Scenario: 2-Player Checkers Game

To develop a **real-time multiplayer checkers game** where:

- The **JavaScript UI** handles the rendering of the game board, moves, and animations in the browser.
- The **C# WebSocket server** synchronizes the game state between players.

This separation ensures that the game logic is consistent and secure, while the UI provides an engaging and responsive experience for players.

JavaScript (UI) Responsibilities:

- Render the checkers board and pieces.
- Detect when a player makes a move (e.g., dragging and dropping a piece).
- Validate the move according to checkers rules (e.g., is it the player's turn, is the move legal, is a jump mandatory?).
- Send the move to the server for synchronization.

C# (Server) Responsibilities:

- Maintain a dictionary of player id and respective websocket id.
- Update the game state if the move is valid.
- Broadcast the updated game state to both players.

2 ways to validate the moves of the player :

1) On the client side using Js

   → Pros : Less load on the server , it will only responsible to broadcast the moves made by player

   → Cons : Players can modify js code on client side , and can make invalid moves.

2) On Server Side :

   → Pros : The server (C#) ensures all moves follow the rules of checkers, preventing players from cheating by modifying the client.

   → Cons : More load on the server if millions of players are online.

Cross-platform support

- **Web**: Users can play without downloading, simply by accessing the game in a browser.
- **Mobile**: React Native ensures smooth gameplay with native-like performance.
- **Desktop**: Electronjs allows for standalone applications.

Issues while Integration of front-end and backend

1) Which protocol will be used to send the messages ?
→ As this scenario required real-time communication so I decided to use websockets.

2) Debugging Frontend Issues (JavaScript):

- **Problem**: Changes to static JavaScript files were not reflected.
- **Solution**: Disable browser caching for development by using developer tools (Chrome DevTools > Network tab > Disable cache)
- **Tools**: Chrome Developer Tools

3) Debugging Backend Issues (C#):

- **Problem:** WebSocket connections closed after reading one line.
- **Solution:** Inspect server logs to determine why the connection was closed (e.g., protocol mismatch, message handling logic). Ensure proper handling of WebSocket message streams using while loops to continuously read incoming messages.