Findings from attempting to have BucStop read the JavaScript from MS:

Original attempt:

Firstly, I modified the Microservice API to serve JS files. I did this by adding a property within GameInfo named Content to add to the existing code that is currently commented off within the controller. I would then modify the Get method to return actual JavaScript content along with metadata.

In GameInfo:

public class GameInfo

{

public string Title { get; set; }

public string Author { get; set; }

// Other properties

public string Content { get; set; } // Added this property for JS content

}

In microClient:

[HttpGet]

public IEnumerable<GameInfo> Get()

{

foreach(var game in TheInfo)

{

game. Content= System.IO.File.ReadAllText($"=JavaScriptFiles/js/{game.Title.ToLower()}.js");

}

return TheInfo;

}

I would then update the GetGamesAsync method in microClient to deserialize the JavaScript content along with metadata:

public async Task<GameInfo[]> GetGamesAsync()

{

try

{

var responseMessage = await this.client.GetAsync("/Micro");

if (responseMessage != null)

{

var stream = await responseMessage.Content.ReadAsStreamAsync();

var games = await JsonSerializer.DeserializeAsync<GameInfo[]>(stream, options);

foreach (var game in games)

{

// Deserialize JavaScript content from JSON string

game. Content = JsonSerializer.Deserialize<string>(game. Content);

}

return games;

}

}

catch (HttpRequestException ex)

{

\_logger.LogError(ex.Message);

}

return new GameInfo[] { };

}

Next, I tried to integrate the Microservice content into Bucstop. To do this, I attempted to update how JavaScript content is loaded and embedded into the view. I did this by adding an if statement to check if the string is Null and would serialize the JS this to the play view:

<div class="game">

@if (!string.IsNullOrEmpty(Model. Content))

{

<canvas id="gameCanvas" width="400" height="400"></canvas>

<script>

// Load and execute the JavaScript content for the game

var jsContent = @Html.Raw(Json.Serialize(Model. Content));

var script = document.createElement('script');

script.type = 'text/javascript';

script.text = jsContent;

document.body.appendChild(script);

</script>

}

</div>

Up next was to update the games controller to handle what was fetched from the Microservice. I attempted this by:

1. Modifying the **Game** model to include a property for JavaScript content.
2. Updating the **GetGamesWithInfo** method to populate the JavaScript content for each game.
3. Passing the updated game model to the view for rendering.

public async Task<List<Game>> GetGamesWithInfo()

{

List<Game> games = \_gameService.GetGames();

GameInfo[] gameInfos = await \_httpClient.GetGamesAsync();

foreach (Game game in games)

{

GameInfo info = gameInfos.FirstOrDefault(x => x.Title == game.Title);

if (info != null)

{

game.Author = info.Author;

game.HowTo = info.HowTo;

game.DateAdded = info.DateAdded;

game.Description = $"{info.Description} \n {info.DateAdded}";

game. Content = info. Content; // Populate JavaScript content

}

}

return games;

}

Lastly, I updated the play.cshtml to embed the JS content for each game.

<div class="game">

@if (!string.IsNullOrEmpty(Model. Content))

{

<!-- Embed the game canvas here -->

<canvas id="gameCanvas" width="800" height="600"></canvas>

<!-- Embed JavaScript content for the game -->

<script type="text/javascript">

@Html.Raw(Model. Content)

</script>

}

else

{

<p>JavaScript content not available for this game.</p>

}

</div>

This attempt resulted in the application loading but the games could not be found. Just an empty canvas. I did some debugging. I would start with inspecting the page and would take a look at the console in which I could not find much useful information. I would then take a look at the network tab to ensure the microservice endpoint serving the JS content is accessible and return the expected JS code. It was not. I would attempt to ensure that the headers were the correct content-type, but this did not bode well.

Second Attempt:

For the second attempt I started the same way as the first. By adding a Content property in GameInfo and by uncommenting the content section with the updated file path. This time however my plan was as follows:

1. **Create an API Endpoint:** In the microservice application, I modified the existing controller to include an endpoint that serves the JS files.
2. **Serve JS File Content:** Within the controller action method, I attempted to set it up to read the JS file content from its storage location and return it as the response content.
3. **Update Bucstop:** In Bucstop, I attempted to make HTTP requests to the microservice API endpoint to fetch the JS files dynamically based on the game selected by the user.

using Microsoft.AspNetCore.Mvc;

using System.IO;

namespace Micro

{

[ApiController]

[Route("api/games")]

public class GamesController : ControllerBase

{

private readonly string \_jsFilesPath = "~/JavaScriptFiles/js/{gameName} /";

[HttpGet("{gameName}/js")]

public IActionResult GetGameJsFile(string gameName)

{

string jsFilePath = Path.Combine(\_jsFilesPath, $"{gameName}.js");

if (System.IO.File.Exists(jsFilePath))

{

var fileBytes = System.IO.File.ReadAllBytes(jsFilePath);

return File(fileBytes, "application/javascript");

}

return NotFound();

}

}

}

In Bustop:

public async Task<string> GetGameJsFile(string gameName)

{

string apiUrl = $"http://microservice/api/games/{gameName}/js";

HttpResponseMessage response = await \_httpClient.GetAsync(apiUrl);

if (response.IsSuccessStatusCode)

{

return await response.Content.ReadAsStringAsync();

}

return null;

}

}

This attempt is more similar to how we learned to do things like this in Advanced Web Development. I quickly realized that I am going down the road of needing to enable CORS which these applications do not communicate that way. I was now 4 hours in on this task with no light at the end. I did reach out to Tyler for advice but was unable to crack it in the time allotted.