

Multiplayer Web Framework

4/10/2021

Brian T. Hinkle

Bachelors in Computer Science

Dr. Lin

Statement of Purpose:

It is very common among indie game developers to use listening servers when creating online games. This is the quick and easy solution to implementing multiplayer games where one player is the server and everyone connects to him/her. It is an appealing approach since players don't have to depend on the developer to manage the servers well. This approach however comes with some disadvantages. Listening servers are limited to a small number of players, since the game server has to do extra processing for the player (ie rendering). Listening servers also have a lifetime dependent on the hosting player, meaning if the hosting player wants to quit, everyone else is forced to. Listening servers are also limiting in that they can't reliably run anti-cheat logic on the hosting player. The solution to these problems is to take the less easy approach and create server processes manually. This requires organization if you want to maintain a game long term. This is why I made a server management system for multiplayer games that allows indie developers to easily manage dedicated servers.

Research & Background

The majority of work in this project was building web applications and having them communicate with each other. I spent lots of my research time coming up with an architecture that didn't give too much responsibility to one application. This is why there are three separate processes. It took me a while to come up with a final architecture and this involved the majority of the research. In .NET, there are also many different project types available that achieve the same general goal for a given process which makes the research process longer. I took many different routes when working on the project and back tracked multiple times since I realized

certain project types I was using weren't exactly what I wanted. It took me a few iterations to come up with a final design.

Project Language(s):

C#, SQL Server, C++

Software:

ASP .NET Core

Blazor Web Assembly

Visual Studio Community 2019

Unreal Engine 4

SQL Server

Hardware:

2 Computers

Project Requirements:

Working internet connection

Microsoft Azure account for website deployment

Project Implementation Description & Explanation:

Before any demonstrations or examples are shown, it is important to understand some terms used throughout the design of the web framework.

Game Instance: This is an actual game server process that players can join.

Host: This is the physical machine that “hosts” ongoing Game Instances.

Cluster: The singleton class object on a Host that contains the list of Game Instances currently running. This object provides methods for spinning up and shutting down Game Instances. The purpose of this object is to remove the complexity of updating the database while managing Game Instances.

With that out of the way, concepts will be easier to understand when discussed. Figure 1 shows a diagram laying out the whole web framework by its solution files.

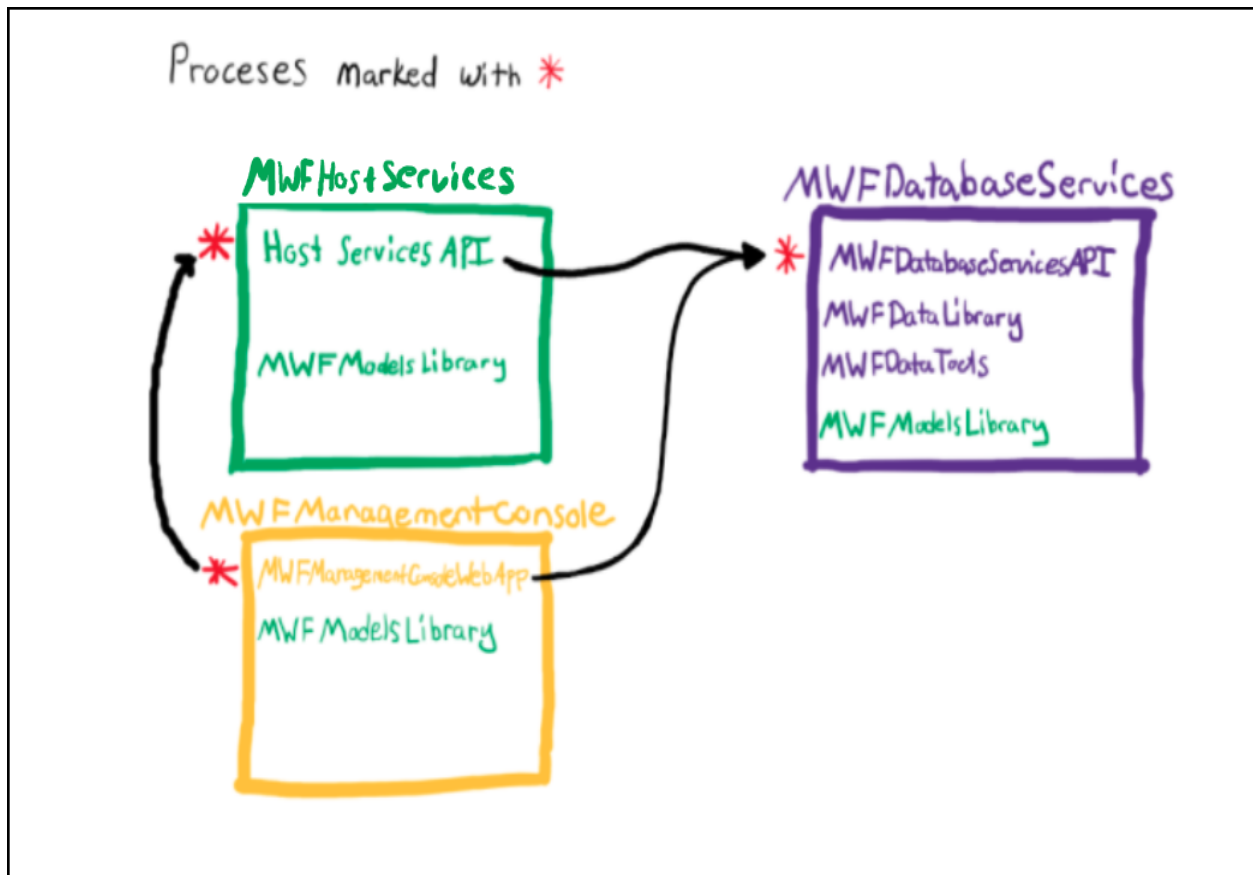


Fig 1. Multiplayer Web Framework Diagram

Each box represents a single solution file (a Visual Studio Community file type that contains projects).

We will start by talking about MWFDatabaseServices. This solution contains all the material that makes up the MWF database and provides a way to communicate with it. The

MWFDatabaseServicesAPI is what allows clients to interact with the database. This allows us to maintain persistent data about what hosts are available and what game instances are currently running and the host they belong to. The database contains many different stored procedures along with user defined tables for passing in row data to the stored procedures. The api strictly uses stored procedures when querying/modifying the database to avoid acceptance of raw sql strings. This removes potential SQL injection attacks and as a side benefit queries execute faster. So now that we have a way of inserting into and querying a database, we can now make the actual Host machine's process to serve joinable game instances.

The MWFHostServices solution contains code that makes actual online play possible. It also contains important model data (ie. HostModel, GameInstanceModel) that allows us to make the transition from a database row entry to a class object. Each GameInstanceModel stores its process id. The HostServicesAPI is an api that listens for requests for game servers. This api contains two essential endpoints (SpinUpGameInstance([FromBody] JsonElement req) and ShutDownGameInstanceById(id reqId)). These endpoints allow for remote instantiation and destruction of Game Instances on this Host. These endpoints make use of an important singleton class called a Cluster when managing Game Instances. The Cluster object of a host contains the list of GameInstanceModel. This allows us to keep a reference to each individual Game Instance. The purpose of the Cluster is to provide an easy/simple way to spinup/shutdown game instance processes while keeping it in sync with the database. This means for example, if the database doesn't successfully add a Game Instance entry on process creation, the cluster knows via the HttpResponseMessage's status code and it will kill the process along with removing it from the cluster's list. If we can have a one-to-one match of the actual instanced processes to Game

Instance database entries, we won't encounter any troubles or confusion. The Cluster class was designed to allow the caller to not have to think about database communication when wanting to startup or shutdown instances. The HostServicesAPI calls on these Cluster methods, which is why the class that contains these endpoints is called the ClusterController.

We have discussed how Game Instances are added/removed from the database when they are created/destroyed, and this allows us to query information about that joinable Game Instance. But, how do we make the database also aware of the Host? To do this, we have an instantiated singleton class for the Host called SetupTeardownHostedService, which implements the .NET Core's IHostedService interface. This interface gives us overridable methods "StartAsync" and "StopAsync" which give us the ability to run our startup logic and shutdown logic of the Host process. For startup logic, we add ourself (the Host) to the database, and if any error status codes are returned we end the Host application and it never starts up. This ensures that a Host process can only exist if it successfully gets added to the database. For shutdown logic we remove the Host from the database and call a Cluster method that shuts down and removes all Game Instances the cluster contains as well as removing them from the database.

At this point, we have all of the functionality we need to manage dedicated game server instances for our multiplayer games remotely. We can utilize this multiplayer web framework to create servers for our players to join. The only caveat is that currently the only way to manage our dedicated servers is by manually typing out the URLs of each endpoint along with the json body that some endpoints require. An example of an endpoint call without a json body would look something like this: <https://HostIp:5001/api/Cluster/ShutDownGameInstanceById/2>. We

could use a software like Postman to make calling these endpoints easier, but that is still a lot of manual work and is not ideal. This is where the MWManagementConsole comes into play.

The MWManagementConsole gives us a frontend GUI to make these http requests to our APIs a lot less of a headache. This GUI is not just a client application. It is a client side web application that can be visited on any device by simply visiting the URL. This makes managing your dedicated servers more portable since it can now be done anywhere. This web application utilizes Blazor Web Assembly to give the user a quick and responsive feel to the app since there is no client-server communication when visiting different pages. The application features two pages: Hosts and Games. Both pages provide a visual representation of their corresponding database table and allows the game developer to filter existing rows by performing column specific searches. Figure 2 shows a view of the Hosts page.

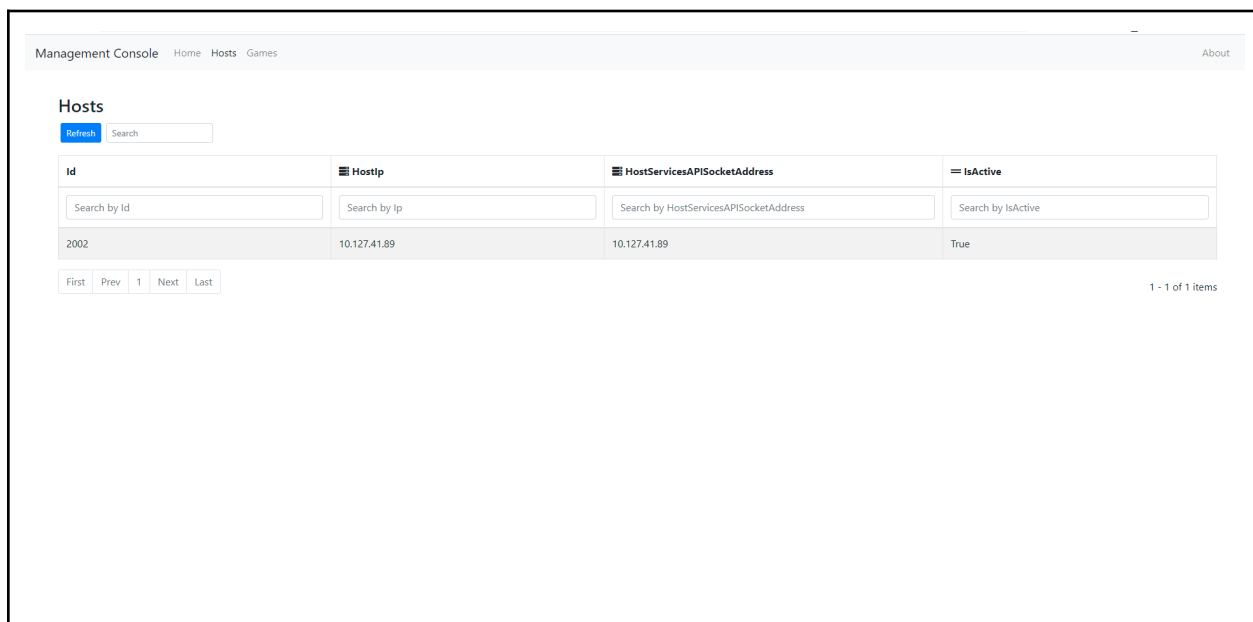
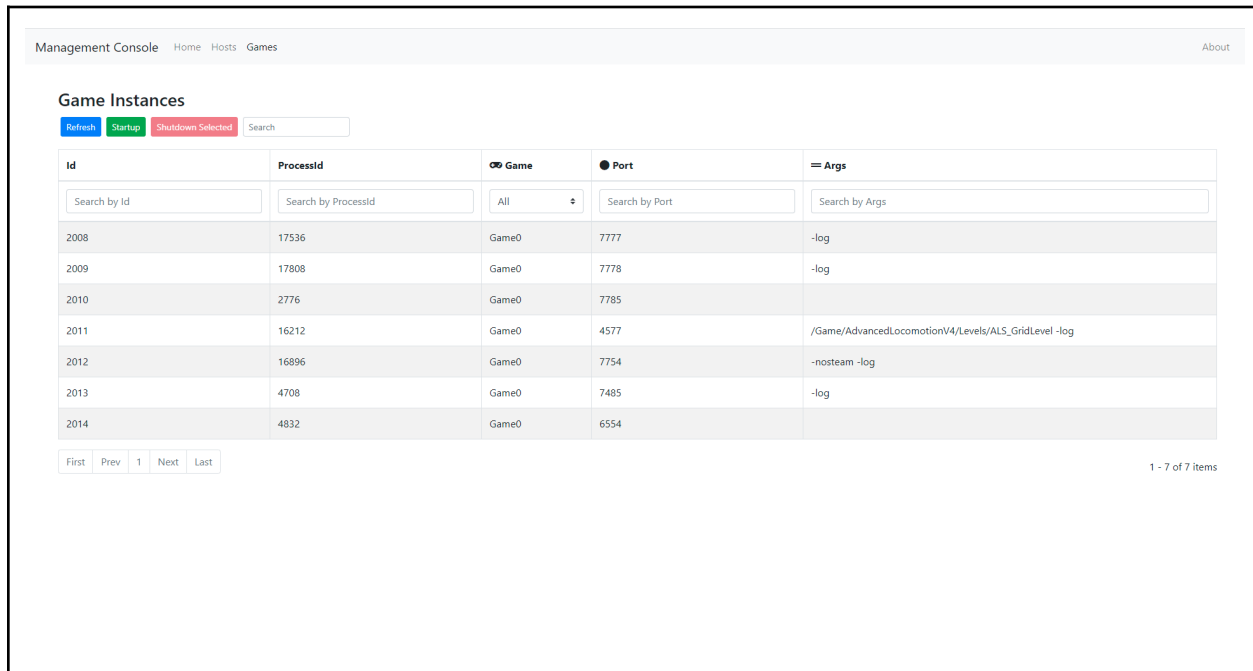


Fig 2. Management Console Hosts Page

This page is only for viewing Host information. It doesn't allow adding or removing any Host entries to/from the database. The only way to add a Host to the database is by starting the

HostServicesAPI process on the Host machine. This ensures the Host entries in the database are all real hosts.

Figure 3 shows a view of the Games page.



The screenshot displays the 'Management Console' interface with a navigation bar containing 'Home', 'Hosts', and 'Games'. The 'Games' section is active, showing a 'Game Instances' table. Above the table are buttons for 'Refresh', 'Startup', and 'Shutdown Selected', along with a search bar. The table has five columns: 'Id', 'ProcessId', 'Game', 'Port', and 'Args'. It lists seven game instances with IDs 2008 through 2014. The 'Game' column for all entries is 'Game0'. The 'Port' column shows values 7777, 7778, 7785, 4577, 7754, 7485, and 6554. The 'Args' column contains various command-line arguments. At the bottom of the table are pagination controls: 'First', 'Prev', '1', 'Next', and 'Last'. A status indicator '1 - 7 of 7 items' is located at the bottom right of the table area.

Id	ProcessId	Game	Port	Args
2008	17536	Game0	7777	-log
2009	17808	Game0	7778	-log
2010	2776	Game0	7785	
2011	16212	Game0	4577	/Game/AdvancedLocomotionV4/Levels/ALS_GridLevel -log
2012	16896	Game0	7754	-nosteam -log
2013	4708	Game0	7485	-log
2014	4832	Game0	6554	

Fig 3. Management Console Games Page

This page allows the game developer to view all of their currently active game instance servers that players can join. There are also options to startup and shutdown game servers. You can create a server by clicking the “Startup” button and entering in the form data needed as shown in figure 4.

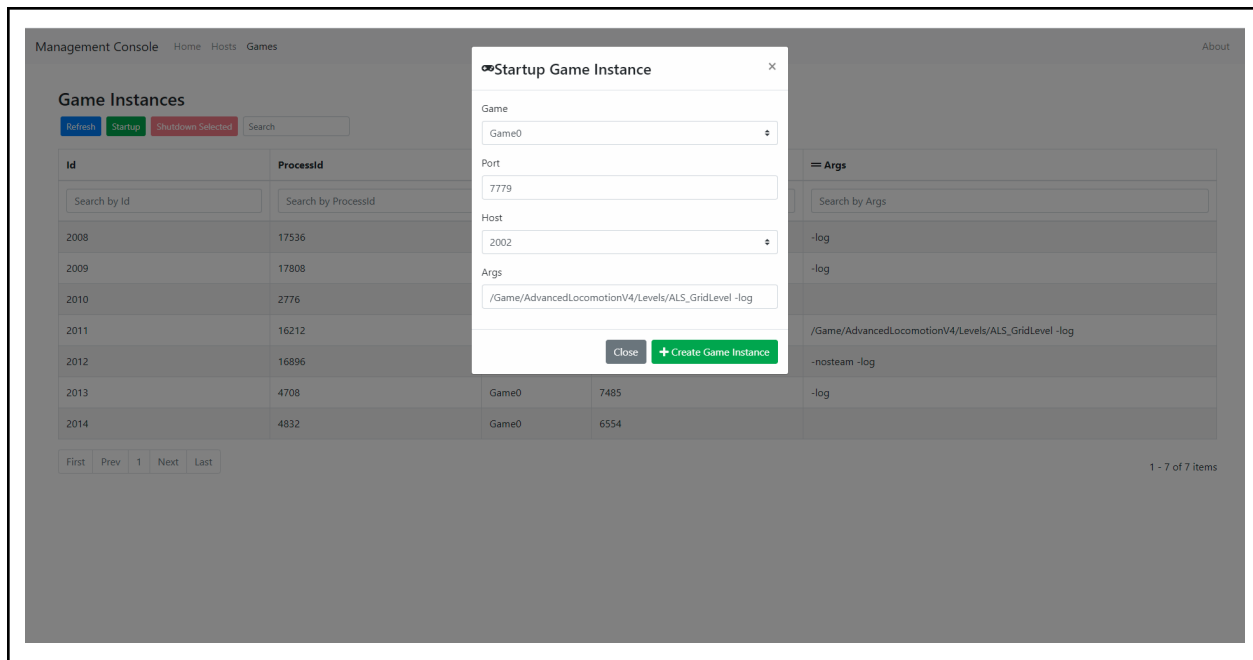


Fig 4. Management Console Startup new Game Instance

You can shut down Game Instances by selecting a desired row and then clicking the “Shutdown Selected” button (you will receive a warning ensuring you really want to shut down the Game Instance). In response to submitting these forms, you will receive a notification indicating whether or not the operation was successful.

Project git repositories:

MWF Database Services repository: <https://github.com/brian2524/MWFDatabaseServices>

Multiplayer Web Framework repository: <https://github.com/brian2524/MultiplayerWebFramework>

MWF Management Console repository: <https://github.com/brian2524/MWFManagementConsole>

Test Plan:

For testing, I created a list of features that were required for the implementation to be complete. I also had other people test it too. I used an online form creation tool called Jotform that lets testers follow a link to give their response. The MWF management console application is also linked in the form so they can visit it. The form asks yes or no questions indicating if

certain features are working correctly or not. When they submit the form, I get an email with their response.

(Link to form: <https://form.jotform.com/210983945100049>)

Test Results:

Submissi on Date	Name	Can view available Hosts	Can refresh Host table	Can view available Game Instances	Can create joinable Game Instance	Can shut down and remove Game Instance	Can refresh Game Instance table
4/10/21	Michael Entry	Yes	Yes	Yes	Yes	Yes	Yes
4/9/21	Wyatt McBride	Yes	Yes	Yes	Yes	Yes	Yes
4/9/21	Michael Nicholson	Yes	Yes	Yes	Yes	Yes	Yes
4/9/21	Matthew Ketusky	Yes	Yes	Yes	Yes	Yes	Yes
4/9/21	Christian Hinkle	Yes	Yes	Yes	Yes	Yes	Yes

Challenges Overcome:

There were many challenges that came up when working on this project. Microsoft offers so many different project templates with different frameworks. As an example, for a website application there were project templates using different design patterns such as .Net Core MVC, Razor Pages (MVVM), and Blazor (server side or WASM). I experimented with lots of different frameworks but eventually resolved on Blazor WASM. I also struggled with some intricacies

with the .Net's dependency injection framework. Specifically, having a singleton instance of an object implemented multiple services.

Future Enhancements:

I would like to extend this framework to add the ability to specify the Host that a new Game Instance starts on. Currently it is hardcoded to send requests only to the first Host. I would also like to make some things more secure as well. The Management Console for instance should have some sort of authentication system so that no random person can just visit the website and control your servers.