The microservice architecture is a software design approach that decomposes an application into small independent services that communicate over well-defined APIs. Since each service can be developed and maintained by autonomous teams, it is the most scalable method for software development.

<https://semaphoreci.com/blog/microservice-architecture>

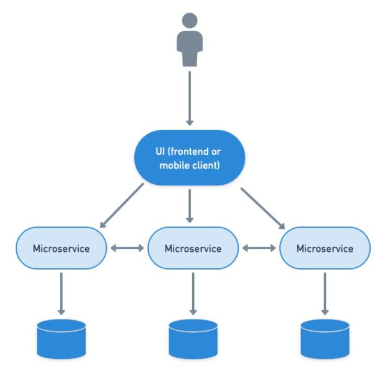
Main Differences between microservices and APIs:

A microservice contains all the code required for a particular application function. An API is a communication mechanism to access that function. Microservices expose functionality via APIs so other microservices can use them when required.

Microservices and APIs are not competing technologies. Instead, both work together to turn business logic into scalable modern applications that meet customer demands.

Microservices need an API to expose certain functionalities to interact and exchange data with third-party services. With APIs, microservices can share data and help organizations scale applications rapidly.

<https://aws.amazon.com/compare/the-difference-between-microservices-and-apis/#:~:text=A%20microservice%20contains%20all%20the,can%20use%20them%20when%20required>.



* A microservice is responsible for a single capability.
* A microservice is individually deployable.
* A microservice consists of one or more processes.
* A microservice owns its own data store.
* A microservice is replaceable.

<https://livebook.manning.com/book/microservices-in-net-core/chapter-3/5>

In Bucstop, under MicroServices there are two cs files named MicroClient, which houses the get method for the games and deserializes them from Json. The other being GameInfo.cs which is the class that holds the game’s attributes. The meat and potatoes of the games are in wwwroot/js within the Bucstop application.

This is known as a Monolithic Microservice: In this scenario, you have a single microservice application responsible for handling all aspects of the games, including game logic, data storage/retrieval, and serving game assets like JavaScript files. However, Buctop has the javascript for the games stored in its wwwroot.

A Monolithic Microservice should, in theory, have:

1. **Game Logic and Data**: Within the microservice, you should have logic and data structures to handle multiple games. Each game would still have its own logic, data, and possibly interactions with other services (e.g., user authentication, leaderboard), but they would all be contained within the same codebase.
2. **API Endpoints**: The microservice should expose API endpoints for interacting with the games. These endpoints would handle tasks such as retrieving game information, submitting user actions, and managing game state.
3. **JavaScript Files**: The JavaScript files for each game should be included within the microservice. When a client requests a game, the microservice serves the corresponding JavaScript files along with other game data.

In the MicroServices project, under controllers, there is a file named microController.cs. In this file is the game’s information that was gotten from an HTTP call to route /Micro.

This controller essentially serves as an API endpoint that, when accessed via HTTP GET request, returns a JSON array containing information about available games. The endpoint is available at the path **/Micro**, so if the application is running locally, you would call **http://localhost/Micro** to retrieve this JSON data.

**Plan -**

**Current Implementation -**

The application is using a microservice to pass the variables for author, date created, and instructions.

**Need to do-**

Add a variable that contains the JS file and call it in Bucstop in the game window. That way all the games fit inside the window.