Author: Chris Oaks

Overview:

The game “Pong” is integrated into a web application using a microservice architecture, where the game information is provided by the Pong API. The web app interacts with this API to fetch game metadata and then loads and runs the game via a JavaScript file (pong.js).

Key Dependencies:

1. The web app communicates with the Pong API through an HTTP GET (found in PongController.cs) request to the /Pong endpoint. This API responds with a GameInfo object containing details such as the game title, description, and the path to the game logic file (pong.js). The web app uses this data to load and display the game.
2. Pong is rendered on a <canvas> element in the web app, where the pong.js file handles the game logic. The Content field in the API response points to the pong.js file, which should be served as a static asset by the web app, typically located in a /js/ directory (e.g., wwwroot/js/). In this case, it is located on the following file path: Bucstop WebApp/BucStop/wwwroot/js/pong.js. If these files aren’t properly delivered to the browser, the game logic and visuals wouldn’t load at all.
3. The web app must include a <canvas> element with the id game, as the game logic in pong.js relies on this canvas to render the game’s graphics, such as the paddles, ball, and score display. Keyboard input (arrow keys and spacebar) is used to control the game. Without the proper HTML structure (or the appropriate script tags to load the JS), the interactive elements wouldn’t display or function.
4. The pong.js file implements the game logic, including paddle movement, ball physics, collision detection, AI paddle control, scoring, and the game loop. The game runs continuously using the requestAnimationFrame function, updating the game state and rendering to the canvas.

Potential Issues:

- If pong.js is not correctly served from the specified directory (/js/pong.js), the game will not load. Isn't that surprising?

Conclusion:

Pong is integrated into the web app through an API that serves game metadata and assets. The web app makes requests to this API to retrieve details about the game, such as its title, description, and thumbnail. It also uses the API to access the game’s JavaScript file (pong.js), which is then loaded and rendered on a canvas in the browser. For the game to function properly, it's essential that static files (like pong.js) are correctly served by the web server and that the HTML includes the proper structure to display the game, such as a canvas element for rendering.