TIC TAC TOE APPLICATION

1. What Are We Building?

We are building a **Tic Tac Toe Web Application**.

It is a simple 2-player game where:

- ❌ Player X clicks a box
- ⭕ Player O clicks next
- 🏆 First player to make 3 in a row wins

The game runs directly inside the browser.

2. Technologies Used

Technology	What it is used for
Python	Main programming language
Flask	Helps create the website (backend)
HTML	Creates structure of webpage
CSS	Makes the page look beautiful
JavaScript	Controls game logic and player turns

3. Before Starting (Setup)

Step 3.1: Install Python

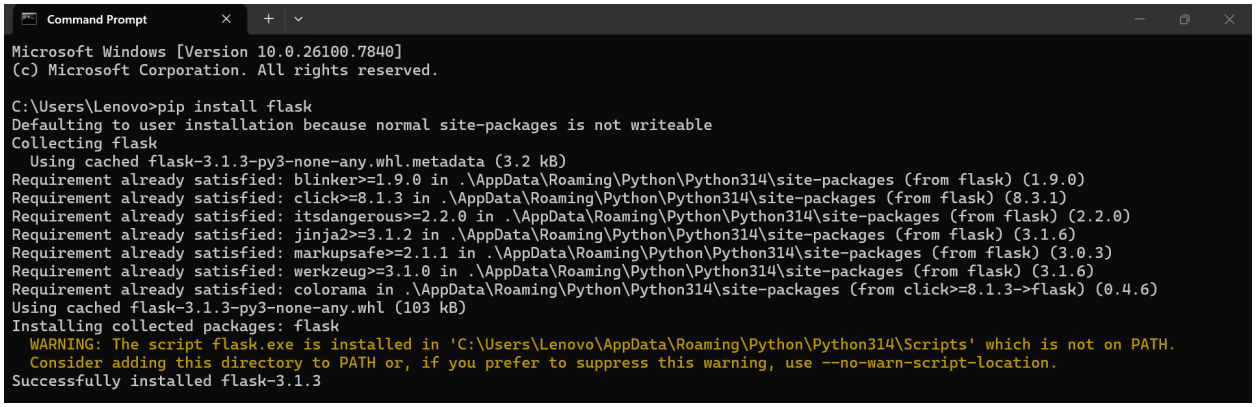
Download from: https://www.python.org

While installing → check “Add Python to PATH”

Step 3.2: Install Flask

Open terminal / command prompt and type:

```
pip install flask
```



This installs Flask, which is required to run the app.

4. Create Project Structure

Create a folder named `TICTACTOE`

Inside it, create:

5. Backend Development

This file is the **backend** of the application.It connects Flask with the webpage.

Step 5.1: Import Required Modules

```
from flask import Flask, render_template
```

These help:

- Create flask app
- Display HTML pages

Step 5.2: Create Flask App

```
app = Flask(__name__)
```

Initializes the Flask application.

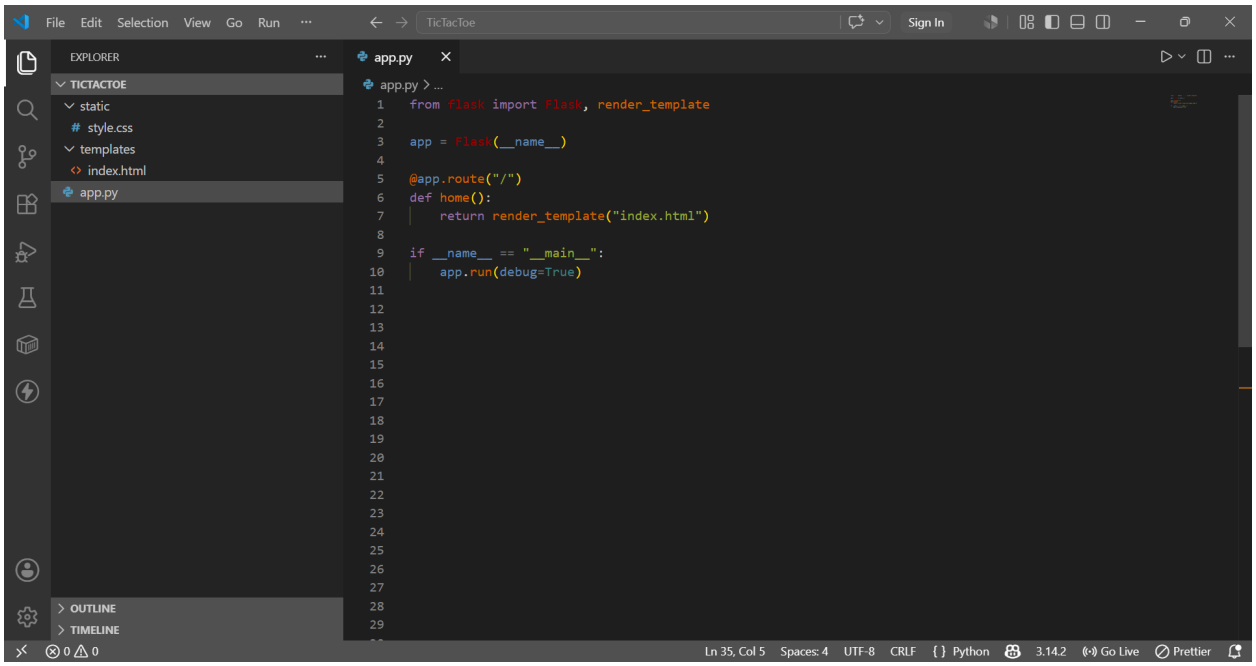
Step 5.3: Create Home Route


```
@app.route("/")
def home():
    return render_template("index.html")
```

- When user opens website
- Flask loads index.html
- The game page appears

Step 5.4: Run Application

```
if __name__ == "__main__":
    app.run(debug=True)
```



6. Frontend Development

index.html (Main Game Page)

This file creates:

- Game board
- Player turn text
- Game logic using JavaScript

Step 6.1: Add Page Title

```
<title>Tic Tac Toe</title>
```

Sets browser tab title.

Step 6.2: Connect CSS File

```
<link
  rel="stylesheet"
  href="{{ url_for('static', filename='style.css') }}"
/>
```

Connects external CSS styling.

Step 6.3: Create Game Board

```
<div class="board">
```

Creates a 3×3 grid.

Each cell is:

```
<div class="cell" onclick="play(this, 0)"></div>'
```

Explanation:

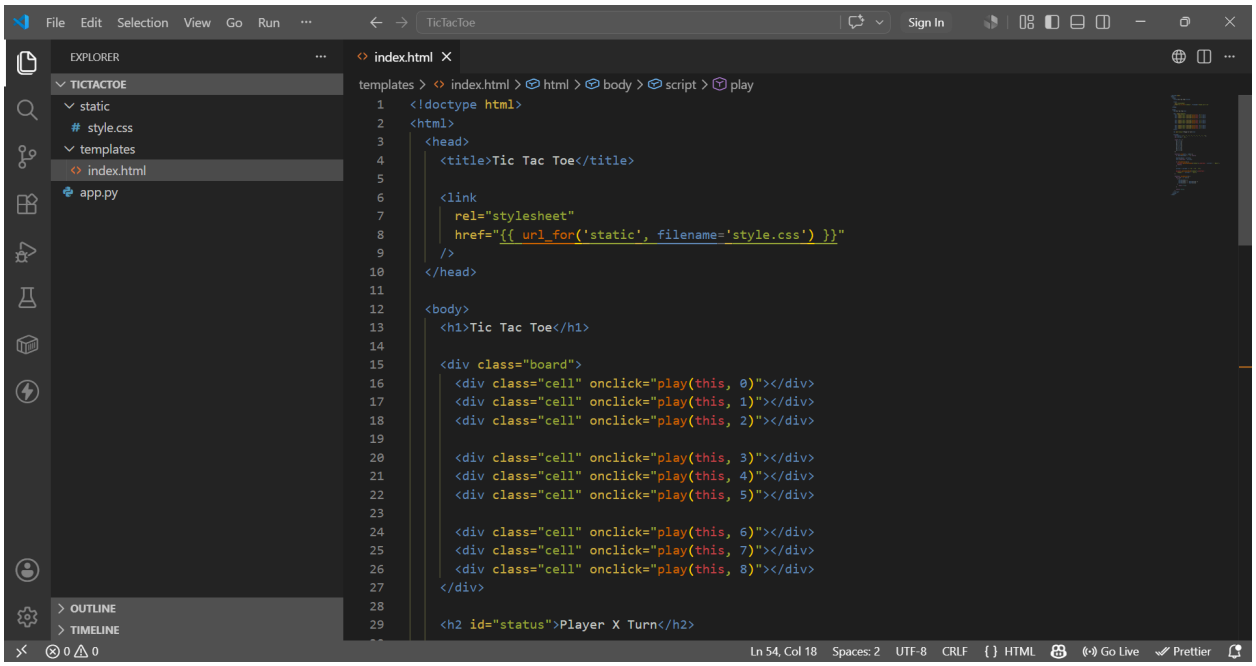
- `cell` = one box
- `onclick` = runs function when clicked
- `play()` handles moves

Step 6.4: Show Player Turn

```
<h2 id="status">Player X Turn</h2>
```


Displays:

- Current player
- Winner message



7. JavaScript Game Logic

JavaScript controls the entire game.

Step 7.1: Create Empty Board

```
let board = ["", "", "", "", "", "", "", "", ""];
```

Stores all 9 positions.

Step 7.2: Set Current Player

```
let current = "X";
```

The game starts with Player X.

Step 7.3: Winning Conditions

```
const win = [
  [0,1,2],
  [3,4,5],
  [6,7,8]
];
```

These combinations decide the winner.

Examples:

- Top row
- Middle row
- Bottom row
- Columns
- Diagonals

Step 7.4: Play Function

```
function play(cell, index)
```


Runs whenever the player clicks a box.

What happens inside?

- Prevents overwriting:

  ```
  if (board[index] != "") return;
  ```

- Stores move:

  ```
  board[index] = current;
  ```

- Displays X or O:

  ```
  cell.innerText = current;
  ```

- Checks winner:

  ```
  if (checkWinner())
  ```

- Changes player turn:

```
current = current === "X" ? "O" : "X";
```

Step 7.5: Winner Checking Function

```
function checkWinner()
```

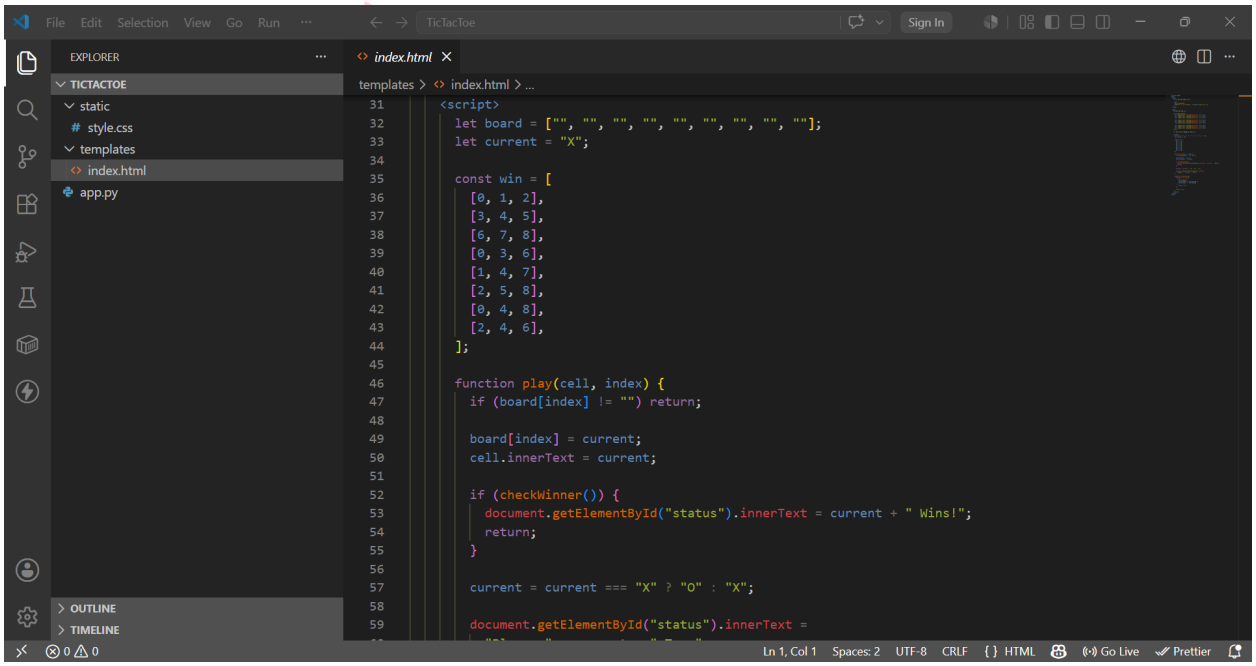
Loops through all winning combinations.

If 3 same symbols match:

```
return true;
```

Otherwise:

```
return false;
```




```
<script>
  let board = ["", "", "", "", "", "", "", "", ""];
  let current = "X";

  const win = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];

  function play(cell, index) {
    if (board[index] != "") return;

    board[index] = current;
    cell.innerText = current;

    if (checkWinner()) {
      document.getElementById("status").innerText = current + " Wins!";
      return;
    }

    current = current === "X" ? "O" : "X";

    document.getElementById("status").innerText =
```

8. Styling

File: `style.css`

This file makes the game attractive.

Step 8.1: Style Page

```
body {
  text-align: center;
  font-family: Arial;
  background: #f2f2f2;
}
```

Centers content and adds background color.

Step 8.2: Style Game Board

```
.board {
  width: 300px;
  display: grid;
  grid-template-columns: repeat(3, 100px);
}
```

Creates 3×3 layout using CSS Grid.

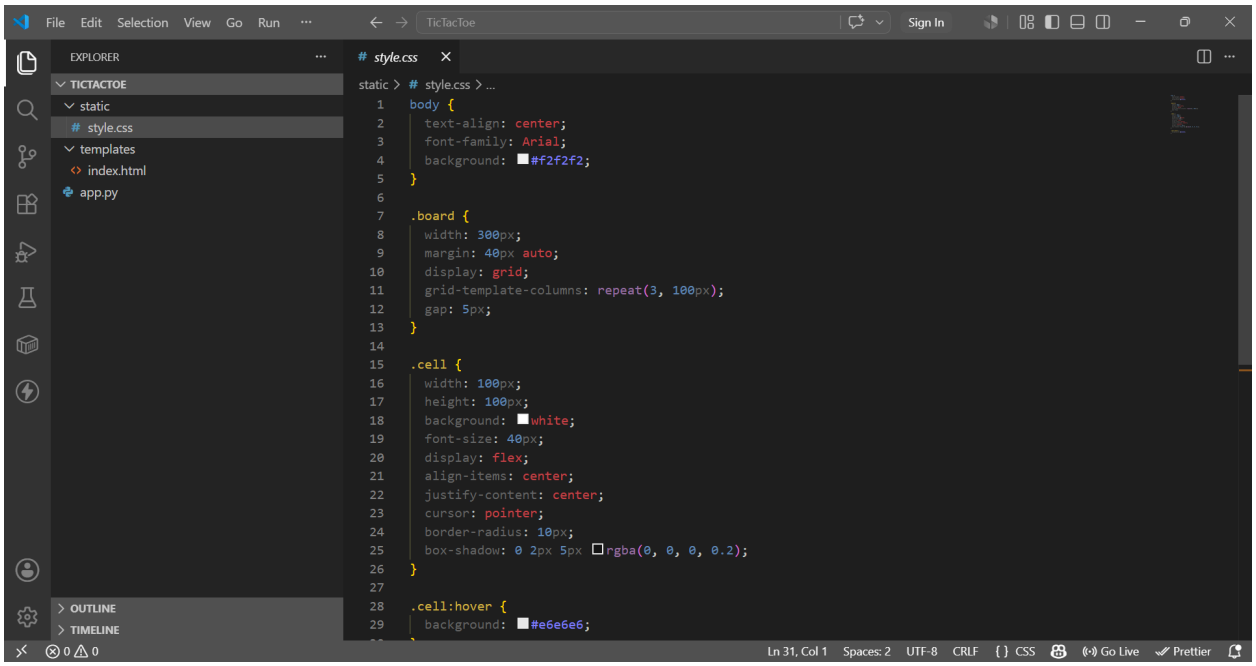
Step 8.3: Style Cells


```
.cell {
  width: 100px;
  height: 100px;
}
```

Creates square boxes.

Other styling adds:

- Rounded corners
- Shadow effects
- Hover effect
- Large text size



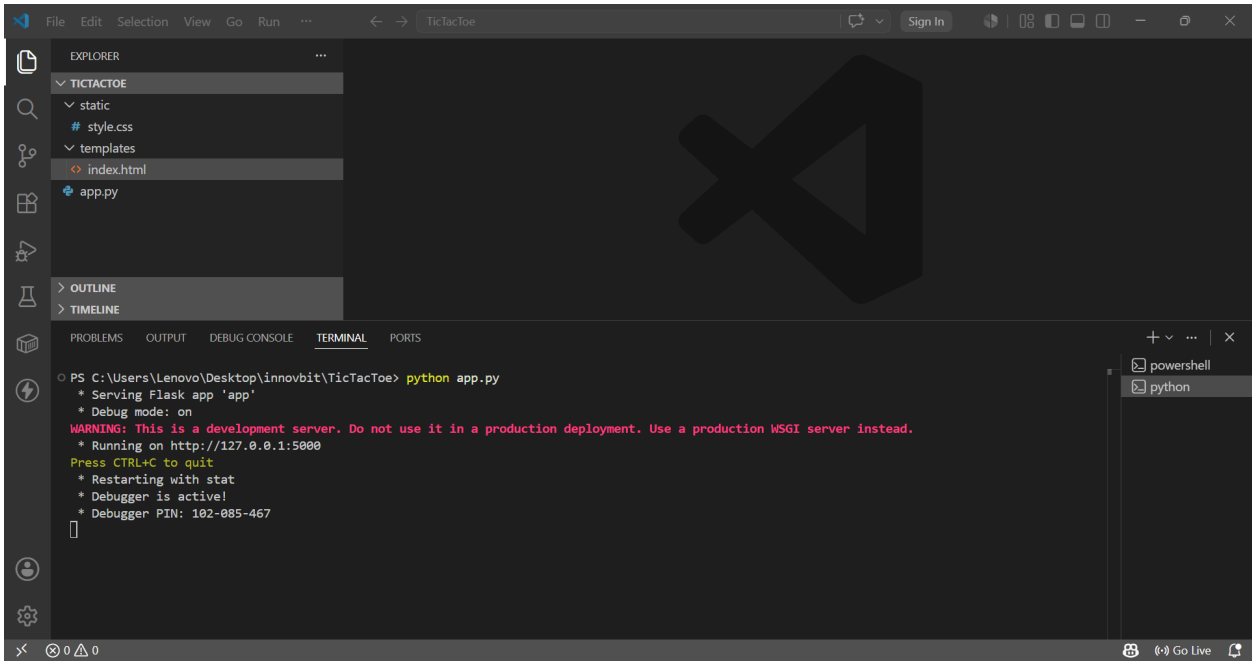
9. Run the App

Open terminal inside project folder:

```
python app.py
```


Then open browser:

```
http://127.0.0.1:5000
```



10. How Everything Works Together (Flow)

- User opens website
- Flask loads `index.html`
- User clicks a box
- JavaScript stores move
- Board updates instantly
- Winner is checked
- Turn changes to next player
- Game continues until someone wins

11. Common Mistakes (Very Important)

- Forgetting to install Flask
- Wrong folder names (`templates, static`)
- Forgetting to connect CSS file
- Using wrong cell indexes
- Not checking if cell is already filled

- Forgetting `debug=True`

12. Tips & Tricks

12.1. Add Restart Button

You can add:

```
<button onclick="location.reload()">Restart</button>
```

Reloads games.

12.2. Add Draw Detection

Check if all boxes are filled without a winner.

12.3. Improve UI

You can:

- Add animations
- Add sound effects
- Use Bootstrap
- Add dark mode

12.4. Add Single Player Mode

Use AI logic so users can play against computers.

12. Final Summary

This project teaches:

- Flask basics
- HTML structure
- CSS Grid
- JavaScript DOM manipulation
- Game logic creation

- Event handling
- Win condition checking

It is a great beginner project for learning how frontend and backend work together in a web application.