.
* **File or database:** To store game results.

### **8. Error Handling**

* Check for invalid input (non-numeric, out-of-bounds) and display appropriate error messages.