Software Architecture

1. System Overview

This project implements a browser-based Tic-Tac-Toe single-page application (SPA) built with native HTML, CSS, and JavaScript.
The application runs entirely on the client side and does not require a backend server or database.
Core features include: placing moves, alternating turns, detecting wins or draws, highlighting winning cells, and restarting a match.
Basic accessibility support (ARIA labels, focus behavior, keyboard input) is also included.

Scope & Constraints:

- Technology stack: Pure HTML + CSS + JavaScript
- Runtime: Modern browsers (Chrome / Edge / Firefox)
- Not included: user accounts, networking, AI logic, persistent server-side storage

---

2. Architectural Goals & Quality Attributes

- **Simplicity & clarity:** All logic resides in `app.js`, making it easy to understand and maintain.
- **Maintainability:** Game rules and UI manipulation are separated into functions.
- **Usability & accessibility:** Screen-reader friendly status updates and keyboard controls.
- **Performance:** Immediate local computations; no network delays.

---

3. Architectural Style

The project adopts a **lightweight, front-end layered architecture** based on the separation of structure, presentation, and behavior:

- **Structure layer (HTML):** Provides the board layout and interactive elements.
- **Presentation layer (CSS):** Controls theming, layout, colors, and win highlights.
- **Behavior layer (JavaScript):** Contains the game state, rules, event handlers, and DOM updates.

Additionally, the application follows an **event-driven** interaction model: user actions (clicks and key presses) trigger updates to game logic and UI state.

---

4. Components & Responsibilities

1) User Interface (HTML + CSS)

- **`#board`**: A 3×3 grid of `<button class="cell">` elements, each mapped to an index (`data-i`).
- **`#status`**: A live region (`aria-live="polite"`) for announcing the current player, wins, or draws.
- **`#reset`**: A restart button that resets the board.
- **CSS (`style.css`)**: Provides
    - dark-theme design,
    - grid layout,
    - cell hover states,
    - win highlighting via `.cell.win`.

2) Game Logic (`js/app.js`)

State variables:

- `cells`: an array of 9 strings (`""`, `"X"`, or `"O"`).
- `current`: whose turn it is (`"X"` or `"O"`).
- `active`: whether moves are still allowed.

Constants:

- `LINES`: the eight winning combinations (rows, columns, diagonals).

Event bindings:

- Click handlers for all `.cell` buttons
- Keyboard support (Enter / Space triggers a move)
- Click handler for the restart button

Core functions:

- `handleMove(i, btn)`: validates the move, updates the board, checks win/draw conditions, switches turns.
- `getWinLine()`: scans all 8 winning combinations and returns the winning line if found.
- `showWin(line)`: highlights winning cells and disables all input.
- `reset()`: resets global state and restores the board to its initial state.

---

5. Data Model

Board Representation

The board is stored as a **one-dimensional array** of length 9:

- `let cells = ["", "", "", "", "", "", "", "", ""];`

Positions map directly to the 3×3 layout (0–8).
Values represent the state of each cell: empty, `"X"`, or `"O"`.

Derived State

- **Current player:** `"X"` always starts.
- **Game activity:** turns allowed until a win or draw occurs.

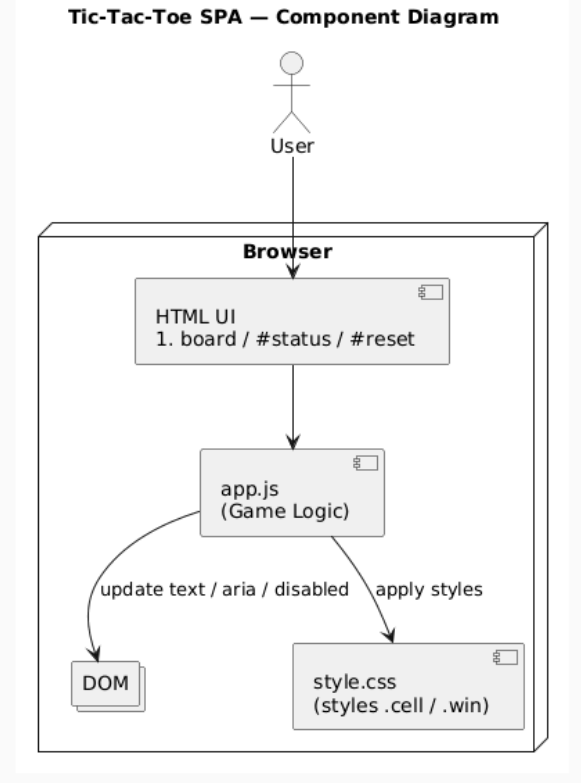
Win/Draw Logic

- A win occurs when three identical non-empty cells form one of the `LINES`.
- A draw occurs when all cells are filled without a winning line.

---

6. Interaction & Data Flow

Architecture Diagram (PlantUML)

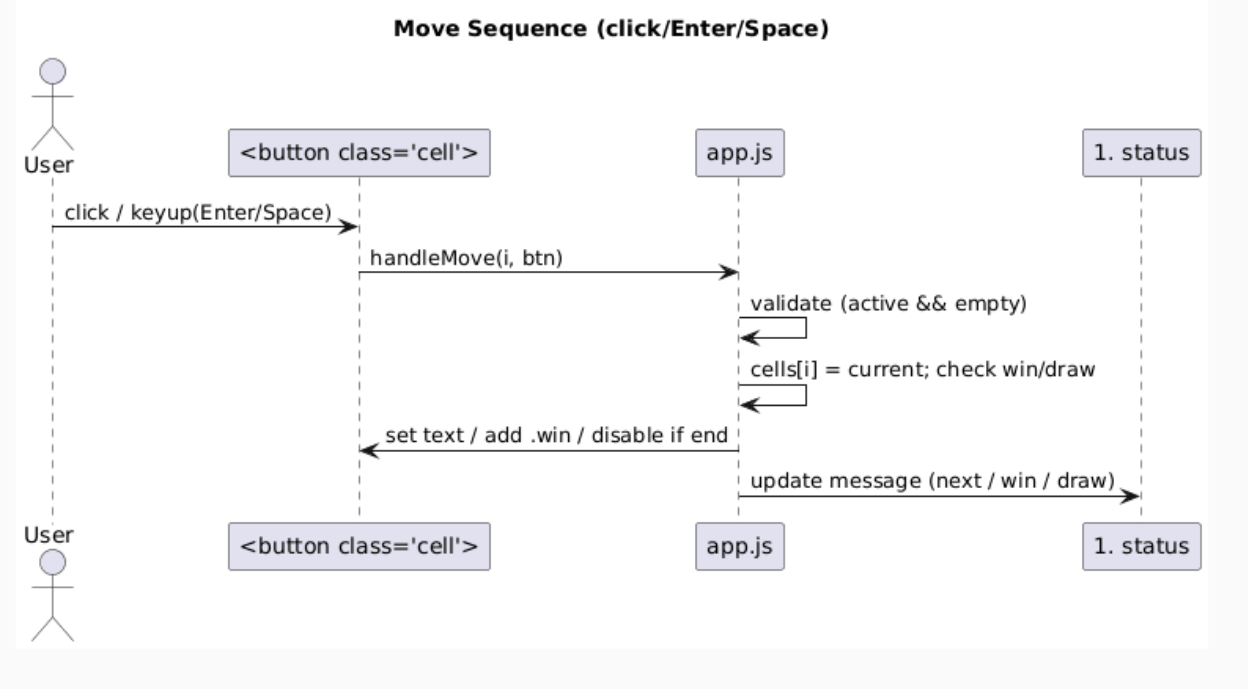


```
@startuml
title Tic-Tac-Toe SPA — Component Diagram
actor User

node "Browser" {
  component "HTML UI\n#board / #status / #reset" as UI
  component "app.js\n(Game Logic)" as JS
  collections "DOM" as DOM
  component "style.css\n(styles .cell / .win)" as CSS
}

User --> UI
UI --> JS
JS --> DOM : update text / aria / disabled
JS --> CSS : apply styles
@enduml
```

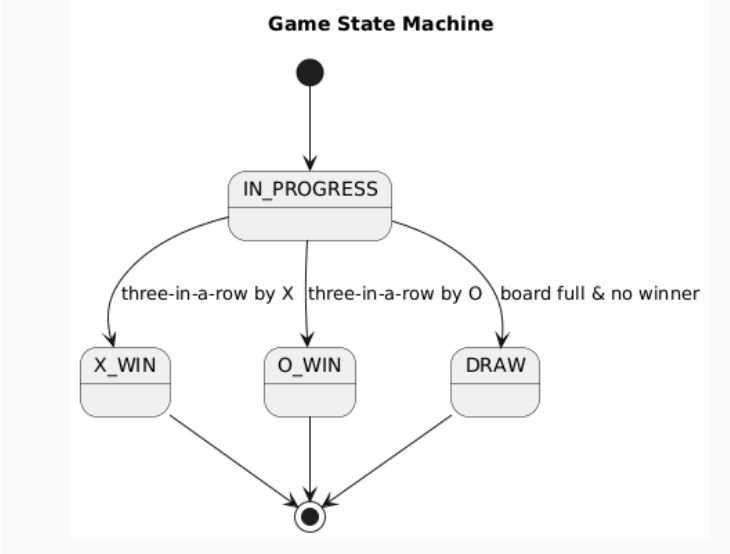
Move Sequence Diagram




```
@startuml
title Move Sequence (click/Enter/Space)
actor User
participant Cell as "<button class='cell'>"
participant App as "app.js"
participant Status as "#status"

User -> Cell: click / keyup(Enter/Space)
Cell -> App: handleMove(i, btn)
App -> App: validate (active && empty)
App -> App: cells[i] = current; check win/draw
App -> Cell: set text / add .win / disable if end
App -> Status: update message (next / win / draw)
@enduml
```

State Machine




```
@startuml

title Game State Machine

[*] --> IN_PROGRESS

IN_PROGRESS --> X_WIN : three-in-a-row by X

IN_PROGRESS --> O_WIN : three-in-a-row by O

IN_PROGRESS --> DRAW  : board full & no winner

X_WIN --> [*]

O_WIN --> [*]

DRAW  --> [*]

@enduml
```

---

7. Error Handling & Accessibility

Error Handling

- Invalid moves (cell already filled or game finished) are ignored.
- Once the game ends, all cells are disabled to prevent further input.

Accessibility

- Buttons are keyboard-accessible by default.
- `aria-label` dynamically updates to announce `"cell 5 X"` when a move is made.
- `aria-live="polite"` allows screen readers to read status changes automatically.

---

8. Deployment & Runtime

- The site runs as a **static website**.
- No backend is required; deployment can be done via:
    - GitHub Pages
    - Netlify
    - Any static hosting service
- Opening `index.html` in a browser is sufficient for offline use.

---

9. Testing Strategy

Manual Testing

- Horizontal/vertical/diagonal wins
- Draw scenario
- Disabled board after win/draw
- Reset functionality
- Keyboard controls

Optional Automated Tests

A future improvement would be to extract `getWinLine()` and create unit tests using Jest or similar frameworks.

---

10. Rationale & Future Enhancements

A pure front-end approach was selected to minimize complexity and focus on core game logic. This design is simple, maintainable, and easy to extend. Potential future enhancements include:

- Adding AI (Minimax)
- Adding persistent storage using `localStorage`
- Adding animations or sound effects
- Extracting logic into modules for unit testing
- Creating a mobile-friendly layout