**FM Systems**

**Software Engineer Coding Exercise**

**Solution**

Contents

[Requirements 2](#_Toc76472118)

[FmApi 2](#_Toc76472119)

[FMUnitTest 2](#_Toc76472120)

[FMClient 2](#_Toc76472121)

[API Requirement 2](#_Toc76472122)

[What it was done and how 2](#_Toc76472123)

[Visual Testing 6](#_Toc76472124)

[Projects and how to Start it without publishing it 7](#_Toc76472125)

[Unit Test Requirement 8](#_Toc76472126)

[Client that Shows Selected City Weather 10](#_Toc76472127)

[Consume External API Requirement 13](#_Toc76472128)

# Requirements

The solution is composed of three projects:

FmApi: This is an ASP Net Core API that provides the list of Cities.

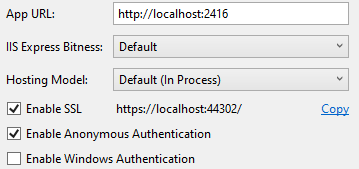
FMUnitTest: A unit test that checks if the API returns the expected cities count.

FMClient: Web site to gather weather information for a selected city.

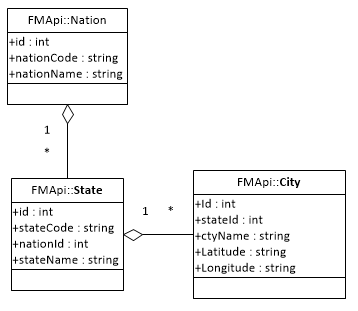
# API Requirement

## What it was done and how

This project is a .NET Core, this API returns a collection of the cities with their pertinent information detailed below. This task is accomplished with the project named.



This diagram shows the class relations in this project, the project could be done with a single class for the cities, but I wanted to show the potential for extensibility. At the moment of documenting the project, only the cities are used. With this structure in place, we can add cascading dropdown controls to the client, where the user selects the Nation first, and the site shows a list of States, the user selects a state, and the site filter out the list of cities belonging to the selected state.



The weather API uses the city’s coordinates to retrieve the weather data. Such data could be obtained dynamically by requesting it from another API; but since that is a kind of static information inserted only once, I decided to add such data to the city class and the corresponding collection of cities.

The following list of cities is hardcode in the application:

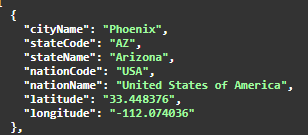
Phoenix, AZ 33.448376 -112.074036

Raleigh, NC 35.779591 -78.638176

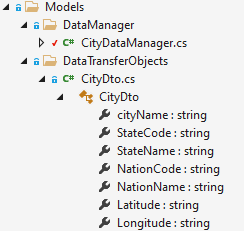
San Diego, CA 32.715736 -117.161087

Saint John, NB (Canada) 45.272812 -66.063026

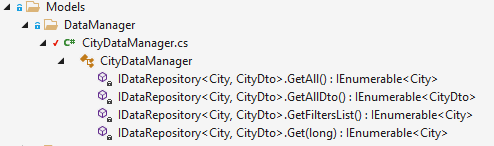
The resulting information has more data than the City class provides, as shown in this partial result illustration:



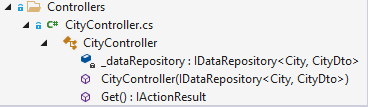
I used the Data Transfer Object Design Pattern to obtain such composite results where the corresponding nation and state are returned. The City class remains intact so that way it can be used for basic CRUD operations. In this case the class CityDto is used to create the collection of cities to be returned by the API, with the nation and stated information decoded.



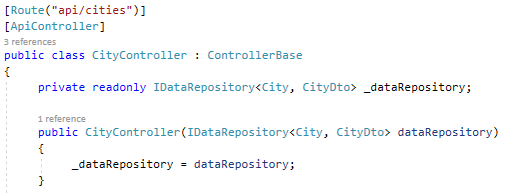
The interface IDataRepository is inherited by any data manager class used to retrieve data from any data source. A concrete class that implements this interface is created, and the class **CitDataManager** performs such a task.



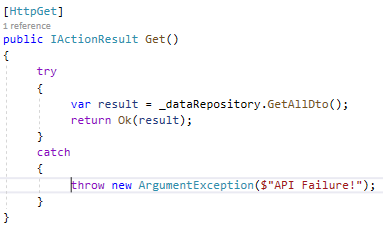
The **CitDataManager** class creates the generic list for the Nations, States, and Cities that perform the corresponding joining and return a composite fully decoded list of CityDto objects to the requesting controller. The **CityController** has only the method needed to return the collection of cities.



The **IDataReposity** interface is injected in the controller:



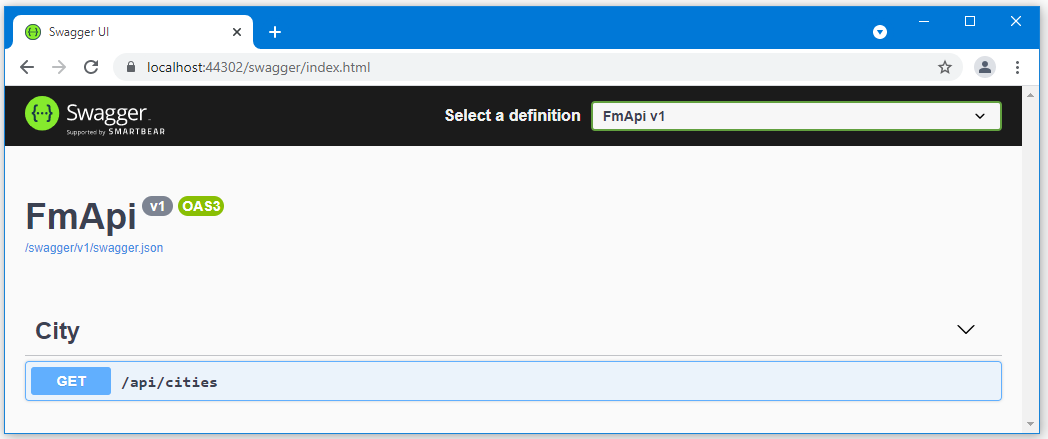
To be used as a gate to access the data via the concreate class named CitDataManager:



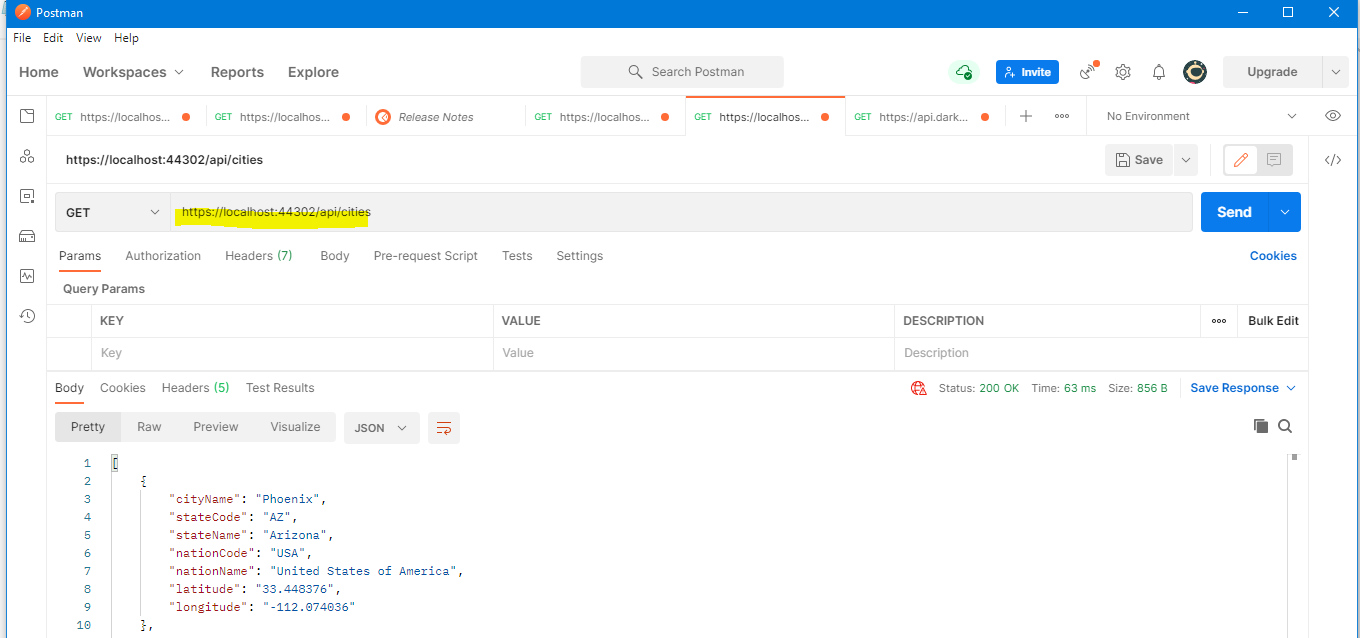
## Visual Testing

Besides the unit test that technically verifies the API, the following test can be performed:

Swagger Ui: this is integrated into the API, and it pops up when running it in Debug mode:

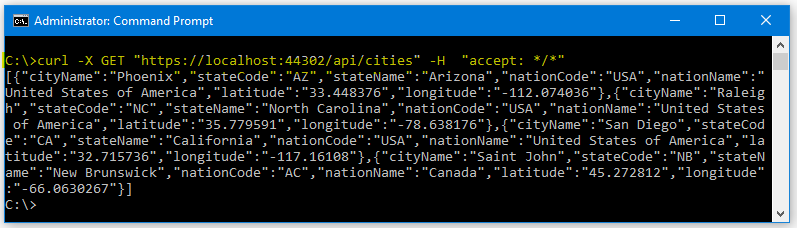


From Postman: send a GET request to https://localhost:44302/api/cities



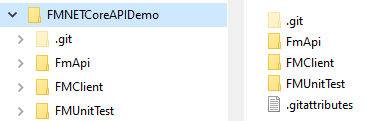
From the command prompt, via this curl command:

curl -X GET "https://localhost:44302/api/cities" -H "accept: \*/\*"

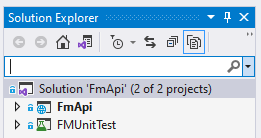


## Projects and how to Start it without publishing it

The entire project is coded using the free version of Visual Studio 2019, and each one is stored in its respective folder under FMNetCoreAPIDemo.



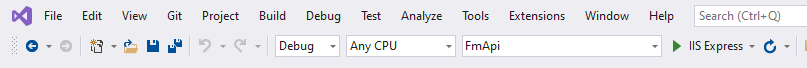
The FmApi and the FMUnitTest projects are under the same Solution called FmApi



Graphical user interface, application, table

Description automatically generated

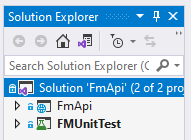
Open the FmApi solution, compile it and run it once you make sure that it is set up as the Startup project, and run it in the local IIS Express



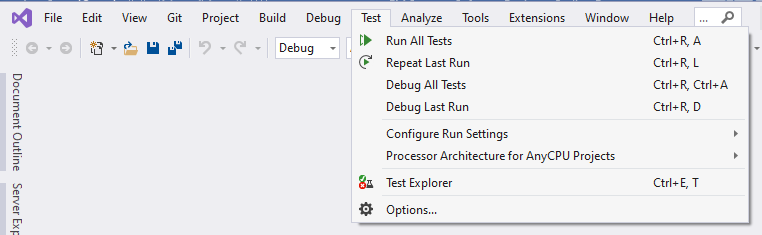
# Unit Test Requirement

*“Create a .NET Core unit test project to test this API action. The project is named FMUnitTest.”*

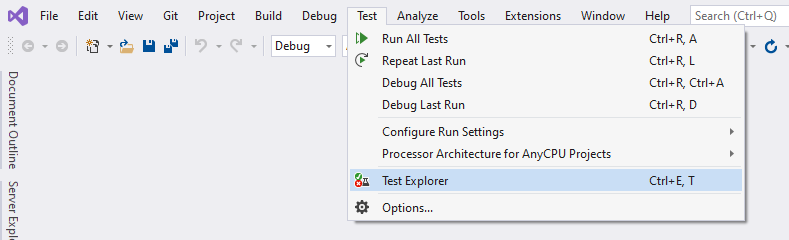
To run this project, right-click on it in the Solution Explorer and make Startup project:



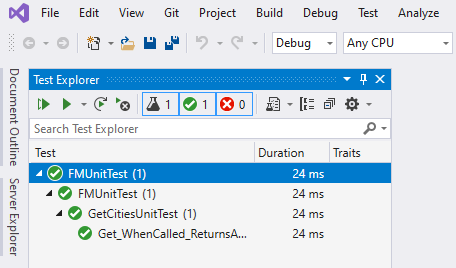
From the main menu, select Run All Tests



If the Test Explorer does not appear; from the main menu, select Test, Test Explorer



The Test Explorer will pop-up on the left side or in the middle of the screen, showing the test and its results:



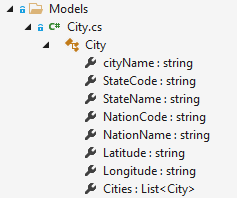
Set the FmApi project back the Startup Project and run it to continue with the client project.

# Client that Shows Selected City Weather

*“Using any web technologies you'd like, display the list of cities in a drop-down list. The list should be populated via a request to the city collection API that you created. The UI design should be simple and take minimal time to develop.”*

This task is accomplished with the FMClient project, also created with Visual Studio 2019, using the MVC Architecture.

In the Models, there is the class City which is a copy of the CityDto from the FmApi project with the addition of a city collection property.

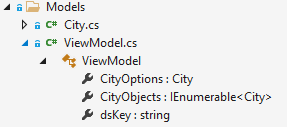


This class with the Cities collection property is used to deliver the list of cities to the Index razor page to populate the cities dropdown controller.

Also, the controller extracts the port for the FMApi and the key to access the Dark Sky API from the appsettings.json file.

We need to deliver more than one model to a single view; two simple models for the cities dropdown list and the Dark Sky API key, and another more complex containing the coordinates data to collect the weather data from the third-party API.

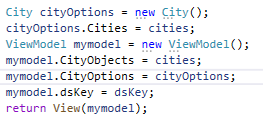
For this purpose, we have the ViewModel class.



The Index view is expecting the custom ViewModel container:

**@model ViewModel**

and the controller composes the data collections accordingly and delivers it to the view:



The view model class acts a placeholder to deliver multiple models to the view.

**appsettings.json file:**

Not that the key value for the Dark Skye API is enclosed in single quotes. If this is not done, the application enters into an error.

With the key

{

"FMClientConfig": {

"FmApiPort": "44302",

"DSKey": "**'**567f968246cd8cc7463??????????????**'**"

}

}

If we do not want to provide the Dark Sky key in the configuration file, then the word manual enclosed in single quotes is used.

Without the key:

{

"FMClientConfig": {

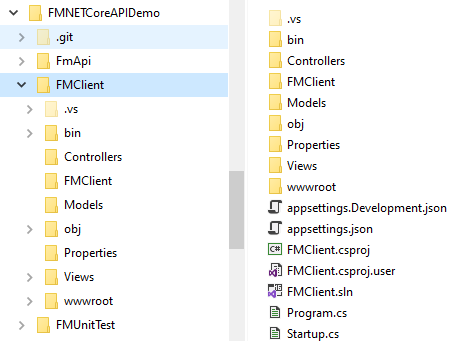
"FmApiPort": "44302",

"DSKey": "**'**manual**'**"

}

}

This illustration shows the file structure for the FMClient solution, double click it to open it with Visual Studio.



# Consume External API Requirement

*“Selecting a city from the drop-down needs to trigger a call out to the*[*DarkSky API*](https://darksky.net/dev)*and retrieve the weather for the selected city from July 4, 2018, at exactly noon local time.*

*On the screen display the noon current summary description*

*Example:*

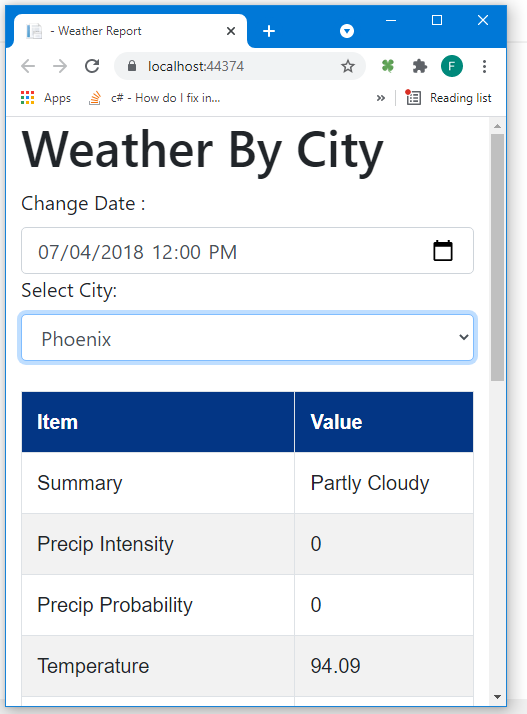
*Mostly Sunny,*

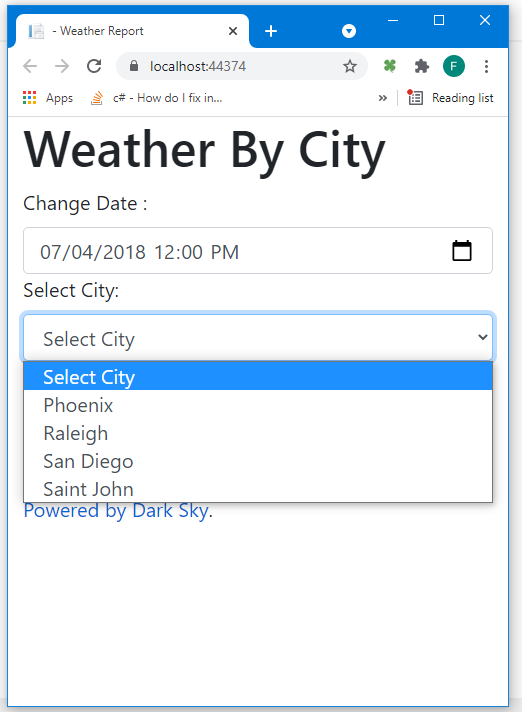
*Temperature (example: 88.81), and*

*UV index (example: 5).”*

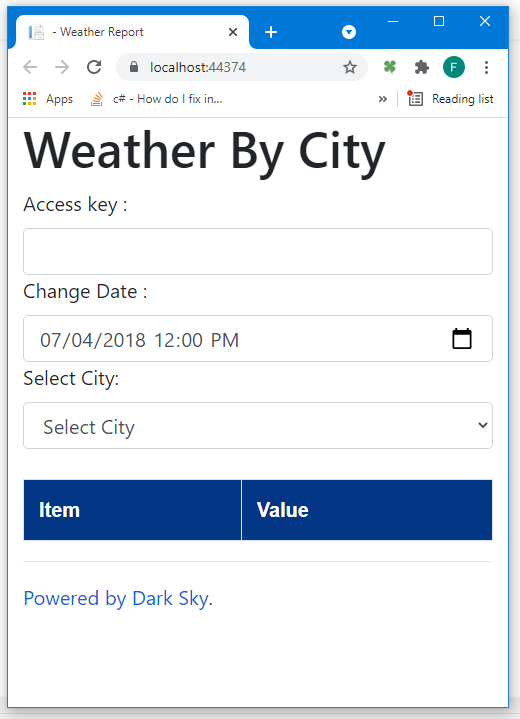
Herein a list of screenshots illustrating the site functionality; if the Dark Sky key is stored in the application settings file, it will not require any manual input. The default date in the date picker controller is set the required date, so the only thing to be done is to select the city, and the data will be fetched and shown:

**With access key stored**





**With access key manual input**



Graphical user interface, application

Description automatically generated