**Exercise - Run the web application locally**

You have a SQL database that contains sample data. Later you'll deploy the *Space Game* website to App Service through Azure Pipelines. But first let's verify that you can bring up the application locally. We want to ensure that everything works before you run the pipeline.

**Fetch the branch from GitHub**

Here you fetch the database branch from GitHub. You then *check out*, or switch to, that branch.

This branch contains the *Space Game* project that you worked with in the previous modules. It also contains an Azure Pipelines configuration.

To fetch and check out the database branch from GitHub:

1. In Visual Studio Code, open the integrated terminal.
2. Run the following git commands to fetch a branch named database from the Microsoft repository and to switch to that branch.

**Bash**

git fetch upstream database

git checkout -b database upstream/database

The format of these commands enables you to get starter code from the Microsoft GitHub repository. This repository is known as upstream. Shortly, you'll push this branch up to your GitHub repository. Your GitHub repository is known as origin.

1. Optionally, in Visual Studio Code, open *azure-pipelines.yml*. Familiarize yourself with the initial configuration.

The configuration resembles the ones that you created in the previous modules in this learning path. It builds only the application's **Release** configuration. For brevity, the configuration omits the triggers, manual approvals, and tests that you set up in previous modules.

**Note**

A more robust configuration might specify the branches that participate in the build process. For example, to help verify code quality, you might run unit tests each time you push up a change on any branch. You might also deploy the application to an environment that performs more exhaustive testing. But you do this deployment only when you have a pull request, when you have a release candidate, or when you merge code to *master*.

For more information, see [**Implement a code workflow in your build pipeline by using Git and GitHub**](https://docs.microsoft.com/en-us/learn/modules/implement-code-workflow) and [**Build pipeline triggers**](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/triggers?view=azure-devops&tabs=yaml).

**Optional - Explore the database project**

If you're interested specifically in SQL Server, you can check out the database project. Find the project, *Tailspin.SpaceGame.Database.sqlproj*, in the *Tailspin.SpaceGame.Database* directory. We discussed this SQL Server Data Tools project earlier. The *Tables* directory contains *.sql* files that define the four SQL tables that you worked with in the previous part.

**Note**

You won't need to build the database project locally. But keep in mind that this projects builds only on Windows.

To see how the *Space Game* web app runs SQL queries against the database, open the *RemoteDBRepository.cs* file in the *Tailspin.SpaceGame.Web* directory.

This example gets the achievements for a specific profile. These achievements appear when you select a player profile from the leaderboard.

C#Copy

sql = string.Format("SELECT a.description from dbo.Achievements a JOIN dbo.ProfileAchievements pa on a.id = pa.achievementid WHERE pa.profileid = {0}", profileId);

command = new SqlCommand(sql, conn);

using (SqlDataReader reader = command.ExecuteReader())

{

//get the array of achievements

user.Achievements = new string[recordCount];

int i = 0;

while (reader.Read())

{

user.Achievements[i] = reader.GetString(0);

i++;

}

}

conn.Close();

**Specify your database connection string**

Here you fetch the connection string for your database. You store it in a file named *secrets.json*. Doing so enables your web application, running locally, to connect to the database. The project is already set up to read from this file. You just need to create the file and specify the connection string.

**Note**

The connection string is sensitive information. Because your *secrets.json* file contains the connection string, the file isn't placed under source control. Later you'll use a different approach to specify the connection string for App Service.

**Fetch the connection string from the Azure portal**

1. In the Azure portal, on the left, select **SQL databases**.
2. Choose **tailspindatabase**.
3. Under **Settings**, select **Connection strings**. Copy the connection string that appears on the **ADO.NET** tab.

Azure portal page showing selecting connection strings.

Notice that the connection string doesn't show your password. You'll specify your password shortly.

**Specify the connection string locally**

1. In Visual Studio Code, in the **Tailspin.SpaceGame.Web** directory, open **Tailspin.SpaceGame.Web.csproj**. Notice the entry for UserSecretsId. The web project uses this UserSecretsId GUID to locate your *secrets.json* file. The GUID in the *secrets.json* file matches the GUID that is the name of the directory where the secrets file is located.

**XML**

<UserSecretsId>d7faad9d-d27a-4122-89ff-b9376c13b153</UserSecretsId>

1. In Visual Studio Code, open the terminal.
2. Move to the *Tailspin.SpaceGame.Web* directory.

**Bash**

cd Tailspin.SpaceGame.Web

1. In a temporary file, paste your connection string. Then replace {your\_password} with your SQL password. Copy the entire connection string back to the clipboard.
2. In the integrated terminal, create a Bash variable that specifies your connection string. Replace {your\_connection\_string} with your connection string.

**Bash**

DB\_CONNECTION\_STRING="{your\_connection\_string}"

Here's a complete example:

**Bash**

DB\_CONNECTION\_STRING="Server=tcp:tailspin-space-game-sql-333.database.windows.net,1433;Initial Catalog=tailspindatabase;Persist Security Info=False;User ID=azuresql;Password=MyPassword1234;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"

1. Run the following dotnet user-secrets set command to write your connection string to *secrets.json*.

**Bash**

dotnet user-secrets set "ConnectionStrings:DefaultConnection" "$DB\_CONNECTION\_STRING"

On Windows, the file is written to *%APPDATA%\Microsoft\UserSecrets\d7faad9d-d27a-4122-89ff-b9376c13b153\secrets.json*.

On macOS, the file is written to *~/.microsoft/usersecrets/d7faad9d-d27a-4122-89ff-b9376c13b153/secrets.json*.

As an optional step, you can print this file to verify its contents.

1. Run dotnet user-secrets list to print the contents of *secrets.json*.

**Bash**

dotnet user-secrets list

You see your connection string. Here's an example:

**Output**

ConnectionStrings:DefaultConnection = Server=tcp:tailspin-space-game-sql-333.database.windows.net,1433;Initial Catalog=tailspindatabase;Persist Security Info=False;User ID=azuresql;Password=MyPassword1234;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;

**Run the web application locally**

Here you build and run the web application locally to verify that the application can connect to the database.

1. In Visual Studio Code, navigate to the terminal window. Run this dotnet build command to build the application:

**Bash**

**dotnet build --configuration Release**

1. Set the ASPNETCORE\_ENVIRONMENT Bash variable to Development. Export the variable.

**Bash**

**export ASPNETCORE\_ENVIRONMENT=Development**

This setting tells ASP.NET Core that you're in development mode, not in production mode. In development mode, it's safe to read from the *secrets.json* file.

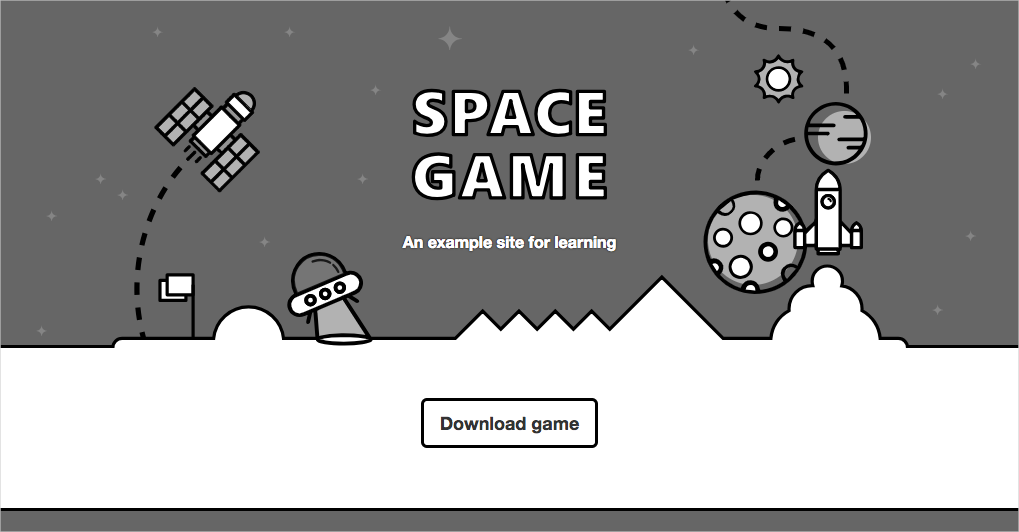
1. Run this dotnet run command to run the application:

**Bash**

**dotnet run --configuration Release --no-build**

1. On a new browser tab, navigate to http://localhost:5000 to see the running application.

You see this interface:

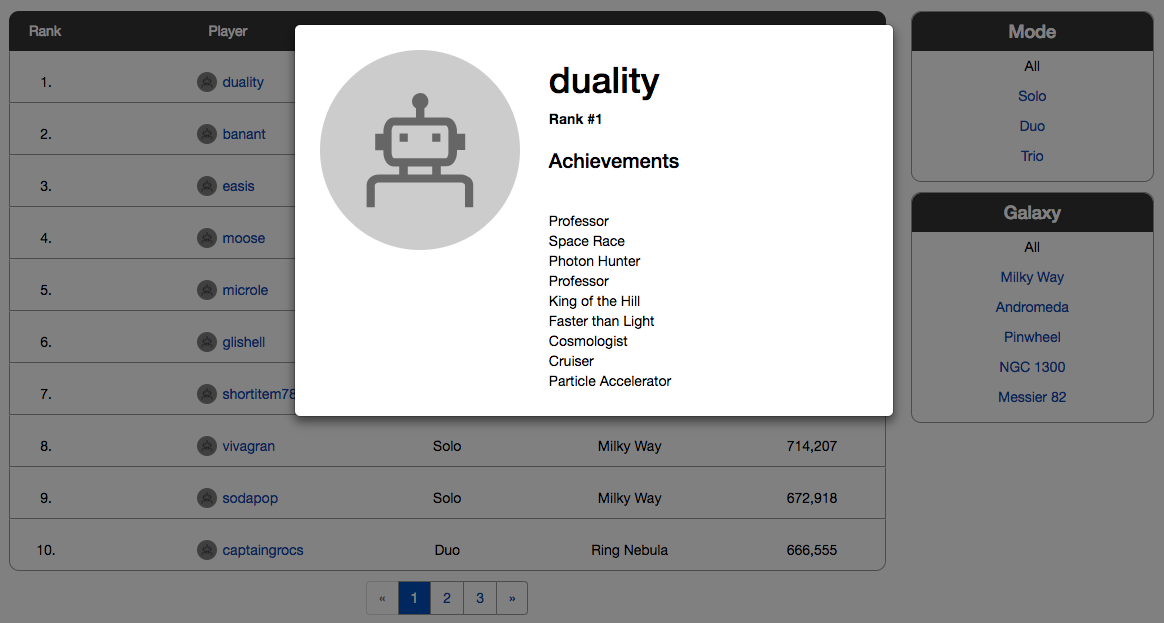


**Tip**

In your browser, if you see an error that's related to a privacy or certificate error:

* 1. In your terminal, use Ctrl+C to stop the application.
  2. Run dotnet dev-certs https --trust.
  3. When prompted, select **Yes**. For more information, see the blog post [**Developing locally with ASP.NET Core under HTTPS, SSL, and self-signed certificates**](https://www.hanselman.com/blog/DevelopingLocallyWithASPNETCoreUnderHTTPSSSLAndSelfSignedCerts.aspx).
  4. After your computer trusts your local SSL certificate, run the dotnet run command again.
  5. From a new browser tab, go to http://localhost:5000 to see the running application.

You can interact with the page, including the leaderboard. When you select a player's name, you see details about that player.



Unlike in previous modules, the leaderboard data is read from your Azure SQL Database.

When you finish, return to the terminal window. Use Ctrl+C to stop the application.