**Lesson 05: Create a Simple Tic Tac Toe Game using GitHub Copilot Generative AI**

**Overview:**

You are working as a developer at Tech Innovators Inc. and are tasked with creating a basic React application for a Tic Tac Toe game. To assist with the development, you are utilizing GitHub Copilot as your coding assistant. The project is broken down into several key phases: Project Setup, UI Creation, Game Logic, Bug Fixes, and UI Enhancement.

Throughout each phase, GitHub Copilot provides valuable code suggestions, streamlining the development process and significantly speeding up workflows. By leveraging AI tools like GitHub Copilot, Tech Innovators Inc. is able to efficiently complete the Tic Tac Toe game, demonstrating the power of AI in software development.

**Instructions:**

1. Initiate the project with GitHub Copilot to generate a React sample project
2. Collaborate with Copilot to create a basic Tic Tac Toe game, including UI design and game logic implementation
3. Update the CSS by generating a new script through ChatGPT to fix the header and footer to the top and bottom of the screen, respectively

**Tasks:**

1. Initialize the GitHub Copilot project to create a React sample project:
   1. Set up the workspace and open the generated React project
   2. Ensure the project environment is configured properly
2. Create a basic Tic Tac Toe Game using GitHub Copilot:
   1. Use prompts to create the basic user interface
   2. Use prompts to develop the basic game logic and determine when a   
      player wins
   3. Use prompts to enhance the board’s user interface aesthetics

**Tools required:**

1. GitHub Copilot
2. Node JS
3. Visual Studio Code

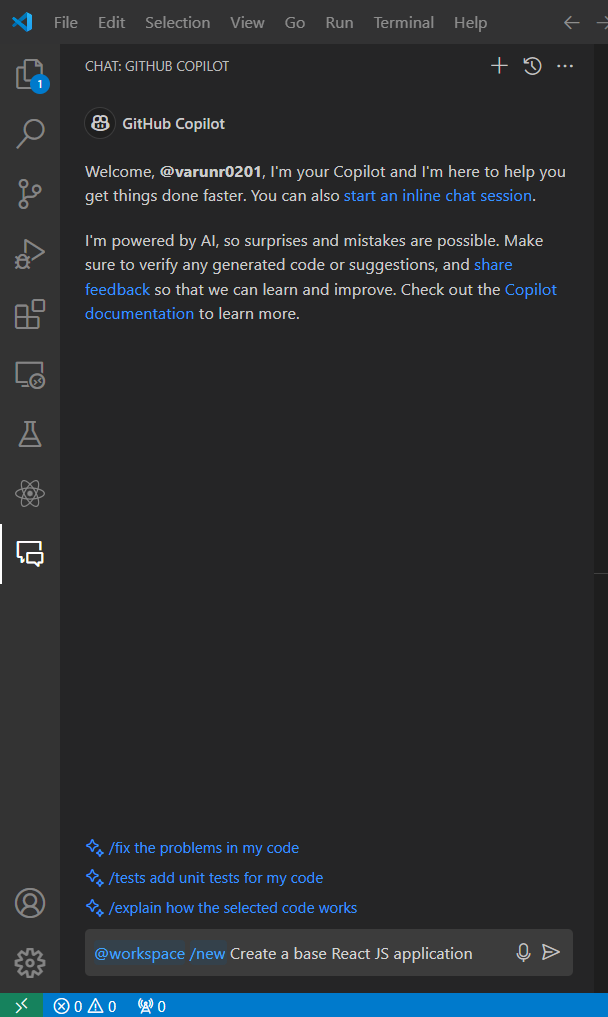
**Guided Practice Solution**

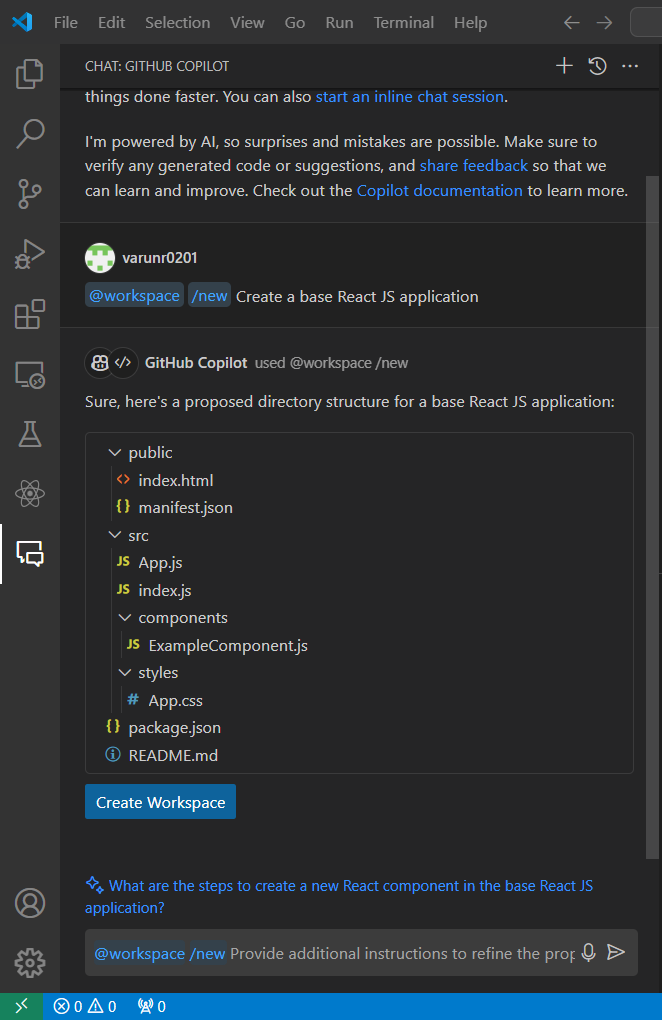
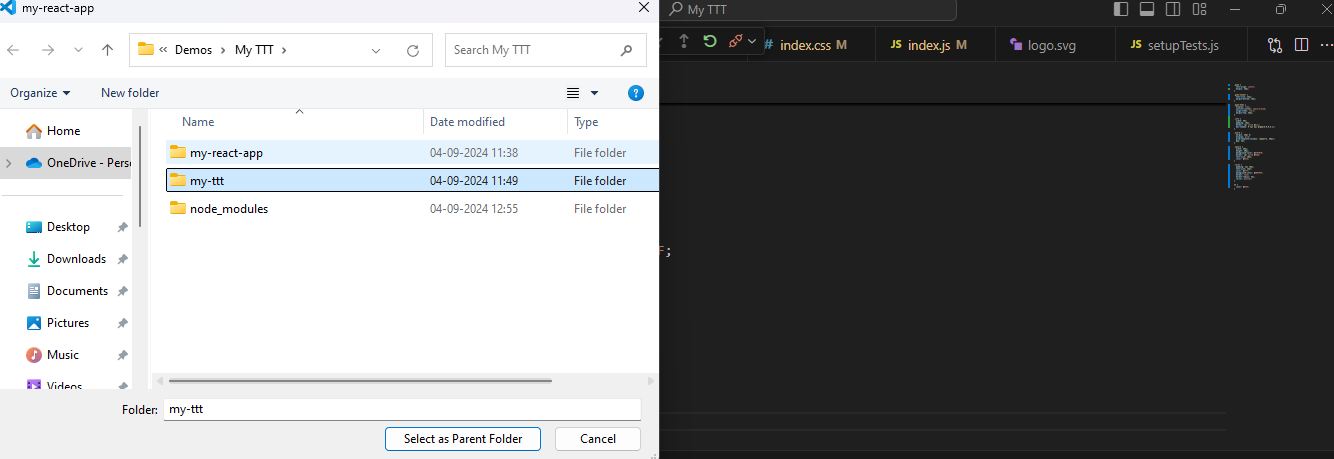
**Disclaimer:** Please note that all the GenAI tools used in this exercise can produce varied outputs even when presented with similar prompts. Thus, you may get different output for the same prompt.

**Task 1: Initialize the GitHub Copilot project to create a React sample project**

**Step 1: Set up the workspace and open the generated React project**

* 1. Navigate to Visual Studio Code and use the following prompt in GitHub Copilot to generate a basic React JS application:  
     **@workspace /new** **create a base React JS application**

****

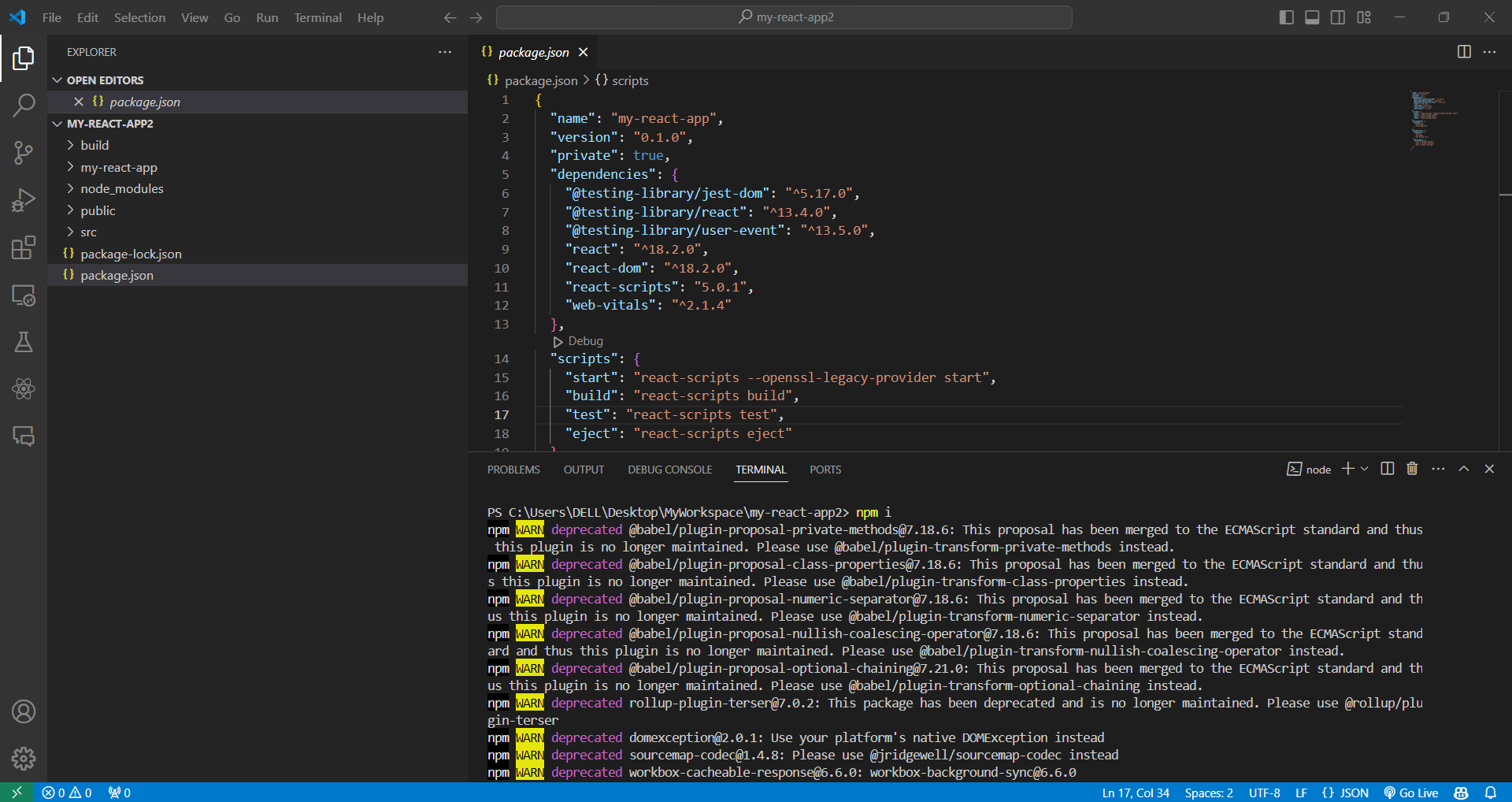
* 1. Create the workspace by clicking on **Create Workspace   
       
     **
  2. Choose the folder for your parent folder and then click on **Select as Parent Folder  
       
     **

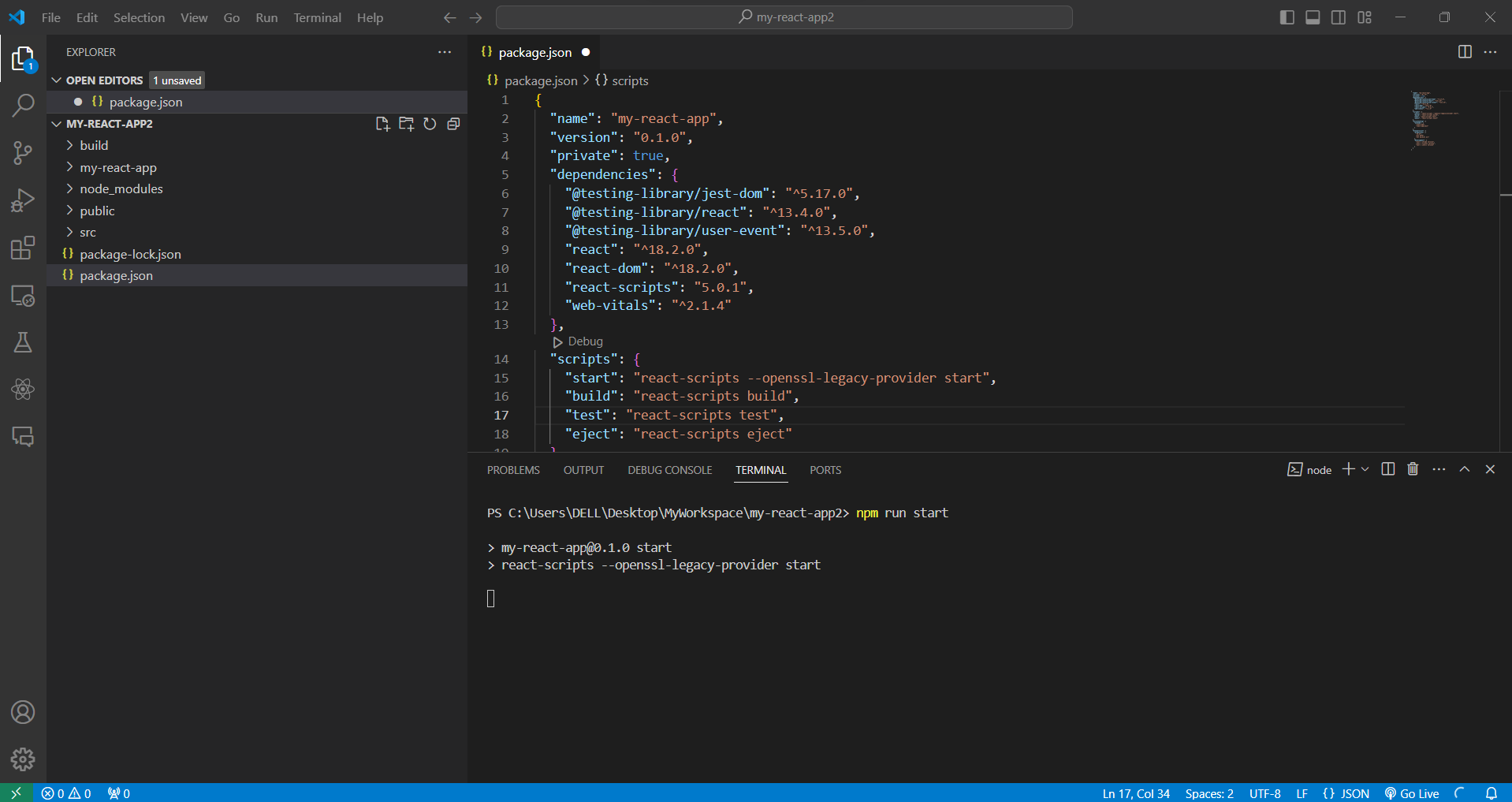
**Step 2: Ensure the project environment is configured properly**

* 1. Run the following command to initiate and create a React application:  
     **npx create-react-app my-ttt**   
       
     A screenshot of a computer

     Description automatically generated
  2. Run the following command to navigate inside the parent folder:  
     **cd parent\_folder\_name**  
       
     
  3. Run the following command to install all the dependencies listed in the project's package.json file:  
     **npm install**  
       
     A computer screen with white text

     Description automatically generated
  4. Run the following command to install packages:  
     **npm i**

****

* 1. Run the project using the following command:   
     **npm run start  
       
     **
  2. Once the project runs successfully and displays the success message, click on the Local link, as shown in the screenshot below:  
       
     **A screenshot of a computer

     Description automatically generated**  
     The output is as follows:  
       
     A blue and white symbol with white text

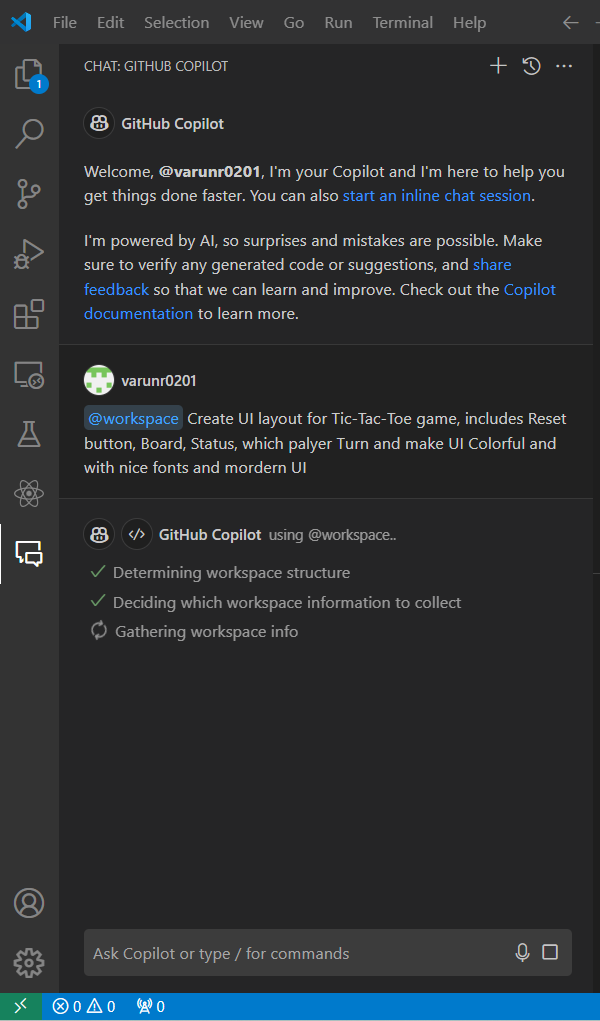
     Description automatically generated

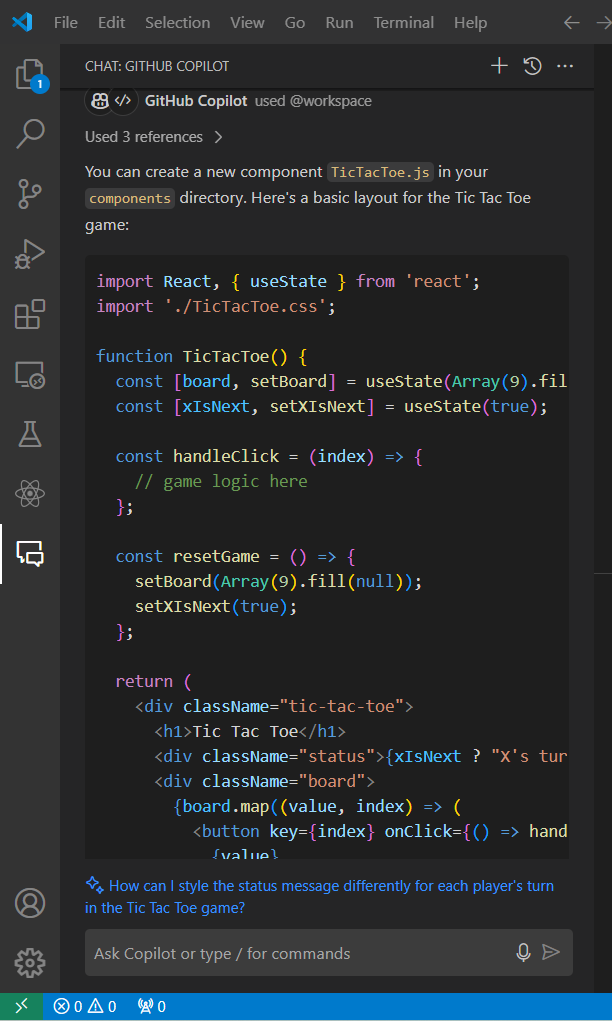
**Task 2: Create Basic Tic-Tac-Toe Game Using GitHub Copilot**

**Step 1: Use prompts to create the basic user interface**

* 1. Use the following prompt to generate a basic UI for the Tic-Tac-Toe game:

**Create UI layout for Tic-Tac-Toe game, includes Reset button, Board, Status, which player Turn and make UI Colorful and with nice fonts and modern UI**

****

* 1. Place the generated script inside the src folder   
       
     ****

* 1. Run the following command to execute scripts:

**npm run**  
****

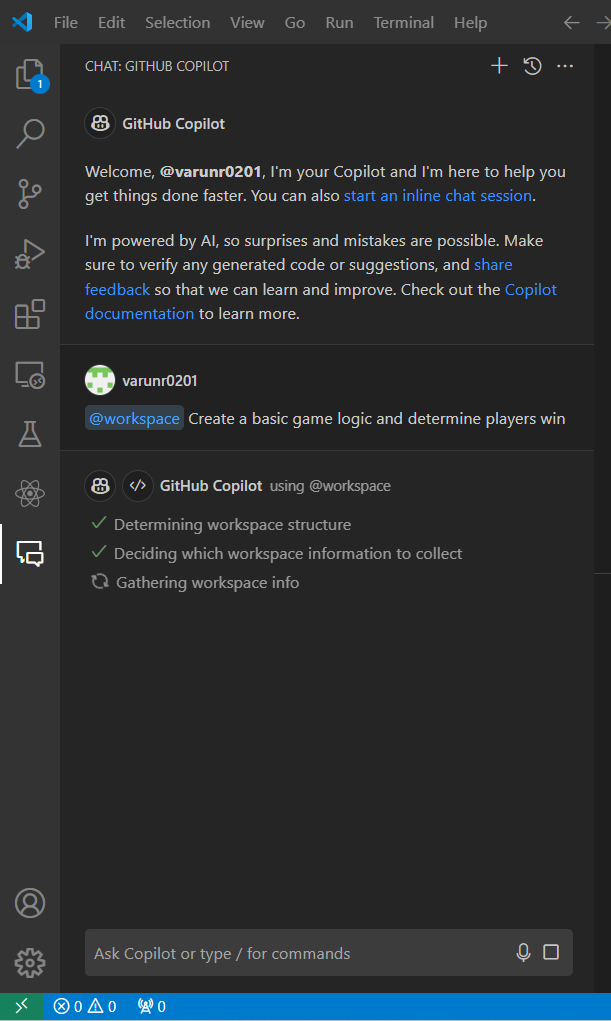
The output after executing the basic Tic-Tac-Toe game is as follows:

**A screenshot of a computer game

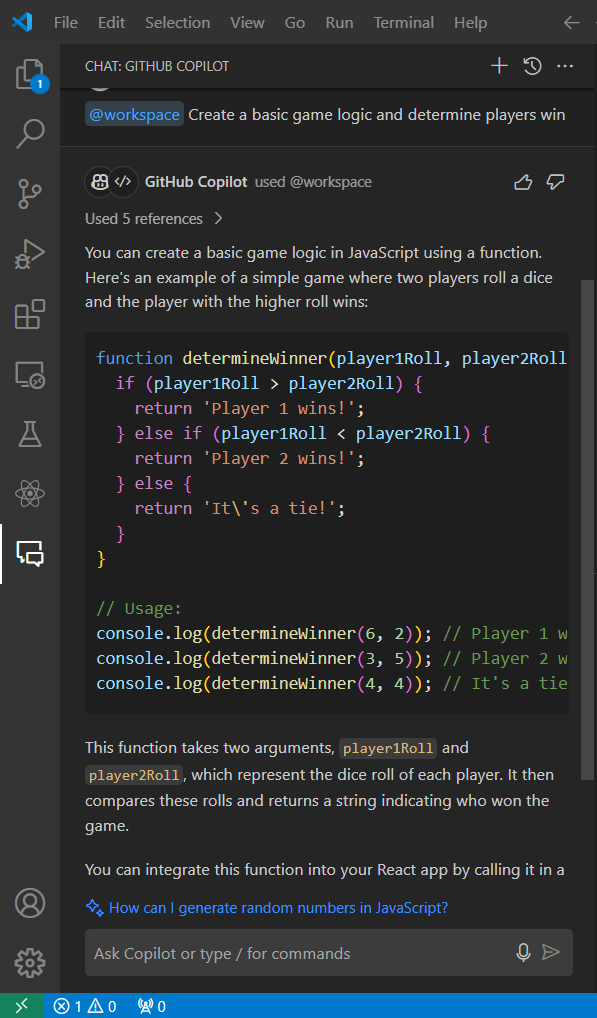
Description automatically generated**

**Step 2: Use prompts to develop the basic game logic and determine when a   
 player wins**

* 1. Use the following prompt to create a basic game logic and determine when a   
     player wins:  
     **@workspace Create a basic game logic and determine players win**

****

The response from GitHub Copilot is as follows:

****

* 1. Add the following prompt inside App.js to enhance the user experience:  
     **import React, { useState } from 'react';**

**import './App.css';**

**function App() {**

**const [board, setBoard] = useState(Array(9).fill(null));**

**const [xIsNext, setXIsNext] = useState(true);**

**const winner = calculateWinner(board);**

**const handleClick = (index) => {**

**if (board[index] || winner) {**

**return;**

**}**

**const newBoard = board.slice();**

**newBoard[index] = xIsNext ? 'X' : 'O';**

**setBoard(newBoard);**

**setXIsNext(!xIsNext);**

**};**

**const renderSquare = (index) => {**

**return (**

**<button className="square" onClick={() => handleClick(index)}>**

**{board[index]}**

**</button>**

**);**

**};**

**return (**

**<div className="game">**

**<div className="header">Tic-Tac-Toe</div>**

**<div className="board">**

**{Array(3).fill(null).map((\_, row) => (**

**<div key={row} className="board-row">**

**{Array(3).fill(null).map((\_, col) => renderSquare(row \* 3 + col))}**

**</div>**

**))}**

**</div>**

**<button onClick={() => setBoard(Array(9).fill(null))} className="reset">**

**Reset**

**</button>**

**<div className="info">**

**{winner ? `Winner: ${winner}` : `Next player: ${xIsNext ? 'X' : 'O'}`}**

**</div>**

**</div>**

**);**

**}**

**function calculateWinner(squares) {**

**const lines = [**

**[0, 1, 2], [3, 4, 5], [6, 7, 8], // Rows**

**[0, 3, 6], [1, 4, 7], [2, 5, 8], // Columns**

**[0, 4, 8], [2, 4, 6] // Diagonals**

**];**

**for (let i = 0; i < lines.length; i++) {**

**const [a, b, c] = lines[i];**

**if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {**

**return squares[a];**

**}**

**}**

**return null;**

**}**

**export default App;  
  
A screenshot of a computer screen

Description automatically generated**

* 1. Add the following code inside App.css to enhance the visual appearance:  
     **.game {**

**font-family: Arial, sans-serif;**

**text-align: center;**

**margin-top: 20px;**

**}**

**.header {**

**font-size: 24px;**

**color: black;**

**margin-bottom: 20px;**

**}**

**.board {**

**display: grid;**

**grid-template-columns: repeat(3, 60px);**

**gap: 5px;**

**justify-content: center;**

**margin: auto;**

**}**

**.square {**

**width: 60px;**

**height: 60px;**

**background-color: green;**

**border: 1px solid darkgreen;**

**font-size: 20px;**

**color: white;**

**font-weight: bold;**

**cursor: pointer;**

**}**

**.reset {**

**background-color: red;**

**color: white;**

**border: none;**

**padding: 10px 20px;**

**margin-top: 20px;**

**cursor: pointer;**

**font-size: 16px;**

**}**

**.info {**

**font-size: 16px;**

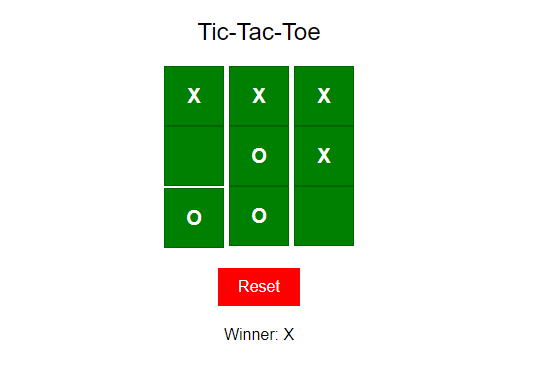
**color: black;**

**margin-top: 20px;**

**}**  
**A screen shot of a computer

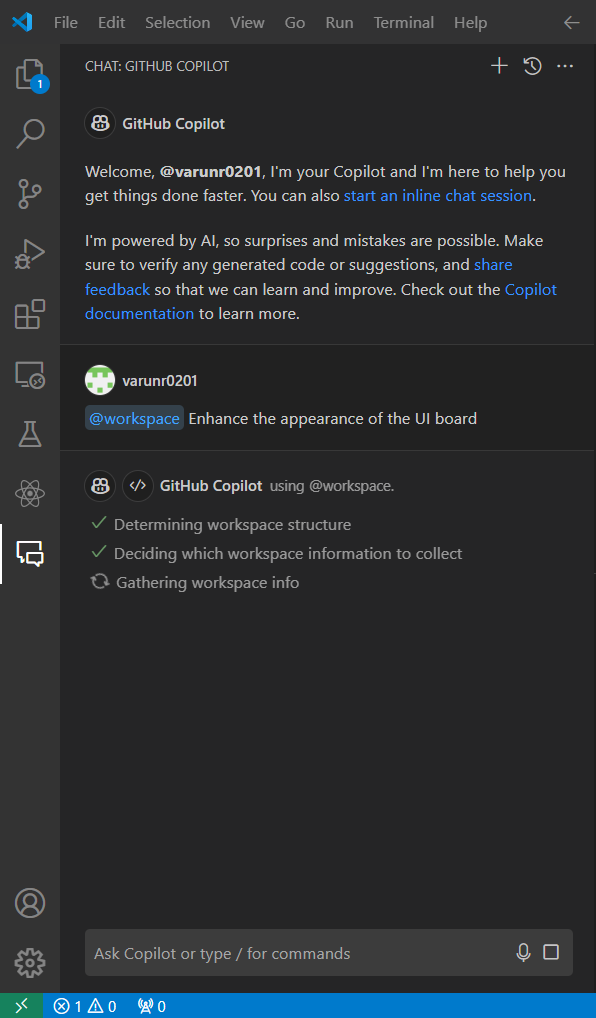
Description automatically generated**

The output after implementing the game logic is as follows:

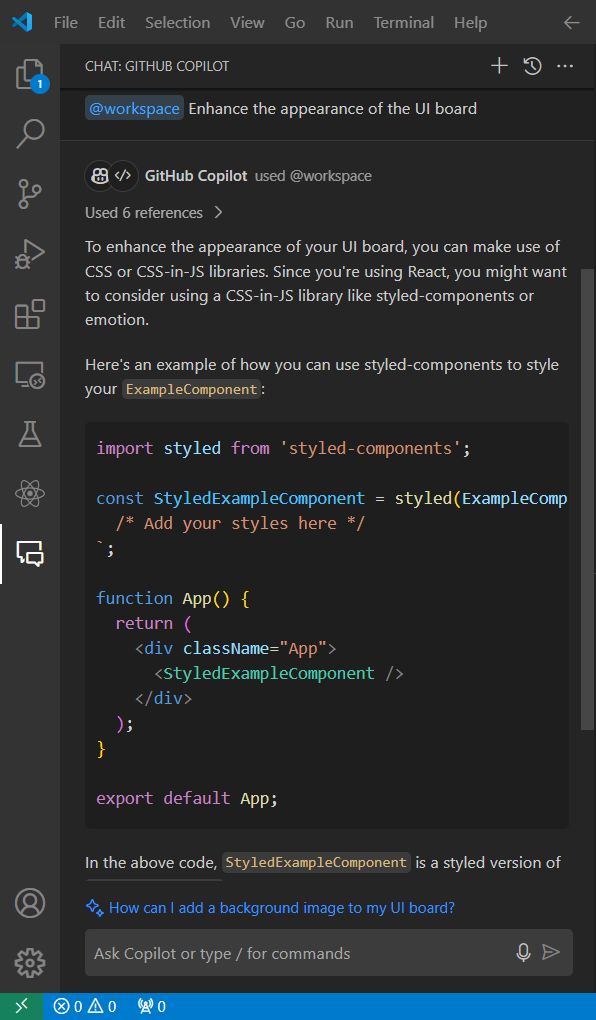
****

**Step 3: Use prompts to enhance the board’s user interface aesthetics**

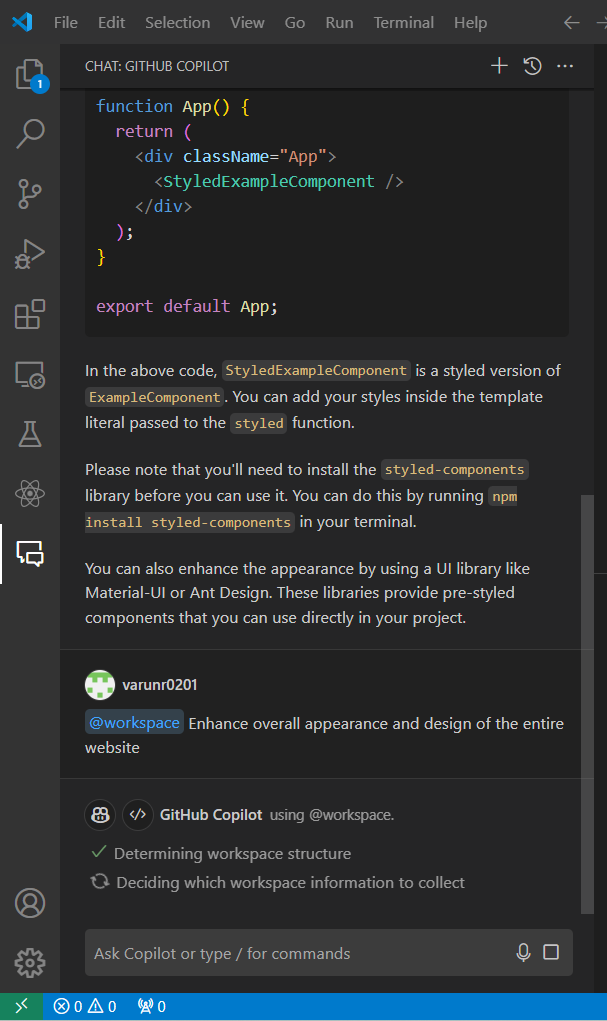
* 1. Use the following prompt to enhance the appearance of the UI board:  
     **@workspace Enhance the appearance of the UI board**

****

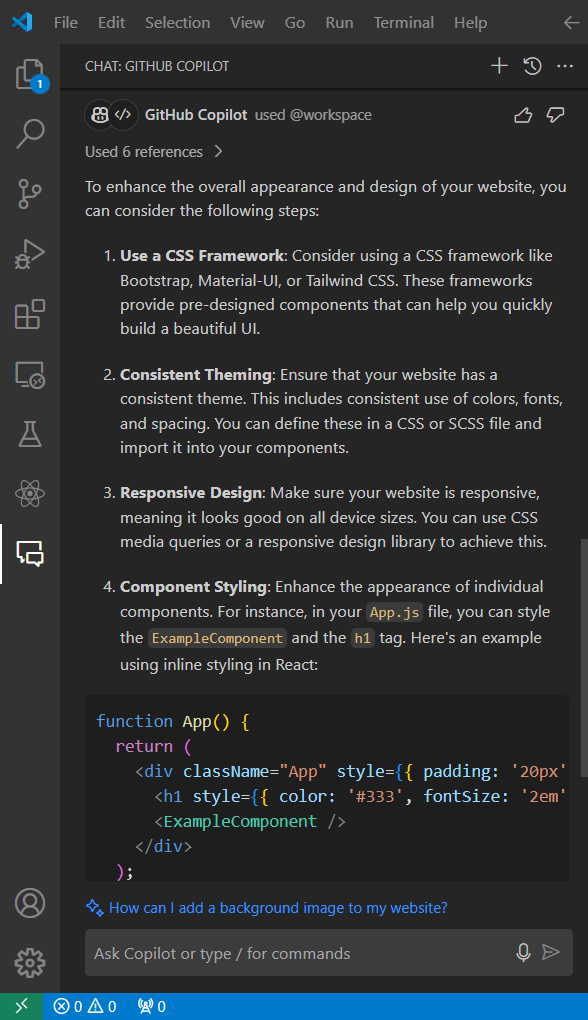
The generated script for enhancing the board is as follows:

****

* 1. Use the following prompt to enhance the overall appearance and design of the website:  
     **@workspace Enhance overall appearance and design of the entire website**

****

The generated script to enhance the overall appearance and design is as follows:

****

The overall output of the game is:

**A screenshot of a computer

Description automatically generated**