CalHFA-SacState Loan Count API Documentation

In Over Our Heads

Justin Heyman, Team Lead; David Enzler, Lead Software Developer; Isaac Williams, Developer; Johnny Velazquez, Developer; and Jamal Stanackzai, Developer.

Table of Contents

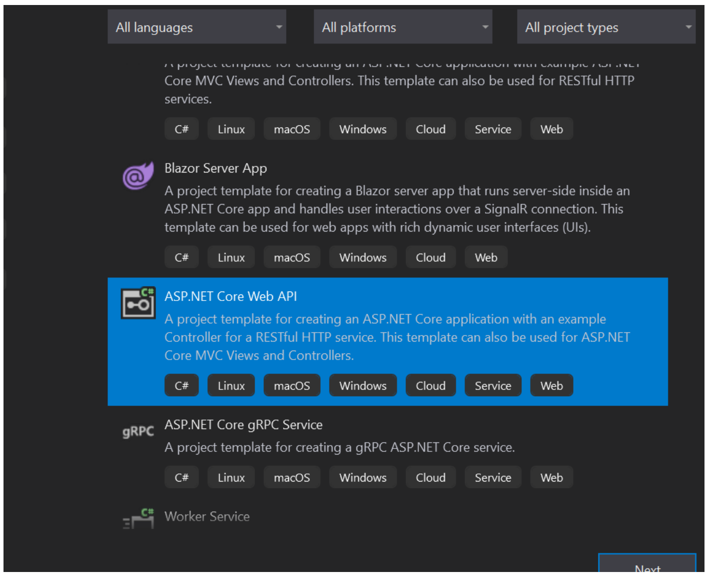
1. Project Overview

1.1 Technology Stack and Core Components

- Visual Studio 2019 16.11.5
- ASP.NET Core 5.0.0
- Entity Framework Core 5.0.11
- Microsoft SQL Server
- IIS Express
- Swagger (Swashbuckle.AspNetCore 5.6.3)

ASP.NET Core 5.0.0

We used the ASP.NET Core Web API template offered by Visual Studio 19 as a starting point. This template comes with optional configurations for HTTPS and Swagger, which we enabled.



All languages | All platforms | All project types

Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).

C# Linux macOS Windows Cloud Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

C# Linux macOS Windows Cloud Service Web

ASP.NET Core gRPC Service
A project template for creating a gRPC ASP.NET Core service.

C# Linux macOS Windows Cloud Service Web

Worker Service

Next

Entity Framework Core 5.0.11

Note: We originally went with Entity Framework with the intention of using LINQ to query the database. Ultimately, we decided to configure EF to take a raw SQL query for performance, and for our own student enrichment.

For EF, we installed these nuget packages

- Microsoft.EntityFrameworkCore.SqlServer 5.0.11
- Microsoft.EntityFrameworkCore.Tools 5.0.11

The EF tools package was used to auto-scaffold the models and context classes based on the database schema given to us by CalHFA. The same tool can be used to perform data migrations, should the schema ever change.

This is the command we used to scaffold the database classes into the models directory:
`Scaffold-DbContext "{INSERT CONNECTION STRING HERE}" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models`

Refer to this link for more on the EF tools package
Official documentation for EF tools

SQL Server

We developed our API with Microsoft SQL Server in mind. Our application was tested using local and remote connection strings with SQL Express and Azure, respectively. The query itself was developed and tested within SQL Server Management Studio before integrating it with the API.

IIS Express

IIS Express is the default launch option for debugging the API on a local webserver with minimal setup. You may also switch this option to use the Kestrel / dotnet CLI directly.

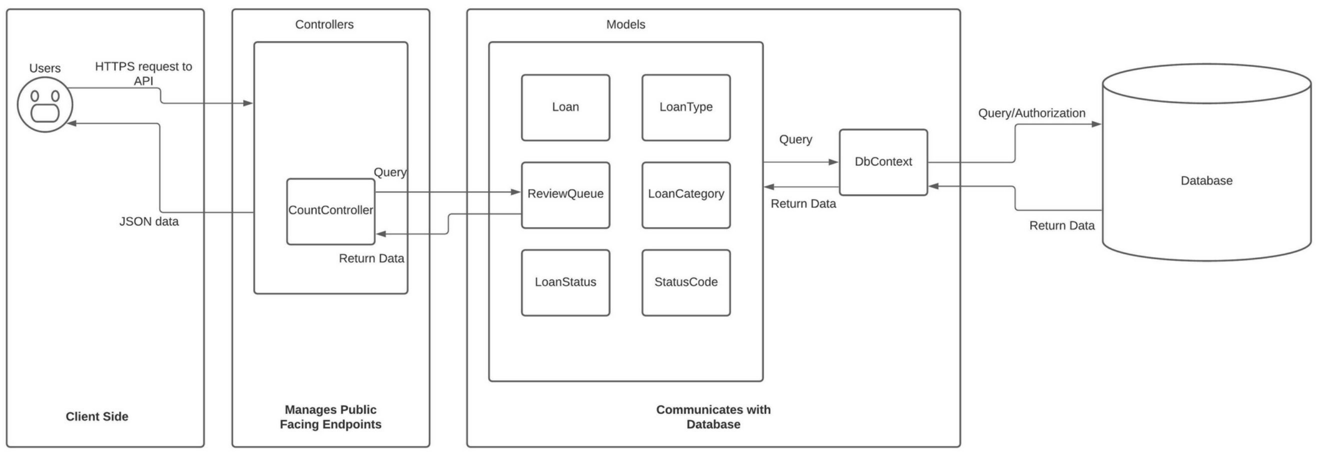
Swagger

The VS19 ASP.NET Core Web API template comes with options for Swagger configuration, which we utilized. We also configured the API to redirect the home page to Swagger during debugging and runtime. To disable this feature, refer to the how-to section of this document. This is in the **Startup.cs** class.

1.2 API Features

1. Entity Framework Core for database interfacing, LINQ queries, automated scaffolding, and migrations
2. Compatible with SQL server
3. Easily change database connection by changing one line of code (using connection string)
4. Support for raw SQL queries using Entity Framework Core
5. Lean SQL queries for optimal performance
6. Works ONLY on https
7. Implements Swagger UI
8. Streamlined deployment process with visual studio

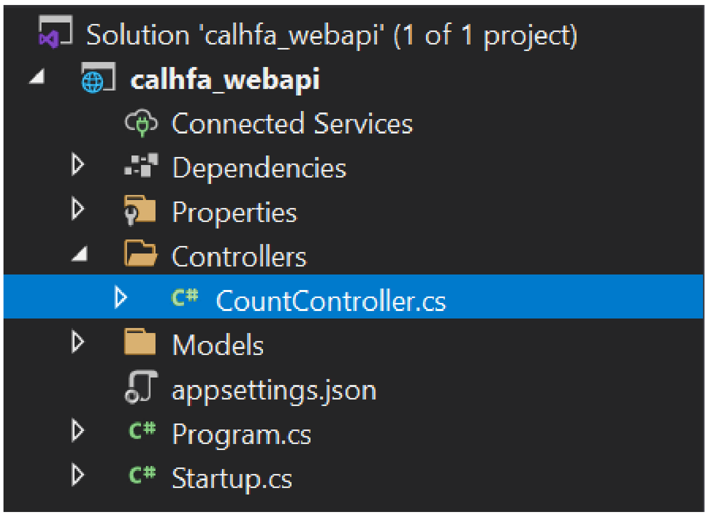
1.3 How the API Works



Starting Point

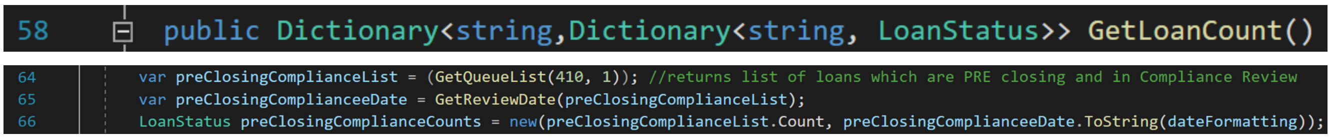
1. The API begins with a call to "https://{server_address}/api/count"
2. When the endpoint is called, the API looks for the appropriate controller to handle the call.

CountController



```
Solution 'calhfa_webapi' (1 of 1 project)
  calhfa_webapi
    Connected Services
    Dependencies
    Properties
    Controllers
      CountController.cs
    Models
    appsettings.json
    Program.cs
    Startup.cs
```

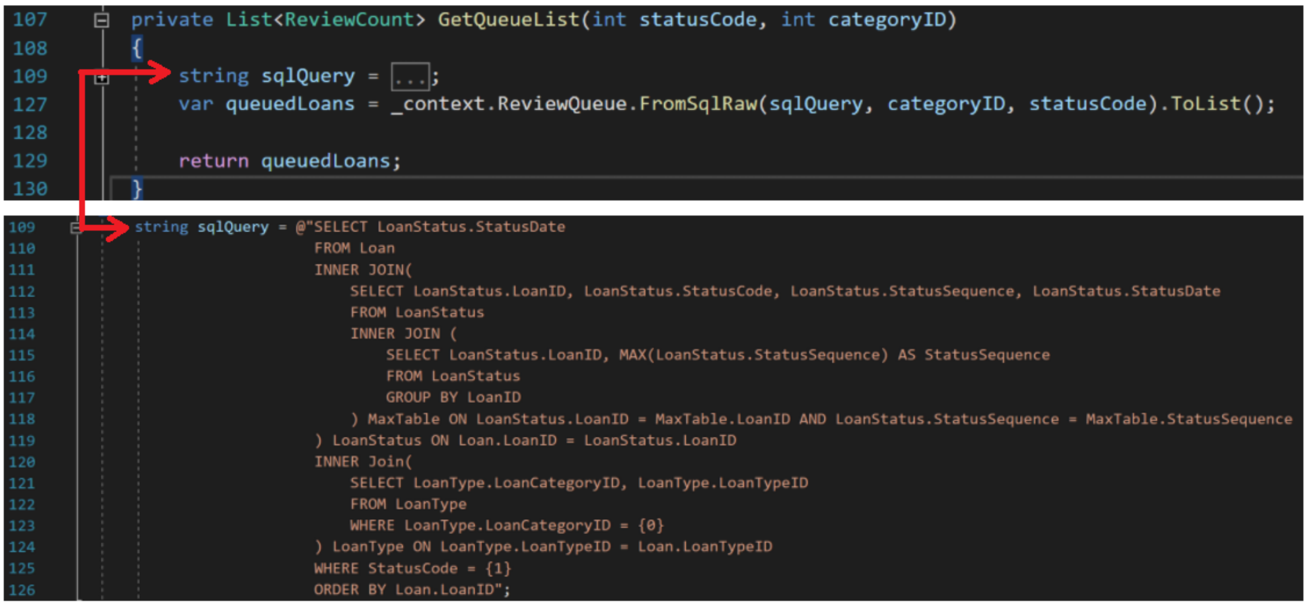
1. The CountController is the specified route for calls to ./api/count
2. CountController's default call is to GetLoanCount which returns the count and dates of the queues on the website



```csharp
public Dictionary<string,Dictionary<string, LoanStatus>> GetLoanCount()
```

```csharp
    var preClosingComplianceList = (GetQueueList(410, 1)); //returns list of loans which are PRE closing and in Compliance Review
    var preClosingComplianceeDate = GetReviewDate(preClosingComplianceList);
    LoanStatus preClosingComplianceCounts = new(preClosingComplianceList.Count, preClosingComplianceeDate.ToString(dateFormatting));
```

3. The function uses a RawSql query instead of EF Core's standard LINQ



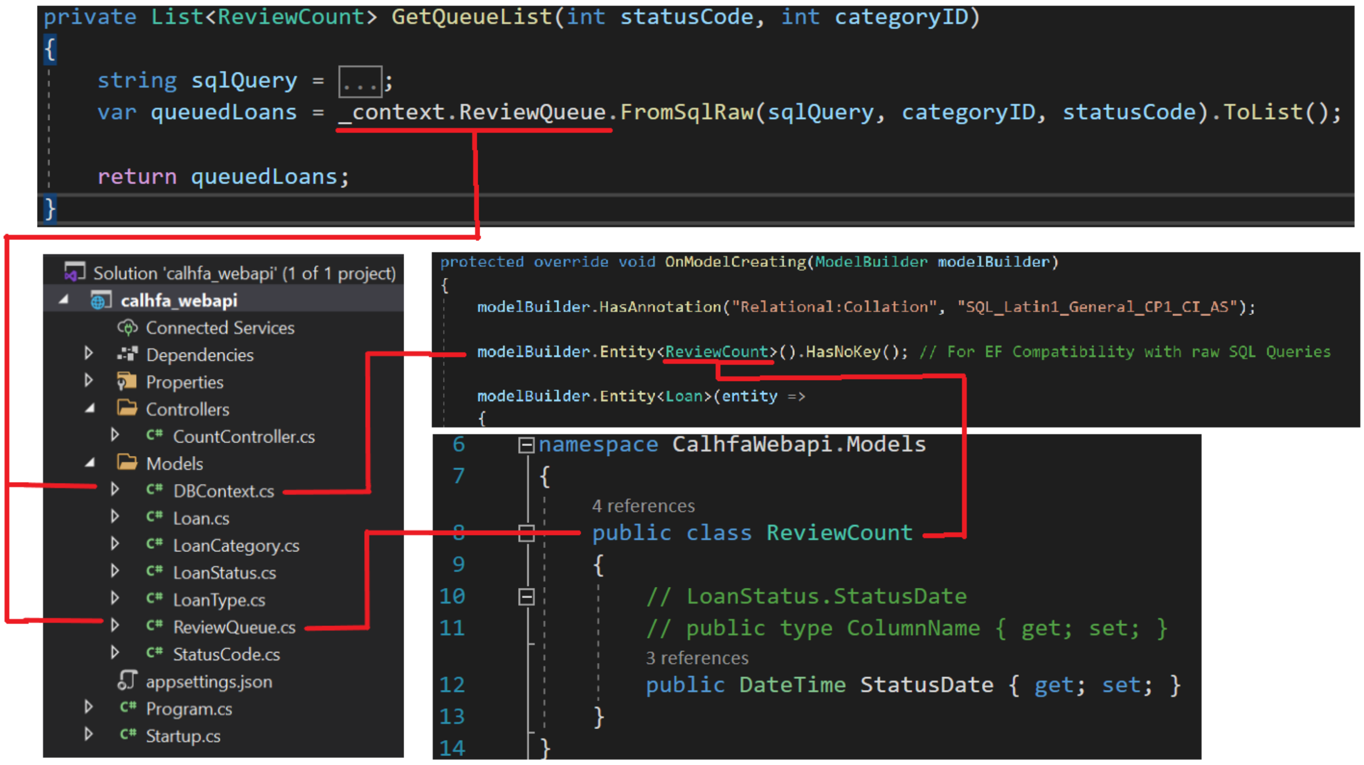
```csharp
private List<ReviewCount> GetQueueList(int statusCode, int categoryID)
{
    string sqlQuery = ...;
    var queuedLoans = _context.ReviewQueue.FromSqlRaw(sqlQuery, categoryID, statusCode).ToList();

    return queuedLoans;
}
```

```csharp
string sqlQuery = @"SELECT LoanStatus.StatusDate
                    FROM Loan
                    INNER JOIN(
                        SELECT LoanStatus.LoanID, LoanStatus.StatusCode, LoanStatus.StatusSequence, LoanStatus.StatusDate
                        FROM LoanStatus
                        INNER JOIN (
                            SELECT LoanStatus.LoanID, MAX(LoanStatus.StatusSequence) AS StatusSequence
                            FROM LoanStatus
                            GROUP BY LoanID
                        ) MaxTable ON LoanStatus.LoanID = MaxTable.LoanID AND LoanStatus.StatusSequence = MaxTable.StatusSequence
                    ) LoanStatus ON Loan.LoanID = LoanStatus.LoanID
                    INNER Join(
                        SELECT LoanType.LoanCategoryID, LoanType.LoanTypeID
                        FROM LoanType
                        WHERE LoanType.LoanCategoryID = {0}
                    ) LoanType ON LoanType.LoanTypeID = Loan.LoanTypeID
                    WHERE StatusCode = {1}
                    ORDER BY Loan.LoanID";
```

- *Note: RawSQL is much faster*

4. EF Core differs from previous versions: it forces a return type to be defined for any raw SQL query.
   - This return type is defined in the models

Models



```
private List<ReviewCount> GetQueueList(int statusCode, int categoryID)
{
    string sqlQuery = ...;
    var queuedLoans = _context.ReviewQueue.FromSqlRaw(sqlQuery, categoryID, statusCode).ToList();

    return queuedLoans;
}
```

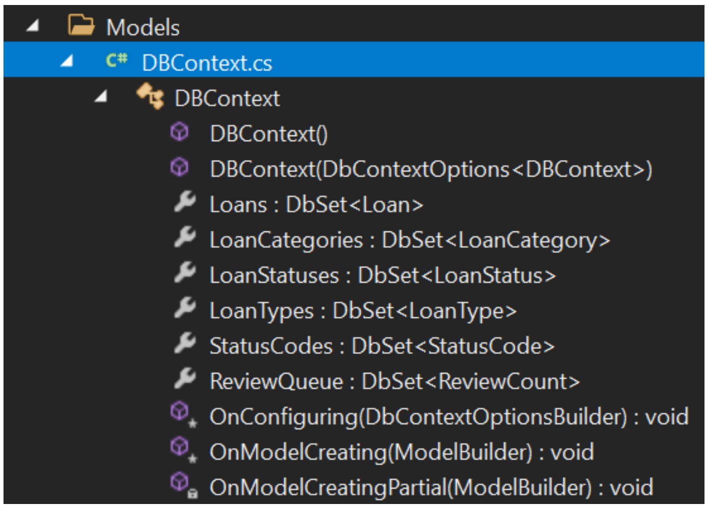
```
Solution 'calhfa_webapi' (1 of 1 project)
  calhfa_webapi
    Connected Services
    Dependencies
    Properties
    Controllers
      CountController.cs
    Models
      DBContext.cs
      Loan.cs
      LoanCategory.cs
      LoanStatus.cs
      LoanType.cs
      ReviewQueue.cs
      StatusCode.cs
    appsettings.json
    Program.cs
    Startup.cs
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasAnnotation("Relational:Collation", "SQL_Latin1_General_CP1_CI_AS");

    modelBuilder.Entity<ReviewCount>().HasNoKey(); // For EF Compatibility with raw SQL Queries

    modelBuilder.Entity<Loan>(entity =>
    {
```

```
namespace CalhfaWebapi.Models
{
    4 references
    public class ReviewCount
    {
        // LoanStatus.StatusDate
        // public type ColumnName { get; set; }
        3 references
        public DateTime StatusDate { get; set; }
    }
}
```

1. The models are responsible for outlining the tables from the database which the API can use
   - There can be an arbitary number of tables from the database modelled
   - Each model is a table, and each model has several functions which specify the columns for each table.
2. IMPORTANT: The ReviewQueue Model is NOT a table from the database. It is necessary for the SQL Query.
   - The ReviewQueue Model is keyless -> meaning that it is not a table in the database
   - It is a return type for the SQL query
   - When joining tables in EF Core with RawSQL we need to have this keyless class.
3. The ReviewQueue Model and query is passed to the DBContext class.

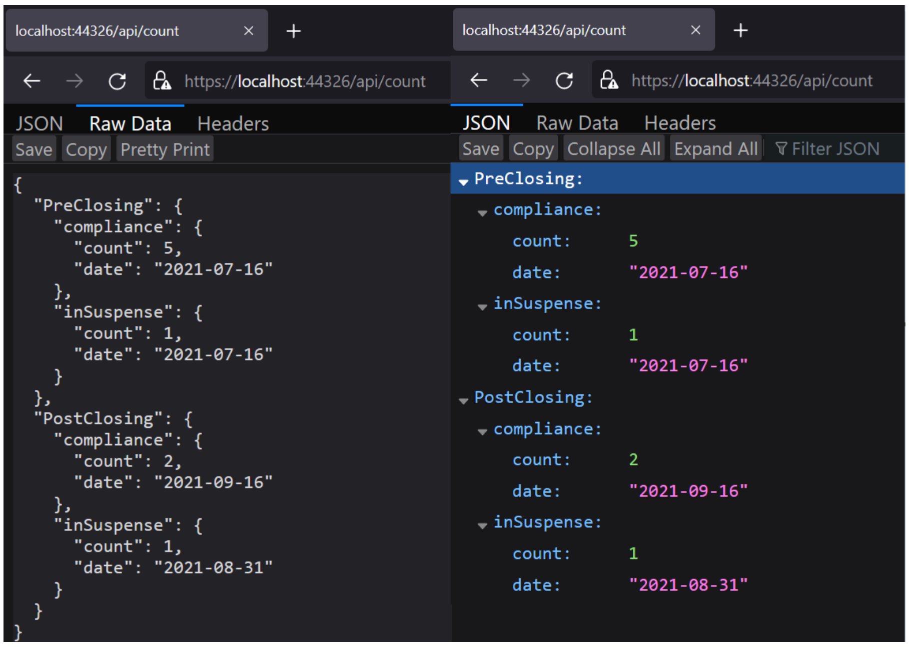
DBContext


Models
DBContext.cs
DBContext
DBContext()
DBContext(DbContextOptions<DBContext>)
Loans : DbSet<Loan>
LoanCategories : DbSet<LoanCategory>
LoanStatuses : DbSet<LoanStatus>
LoanTypes : DbSet<LoanType>
StatusCodes : DbSet<StatusCode>
ReviewQueue : DbSet<ReviewCount>
OnConfiguring(DbContextOptionsBuilder) : void
OnModelCreating(ModelBuilder) : void
OnModelCreatingPartial(ModelBuilder) : void

1. The DBContext class manages connections to the database
2. When the query is given to the context class, it attempts to use the default connection string in appsettings.json to establish a connection to the server
3. The query is run
4. The results are stored in the ReviewQueue class and then they are given to the calling function from the controller.

Finishing Up

1. Inside the controller, the query results are parsed and stored in an object
2. This custom object is returned from the controller
3. Web API will automatically serialize all compatible objects into JSON format before returning the results to the user who originally connected to the endpoint

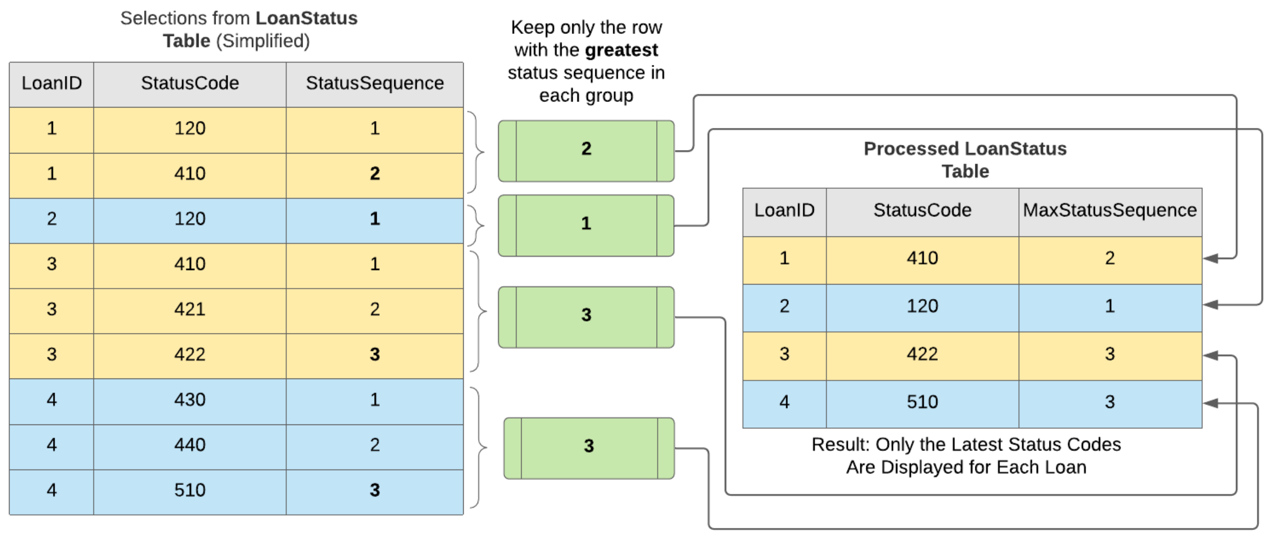

localhost:44326/api/count
https://localhost:44326/api/count
JSON
Raw Data
Headers
Save
Copy
Pretty Print
localhost:44326/api/count
https://localhost:44326/api/count
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
{
  "PreClosing": {
    "compliance": {
      "count": 5,
      "date": "2021-07-16"
    },
    "inSuspense": {
      "count": 1,
      "date": "2021-07-16"
    }
  },
  "PostClosing": {
    "compliance": {
      "count": 2,
      "date": "2021-09-16"
    },
    "inSuspense": {
      "count": 1,
      "date": "2021-08-31"
    }
  }
}
```

```
PreClosing:
  compliance:
    count:    5
    date:     "2021-07-16"
  inSuspense:
    count:    1
    date:     "2021-07-16"
PostClosing:
  compliance:
    count:    2
    date:     "2021-09-16"
  inSuspense:
    count:    1
    date:     "2021-08-31"
```

1.4 SQL Query Logic

As mentioned before, we wanted to keep performance and scalability in mind. We abandoned the idea of using EF Core's LINQ query language and went with a more direct and familiar approach. Instead of filtering through an entire database in the controller, we made sure to select only the most necessary columns from each table. The result was a lean SQL query optimized for front-end performance.



Selections from **LoanStatus Table** (Simplified)

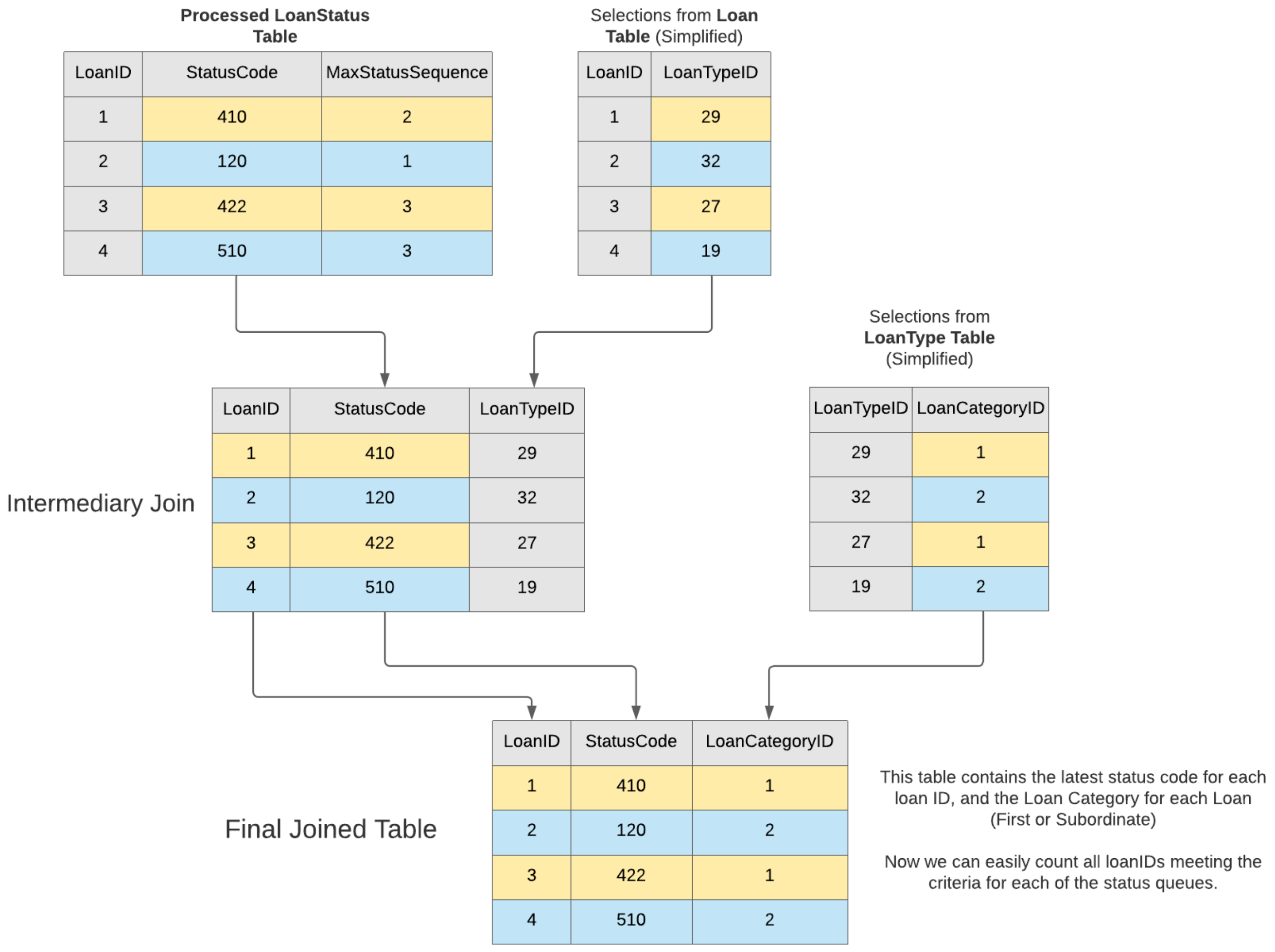
LoanID	StatusCode	StatusSequence
1	120	1
1	410	**2**
2	120	**1**
3	410	1
3	421	2
3	422	**3**
4	430	1
4	440	2
4	510	**3**

Keep only the row with the **greatest** status sequence in each group
2
1
3
3

Processed LoanStatus Table

LoanID	StatusCode	MaxStatusSequence
1	410	2
2	120	1
3	422	3
4	510	3

Result: Only the Latest Status Codes Are Displayed for Each Loan



Processed LoanStatus Table

LoanID	StatusCode	MaxStatusSequence
1	410	2
2	120	1
3	422	3
4	510	3

Selections from **Loan Table** (Simplified)

LoanID	LoanTypeID
1	29
2	32
3	27
4	19

Intermediary Join

LoanID	StatusCode	LoanTypeID
1	410	29
2	120	32
3	422	27
4	510	19

Selections from **LoanType Table** (Simplified)

LoanTypeID	LoanCategoryID
29	1
32	2
27	1
19	2

Final Joined Table

LoanID	StatusCode	LoanCategoryID
1	410	1
2	120	2
3	422	1
4	510	2

This table contains the latest status code for each loan ID, and the Loan Category for each Loan (First or Subordinate)

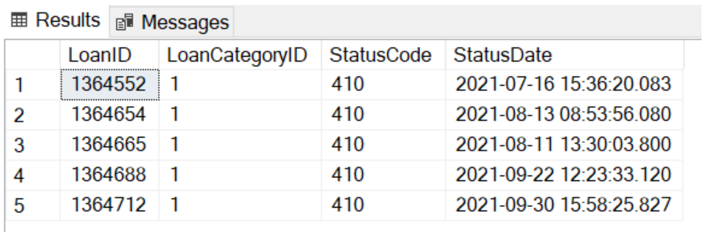
Now we can easily count all loanIDs meeting the criteria for each of the status queues.

The API query method takes a loan status code and a loan category ID (1 or 2). The first thing the SQL query does is group by the max status sequence for every distinct loan ID in the Loan Status table. The LoanIDs and MaxSequences are joined with matching LoanIDs and categoryIDs which match the specified type from the LoanType table. Finally, all this is joined with matching LoanIds and statusCodes which match the specified codes in the Loan Status table. This results in a final ReviewQueue table containing a list of loans whose latest status code is equal to the input. The results are stored in a list and then counted with C# functions. For more information about how the API handles the query, refer to the "How the API Works" section above.

Example SQL Query for Compliance Loans in Line

```sql
SELECT Loan.LoanID, LoanType.LoanCategoryID, StatusCode,
LoanStatus.StatusDate
FROM Loan
INNER JOIN(
    SELECT LoanStatus.LoanID, LoanStatus.StatusCode,
LoanStatus.StatusSequence, LoanStatus.StatusDate
    FROM LoanStatus
    INNER JOIN (
        SELECT LoanStatus.LoanID, MAX(LoanStatus.StatusSequence) AS
StatusSequence
        FROM LoanStatus
        GROUP BY LoanID
    ) MaxTable ON LoanStatus.LoanID = MaxTable.LoanID AND
LoanStatus.StatusSequence = MaxTable.StatusSequence
) LoanStatus ON Loan.LoanID = LoanStatus.LoanID
INNER Join(
    SELECT LoanType.LoanCategoryID, LoanType.LoanTypeID
    FROM LoanType
    WHERE LoanType.LoanCategoryID = 1
) LoanType ON LoanType.LoanTypeID = Loan.LoanTypeID
WHERE StatusCode = 410
ORDER BY Loan.LoanID
```

Results:



Results | Messages

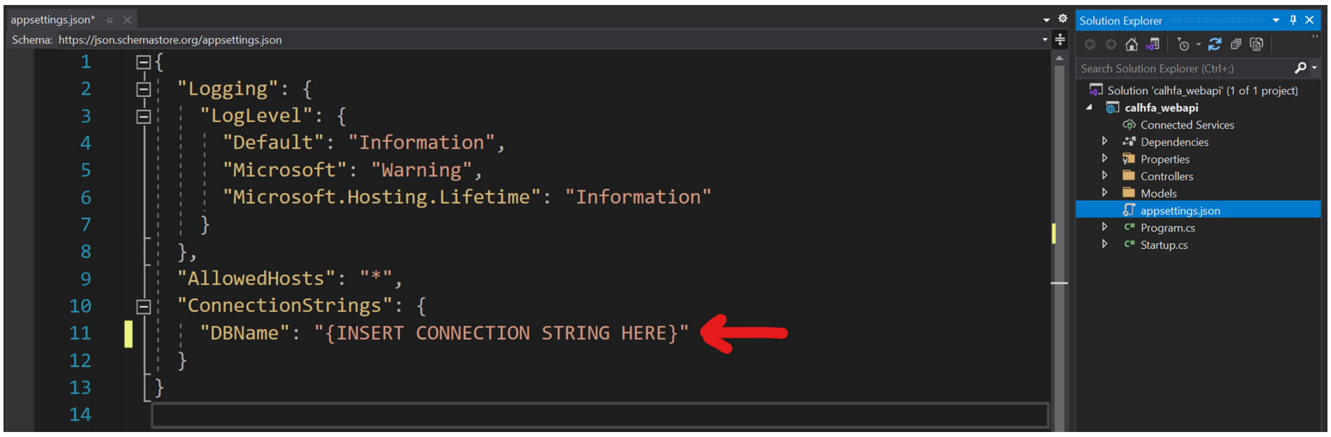
	LoanID	LoanCategoryID	StatusCode	StatusDate
1	1364552	1	410	2021-07-16 15:36:20.083
2	1364654	1	410	2021-08-13 08:53:56.080
3	1364665	1	410	2021-08-11 13:30:03.800
4	1364688	1	410	2021-09-22 12:23:33.120
5	1364712	1	410	2021-09-30 15:58:25.827

Note: This differs slightly from what the API sees. The API uses the same base query, but it only loads the dates field to be returned in the final table since it does not need any other information.

2. Tutorials

2.1 Run and debug the program in the development environment

1. Set the connection string in the appsettings.json file.

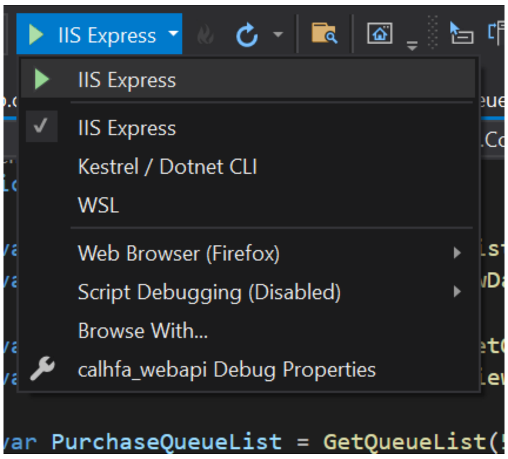


appsettings.json*

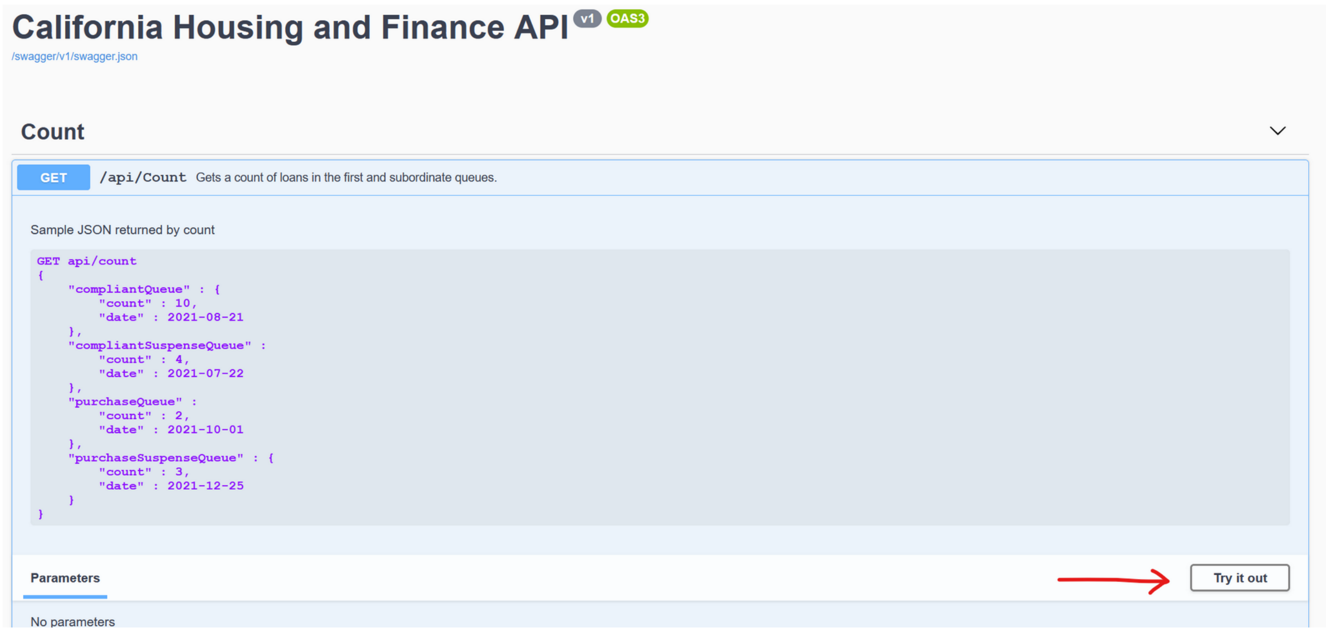
Schema: https://json.schemastore.org/appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DBName": "{INSERT CONNECTION STRING HERE}"
  }
}
```

2. You can run the program in the debugger using IIS Express or Kestrel / Dotnet CLI. The first time it is run, you may have to accept/install a TLS certificate, so that you don't run into TLS errors.


IIS Express
IIS Express
IIS Express
Kestrel / Dotnet CLI
WSL
Web Browser (Firefox)
Script Debugging (Disabled)
Browse With...
calhfa_webapi Debug Properties
var PurchaseQueueList = GetQueueList(

3. Your browser should automatically open up to the Swagger UI homepage. Click the "Try it out" button, and then the "Execute" button.



California Housing and Finance API v1 OAS3

/swagger/v1/swagger.json

Count

GET /api/Count Gets a count of loans in the first and subordinate queues.

Sample JSON returned by count

```
GET api/count
{
    "compliantQueue" : {
        "count" : 10,
        "date" : 2021-08-21
    },
    "compliantSuspenseQueue" :
        "count" : 4,
        "date" : 2021-07-22
    },
    "purchaseQueue" :
        "count" : 2,
        "date" : 2021-10-01
    },
    "purchaseSuspenseQueue" : {
        "count" : 3,
        "date" : 2021-12-25
    }
}
```

Parameters

Try it out

No parameters

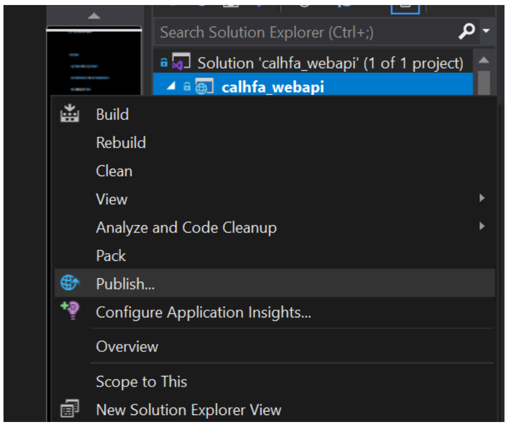
2.2 Stop homepage from redirecting to swagger

1. In the startup.cs class in the solution explorer, remove or comment out these lines

```
// Redirects home page to Swagger
var option = new RewriteOptions();        // Remove this line to stop redirecting the home page
option.AddRedirect("^$", "swagger");     // Remove this line to stop redirecting the home page
app.UseRewriter(option);                  // Remove this line to stop redirecting the home page
```

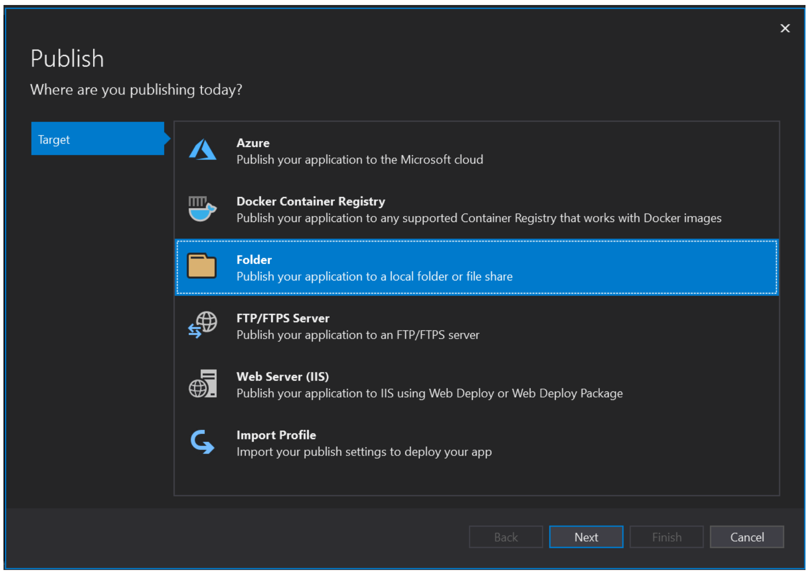
2.3 Publishing for Deployment

1. In the VS solution explorer, right click the parent folder of the project and select publish



Search Solution Explorer (Ctrl+;)
Solution 'calhfa_webapi' (1 of 1 project)
calhfa_webapi
Build
Rebuild
Clean
View
Analyze and Code Cleanup
Pack
Publish...
Configure Application Insights...
Overview
Scope to This
New Solution Explorer View

2. Select "folder" as the publishing target


Publish

Where are you publishing today?

Azure
Publish your application to the Microsoft cloud

Docker Container Registry
Publish your application to any supported Container Registry that works with Docker images

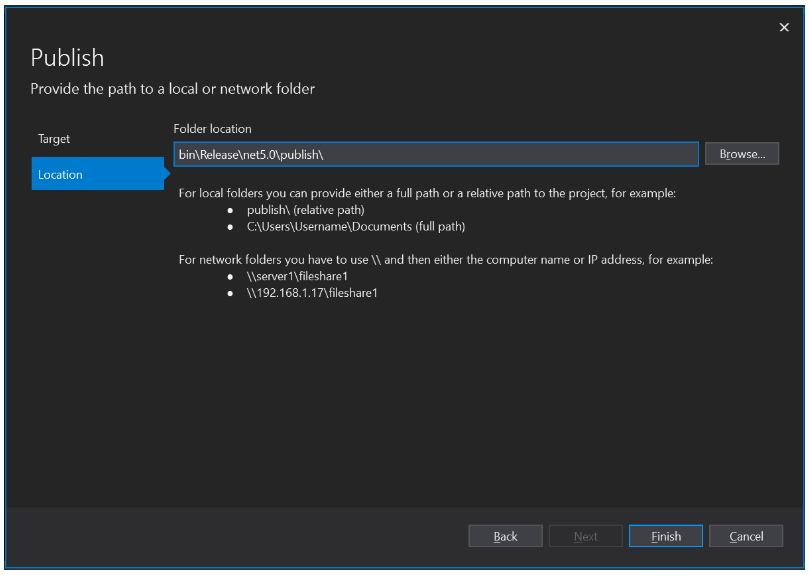
Folder
Publish your application to a local folder or file share

FTP/FTPS Server
Publish your application to an FTP/FTPS server

Web Server (IIS)
Publish your application to IIS using Web Deploy or Web Deploy Package

Import Profile
Import your publish settings to deploy your app

3. Confirm the location and select "Finish." The default path can be found in the project itself.



Publish

Provide the path to a local or network folder

Folder location

bin\Release\net5.0\publish\

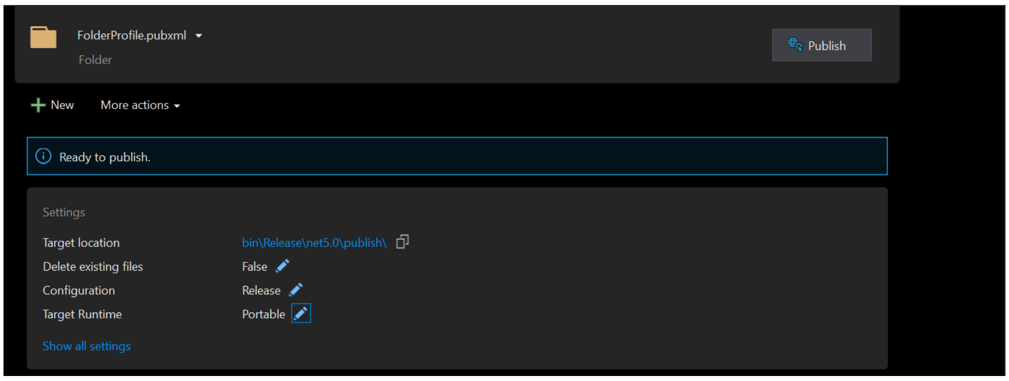
For local folders you can provide either a full path or a relative path to the project, for example:

- publish\ (relative path)
- C:\Users\Username\Documents (full path)

For network folders you have to use \\ and then either the computer name or IP address, for example:

- \\server1\fileshare1
- \\192.168.1.17\fileshare1

4. Set the target runtime to "win-x64" and ensure the deployment mode is Framework dependent. Then click "save."



FolderProfile.pubxml

Folder

Ready to publish.

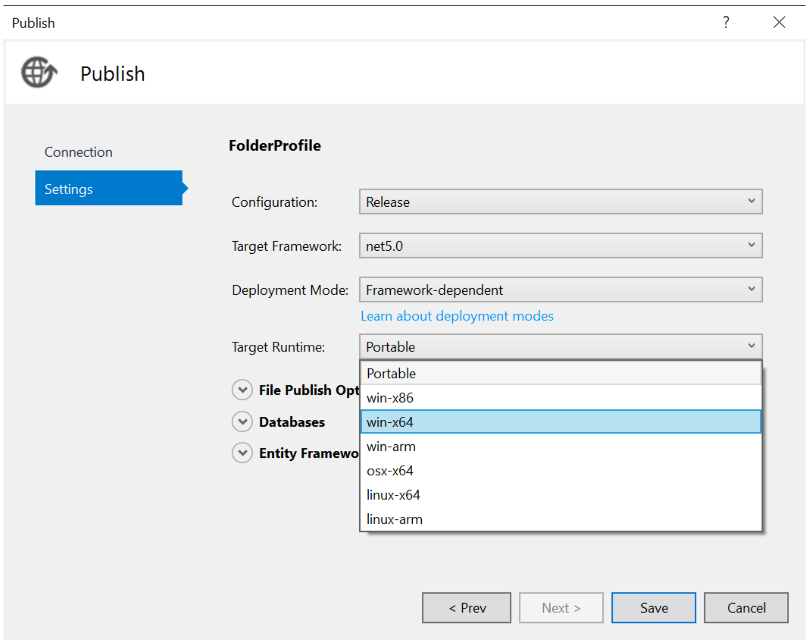
Settings

Target location bin\Release\net5.0\publish\

Delete existing files False

Configuration Release

Target Runtime Portable



Publish

Publish

FolderProfile

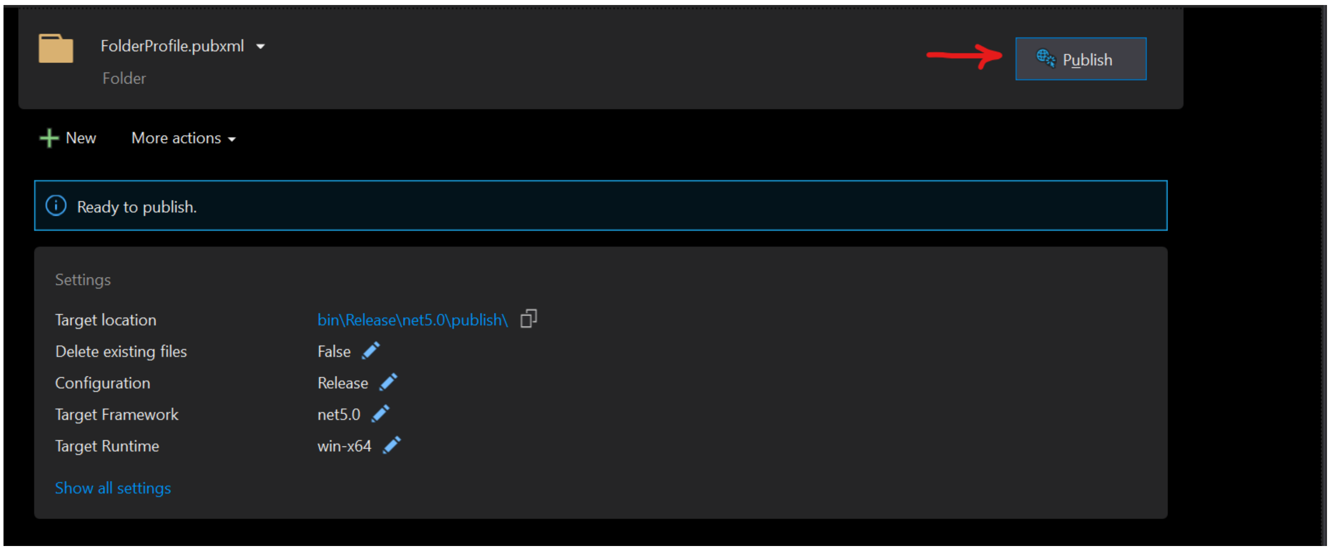
Configuration: Release

Target Framework: net5.0

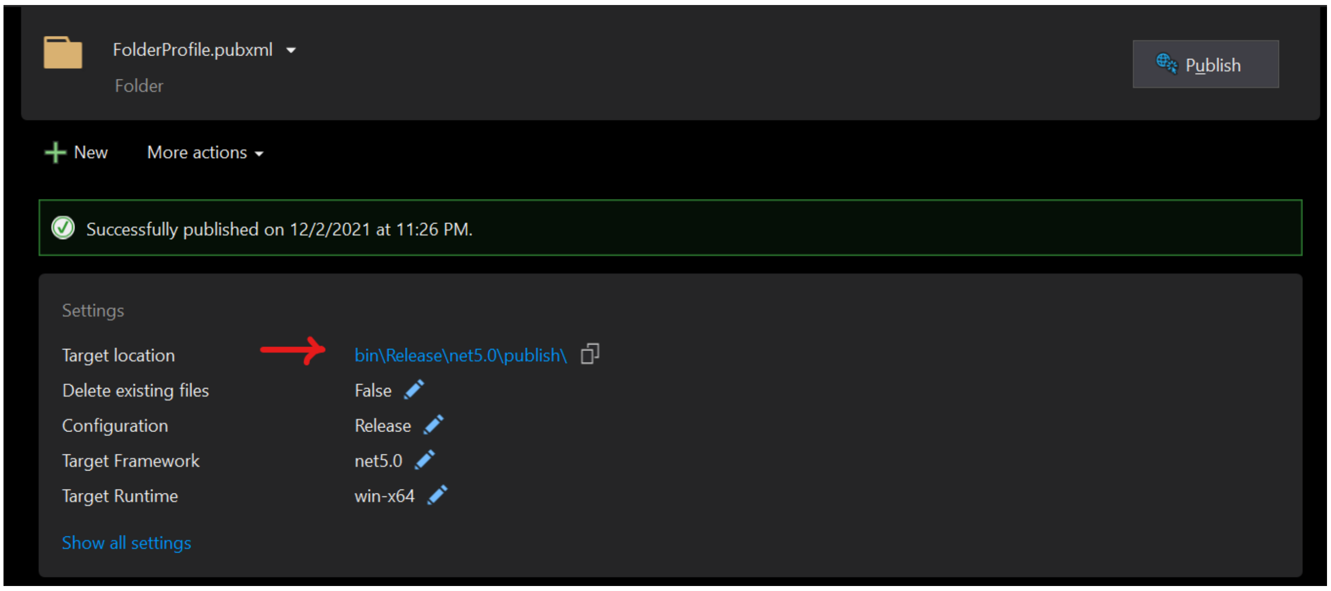
Deployment Mode: Framework-dependent

Target Runtime: Portable

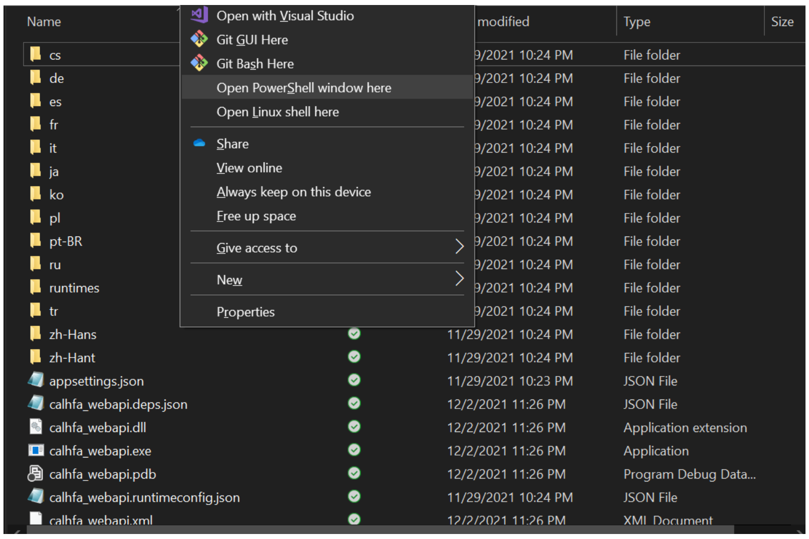
5. Click "Publish." The publishing procedure will begin. When it is complete, the published folder can be found in the target location you set.


FolderProfile.pubxml
Folder
Publish
New
More actions
Ready to publish.
Settings
Target location bin\Release\net5.0\publish\
Delete existing files False
Configuration Release
Target Framework net5.0
Target Runtime win-x64
Show all settings

6. To test the build folder before hosting, navigate to the published folder location. You can do this quickly by clicking the target location

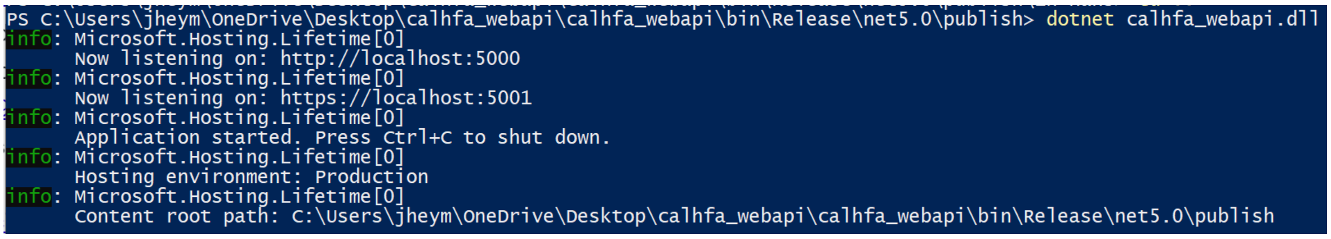

FolderProfile.pubxml
Folder
Publish
New
More actions
Successfully published on 12/2/2021 at 11:26 PM.
Settings
Target location bin\Release\net5.0\publish\
Delete existing files False
Configuration Release
Target Framework net5.0
Target Runtime win-x64
Show all settings

7. In the folder, make sure the connection string is set in appsettings.json, then shift+rightclick to open a powershell in the current directory.



Name	modified	Type	Size
cs	9/2021 10:24 PM	File folder	
de	9/2021 10:24 PM	File folder	
es	9/2021 10:24 PM	File folder	
fr	9/2021 10:24 PM	File folder	
it	9/2021 10:24 PM	File folder	
ja	9/2021 10:24 PM	File folder	
ko	9/2021 10:24 PM	File folder	
pl	9/2021 10:24 PM	File folder	
pt-BR	9/2021 10:24 PM	File folder	
ru	9/2021 10:24 PM	File folder	
runtimes	9/2021 10:24 PM	File folder	
tr	9/2021 10:24 PM	File folder	
zh-Hans	11/29/2021 10:24 PM	File folder	
zh-Hant	11/29/2021 10:24 PM	File folder	
appsettings.json	11/29/2021 10:23 PM	JSON File	
calhfa_webapi.deps.json	12/2/2021 11:26 PM	JSON File	
calhfa_webapi.dll	12/2/2021 11:26 PM	Application extension	
calhfa_webapi.exe	12/2/2021 11:26 PM	Application	
calhfa_webapi.pdb	12/2/2021 11:26 PM	Program Debug Data...	
calhfa_webapi.runtimeconfig.json	11/29/2021 10:24 PM	JSON File	
calhfa_webapi.xml	12/2/2021 11:26 PM	XML Document	

8. In the powershell terminal, type `dotnet calhfa_webapi.dll`. You should see something similar to the image below.



```
PS C:\Users\jheym\OneDrive\Desktop\calhfa_webapi\calhfa_webapi\bin\Release\net5.0\publish> dotnet calhfa_webapi.dll
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\jheym\OneDrive\Desktop\calhfa_webapi\calhfa_webapi\bin\Release\net5.0\publish
```

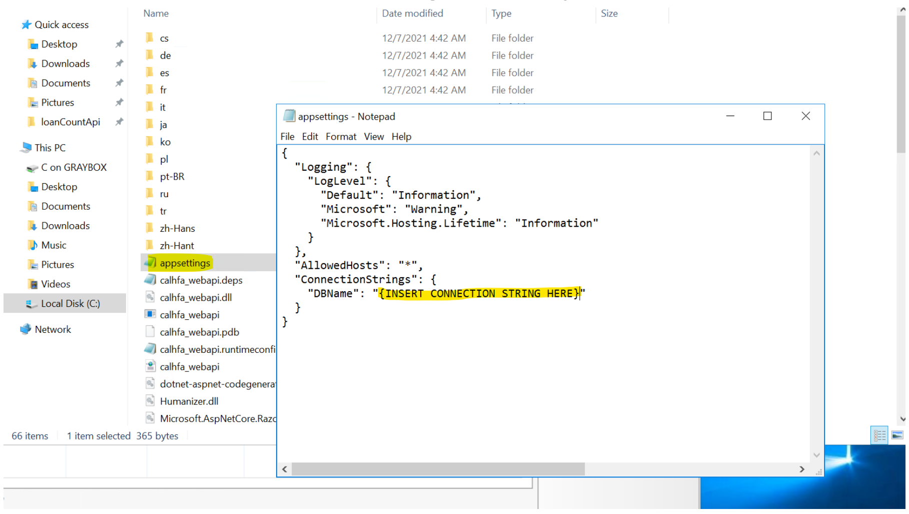
9. Navigate to https://localhost:5001 and confirm that the API is working as intended. If something goes wrong, check the powershell terminal for any errors.

That's all for creating the publish folder. You will need to move the publish folder to your production environment for hosting. Refer to the IIS instructions for hosting.

2.4 Deploying to IIS

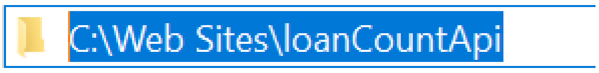
This tutorial is a step-by-step guide to setting up our API on IIS. We assume you will have some knowledge of how to do this, but this is what we did.

1. Move the build folder to the desired directory of your webroot
   - *Note: If the application is not under the inetpub directory, ensure that the IIS user (IIS_USRS) account has read and write access to the path
2. Ensure that the connection string is set to your SQL Server Database.



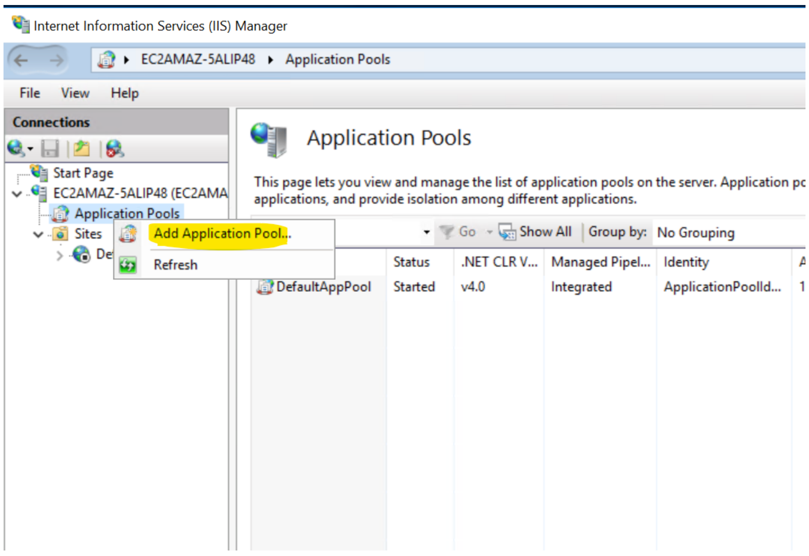
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DBName": "{INSERT CONNECTION STRING HERE}"
  }
}
```

3. Copy the path of the application directory to your clipboard



C:\Web Sites\loanCountApi

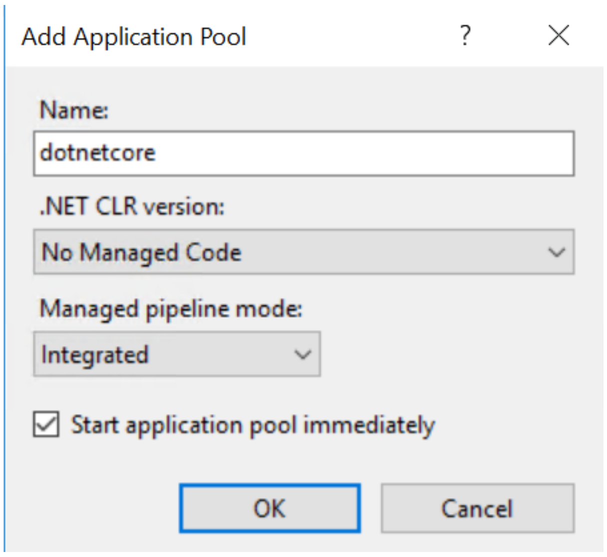
4. In IIS, add an application pool



This page lets you view and manage the list of application pools on the server. Application pc
applications, and provide isolation among different applications.

	Status	.NET CLR V...	Managed Pipel...	Identity	A
DefaultAppPool	Started	v4.0	Integrated	ApplicationPoolId...	1

5. Name the application pool and select "no managed code" in the .NET CLR dropdown. Select "OK".



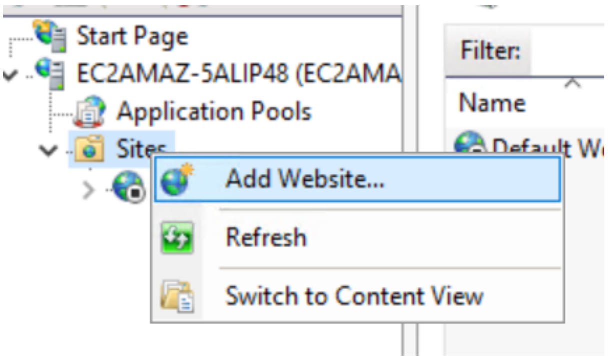
Name: dotnetcore

.NET CLR version: No Managed Code

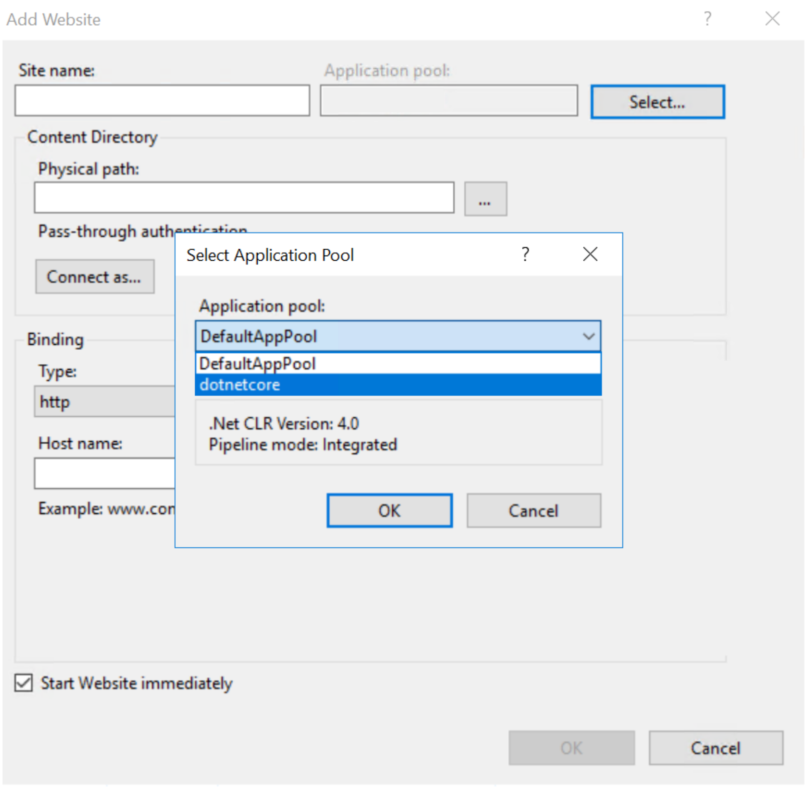
Managed pipeline mode: Integrated

☑ Start application pool immediately

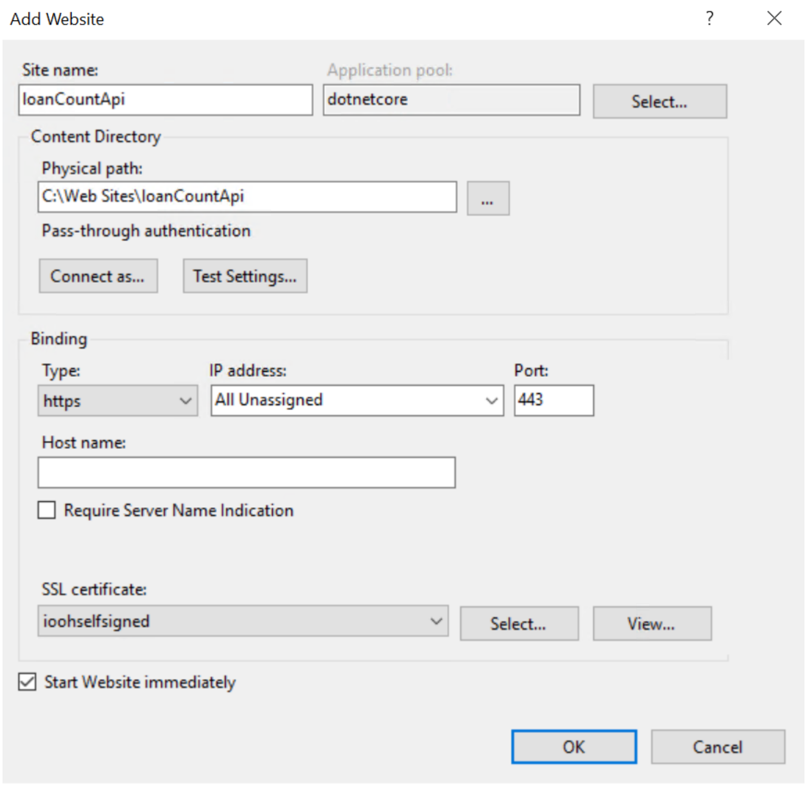
6. Add a new website by right clicking the "Sites" folder



7. In the Add website window, click the "Select..." button and change the application pool to the one you created



8. *Note: In this step you will have to select your SSL certificate, so make sure you add it to IIS if you haven't already. For information on how to add an SSL certificate to IIS, refer to this article.*
Name your website and select the physical path where you placed the application folder in step 1. If it is no longer in your clipboard, you can specify the path via the menu button to the right of the field. Change the binding type to https and specify the port you want to run on. Then select your SSL certificate.



Add Website

Site name: loanCountApi
Application pool: dotnetcore

Content Directory
Physical path: C:\Web Sites\loanCountApi

Pass-through authentication

Binding
Type: https
IP address: All Unassigned
Port: 443
Host name:
Require Server Name Indication

SSL certificate: ioohselfsigned

Start Website immediately

9. That's it! The website should now be running on https. If you try to connect on http you will get a hanging page. You may add an http binding to prevent this, but you should set up http to https redirects as the API will not function on http. For more information on how to redirect all http traffic to https, refer to this stackexchange answer.

In Over
Our Heads

Documentation was written by Justin Heyman and David Enzler
heymanj@protonmail.com
davidenzler@hotmail.com