Component	Type	Purpose
AppEntry	Structure	Starts the App, loads StartView
StartView	Structure (view)	Displays GameMode selection/player input
GameView	Structure (view)	Displays Tic Tac Toe board/handles player turns
SquareView	Structure (view)	Represents a single tile on the board
GameService	Class	Manages game logic (turns, wins, resetting)
GameSquare	Structure	Represents a single board title
Player	Structure	Stores player info
GameType	Enum	Defines the different game types
Moves	Enum	Stores all possible moves and win conditions

How it works:

The Tic-Tac-Toe app is structured with a combination of SwiftUI views, data models, and game logic classes that work together to manage the user experience and gameplay flow. The app begins execution in AppEntry.swift, which initializes an instance of GameService as a @StateObject and loads StartView.swift as the first screen. StartView.swift provides the user interface for selecting a game mode, inputting player names, and starting the game. When the "Start Game" button is pressed, game.setupGame() is called, which initializes the game settings and transitions the user to GameView.swift, where the game board is displayed.

Within GameView.swift, the Tic-Tac-Toe board is visually structured using multiple HStack groups, each containing SquareView components that represent individual tiles. When a player taps a square, SquareView.swift calls game.makeMove(at: index), which is handled by GameService.swift. This game controller class tracks the game state, manages turns between players, and determines win conditions. The game logic is further supported by

GameSquare.swift, which defines the structure of each board tile, and Player.swift, which stores player information such as their name, game piece (X or O), and the moves they have made.

To determine the outcome of the game, GameService.swift calls checkIfWinner(), which verifies whether a player has matched one of the predefined winning patterns stored in Moves.swift. If a win is detected, the gameOver flag is set to true, and the game presents the final result, displaying either the winning player's name or a "Nobody wins" message in case of a tie. The user can then restart the game by tapping the "New Game" button, which triggers game.reset(), clearing the board and resetting all player moves.

Additionally, GameType.swift provides an enumeration defining different game modes, such as single (two players sharing a device), bot (playing against an AI opponent), and peer (playing remotely with another user). These game types influence how GameService.swift processes moves and turn-taking behavior. Throughout the entire app, the @EnvironmentObject var game: GameService property allows different views, such as StartView.swift, GameView.swift, and SquareView.swift, to access and modify the shared game state.

The entire app follows a structured and modular approach, ensuring that user interactions, board updates, and game logic execution are handled efficiently across different components. The interaction between views, data models, and logic classes ensures that the game runs smoothly, whether a user plays a complete round, restarts the game, or exits to the main menu.