MONASH University

FIT5032 - Internet Applications Development

Week 10 - Modern javaScript Development

Last Updated: 6th August 2018
Author: Jian Liew

Housekeeping

Before we begin, it is highly recommended for you to use your own personal computer for this subject. If you are planning to use the Monash computers, it is highly recommended to save all your work properly. Towards, the end of the semester, there will be a portfolio submission where you need to showcase all the work you have done so far. To prevent difficulties during this portfolio submission, it is suggested that you keep all your work neat and organised based on a **week by week structure.**

Tutorial Structure

The tutorials in this unit are designed to be of a **self-paced and self-taught structure.** So, the aim of your tutor is to aid you in your learning experience. He or she will not go through all the materials that are covered in the labs and will **not do the exercise one by one as a class**. If help is needed, you can either post on the Moodle forums or you can email your tutor asking for clarification. This is slightly different in comparison to other units, where your tutor will lead the discussions. **Please take note of this, you will only be provided help if you make it known that you need help.**

The contents of this lab has been condensed significantly as it is impossible to even cover the basics of what is needed for this topic. In a sense, this week should be taught as a multiple subject itself which ranges from Software Architecture to Enterprise Development and multiple other aspects.

Objectives

Estimated Time To Complete - 2+ hours

Upon the completion of this tutorial, you will gain a basic understanding of

- How to create and consume a RESTful Web API
- Understand a simple AngularJS application

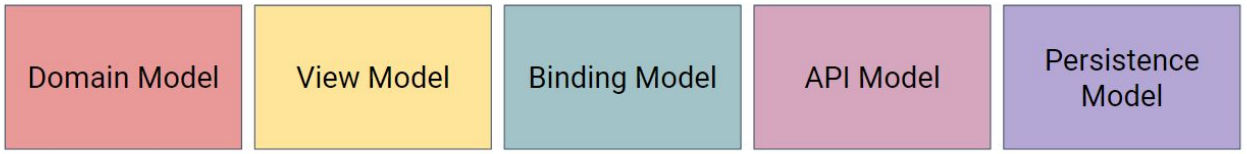
DoubtFire Submission

- A document detailing if the automatically generated web API from Visual Studio follows the best practices
- OR
- A Document showing the screen shots of your web api application running (in PDF document)

What is a WEB API?

In layman's term, the WEB API is used to exchange information between parties. In this lab, we will first create RESTful endpoints to be used for application. Our JavaScript application will then consume these resources.

So, we would have covered all the different models in our tutorials.



Visual Studio MVC has features that simplifies this process by having it automatically done for us.

Persistence Model on Week 4 (Using the Entity Framework)

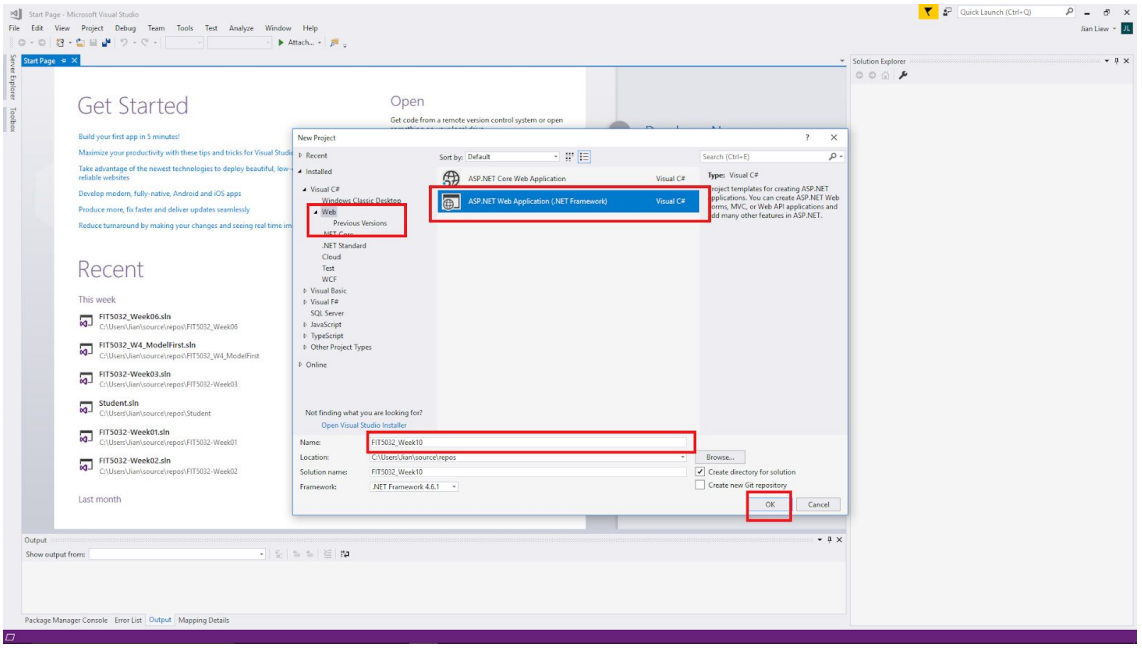
View Model & Binding Model covered on Week 6 (Validation)

Domain Model (This is normally introduced as either "DTO" or "Services" think of this as an additional layer or namespace)

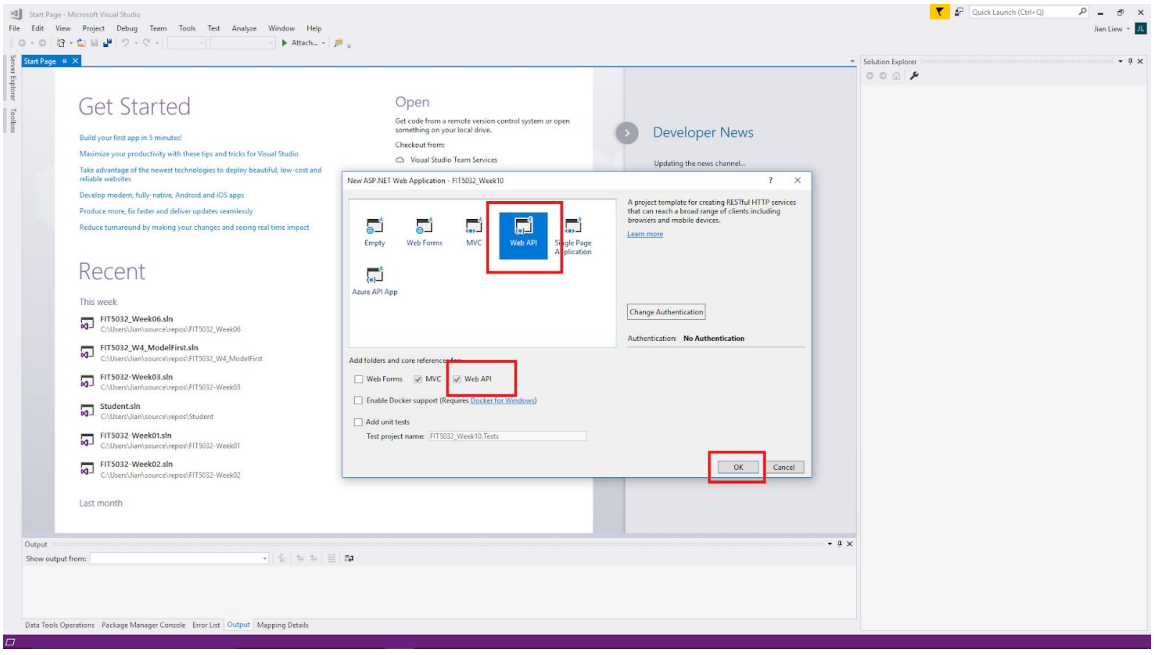
Creating the Web API

The way the project is created for this week would be slightly different in comparison to previous weeks. Please proceed with caution. (You can actually introduce the WEB API to the default project, but it would require a slight configuration change)

Step 1

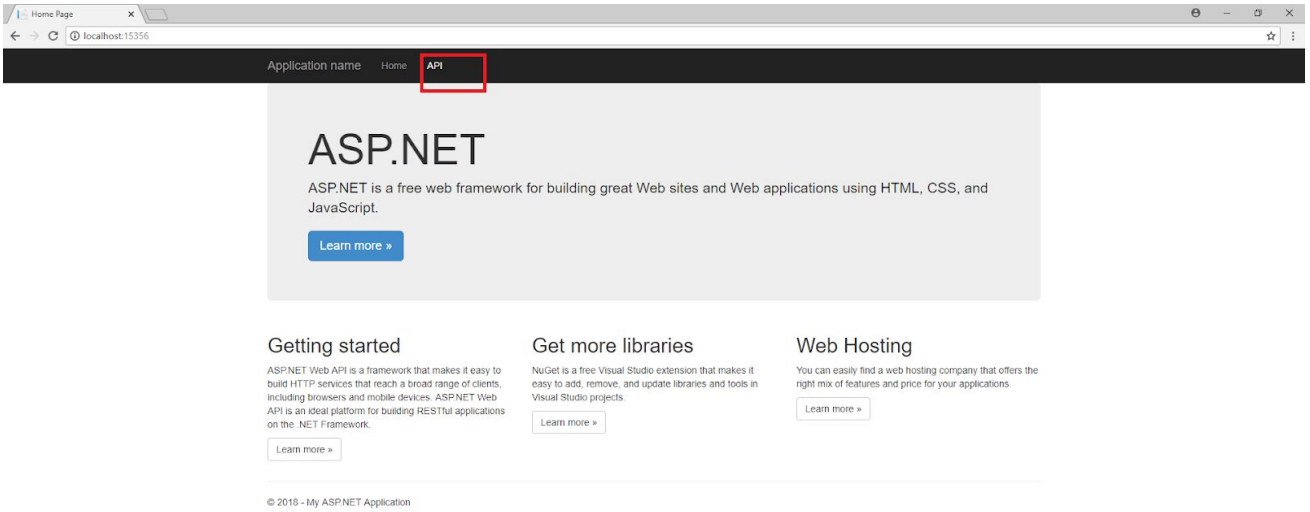


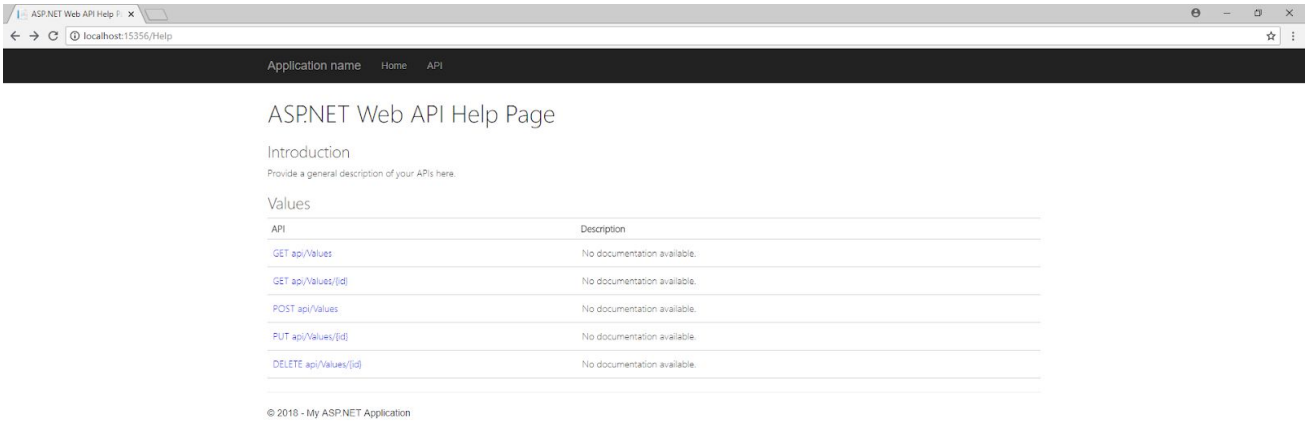
Step 2



Step 3

The project can be ran. Take a look at the API link.

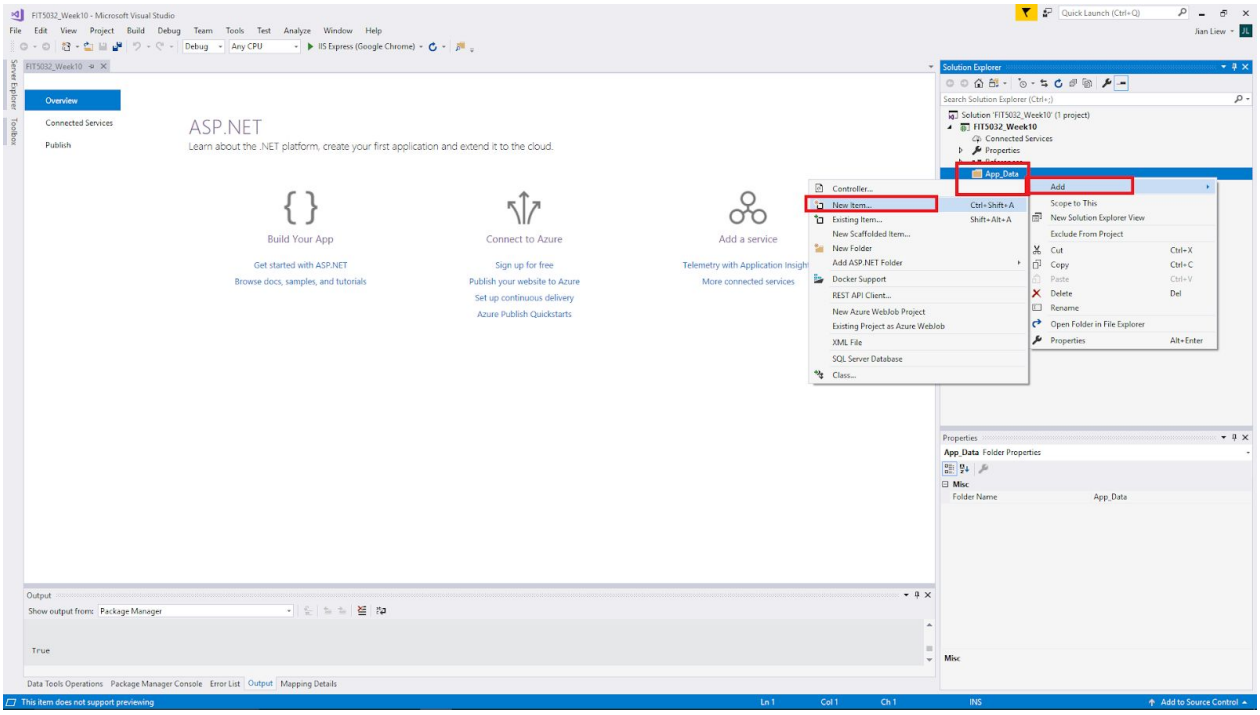


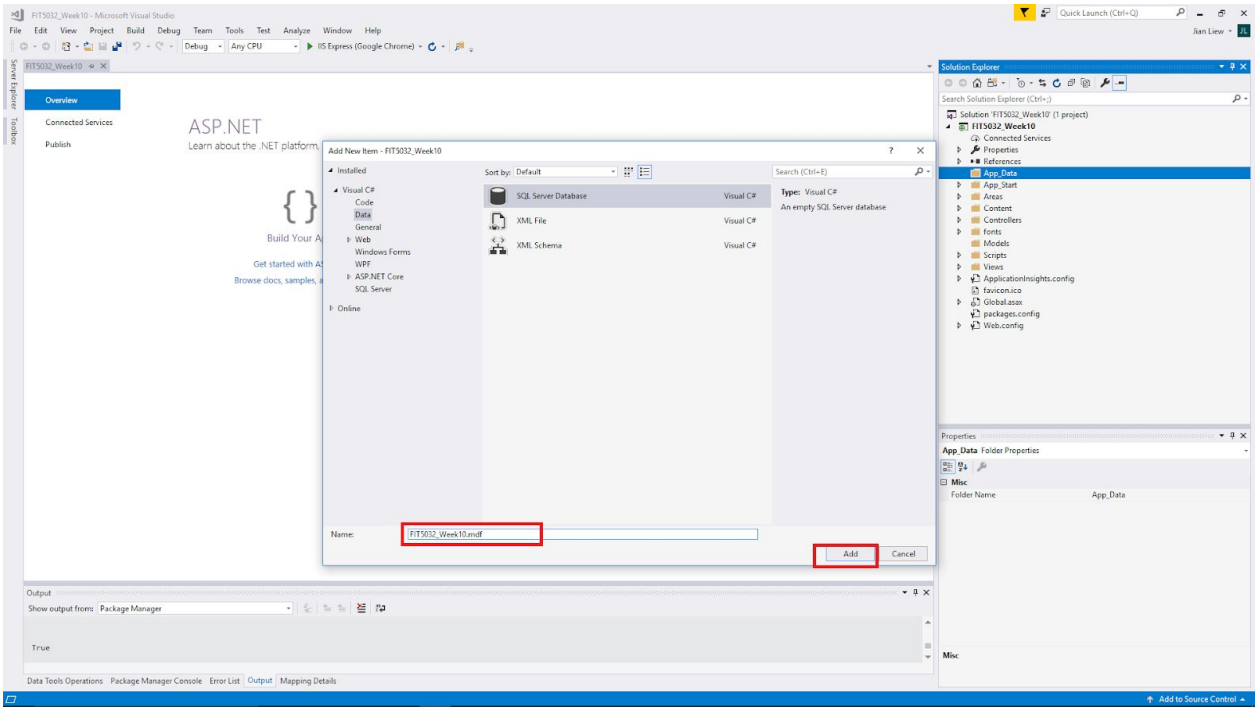


By default, Visual Studio gives some basic documentation. What we will be doing shortly is creating our own Web API so that it can be consumed by 3rd parties or ourselves.

Step 4

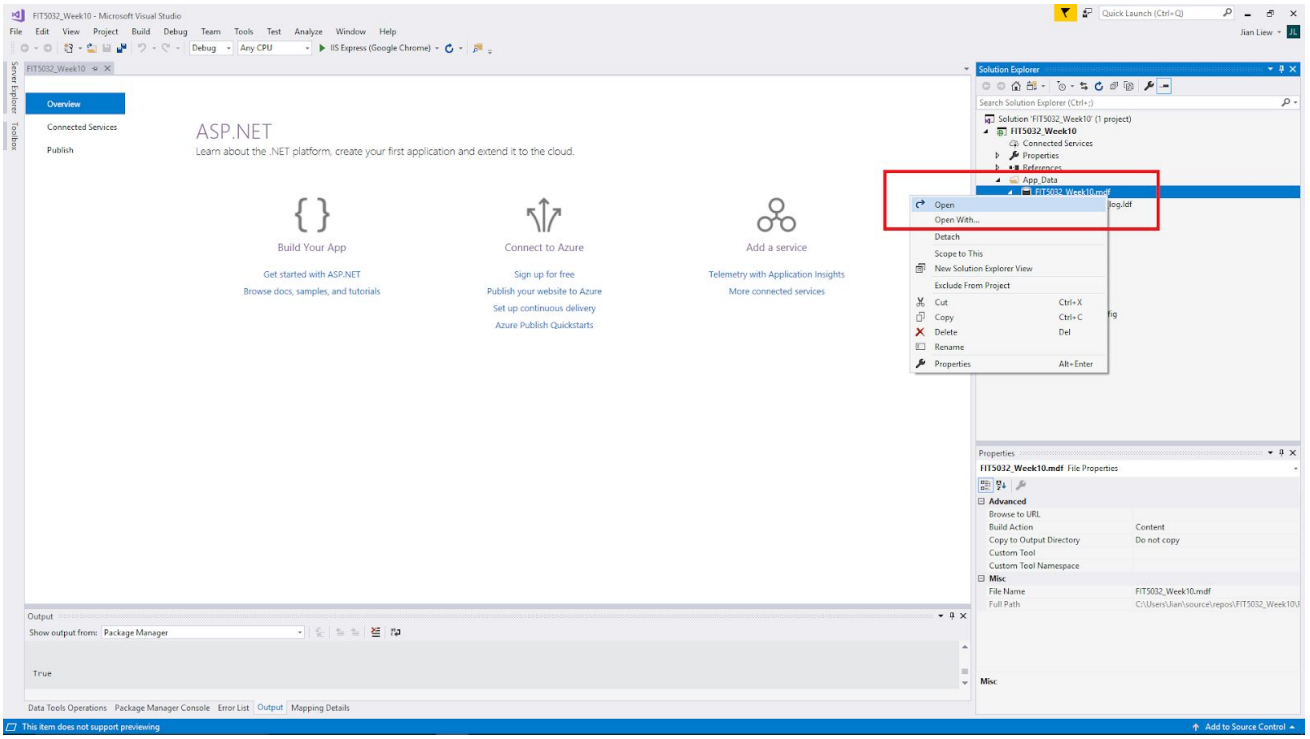
I will now use a Database first approach to create my models.

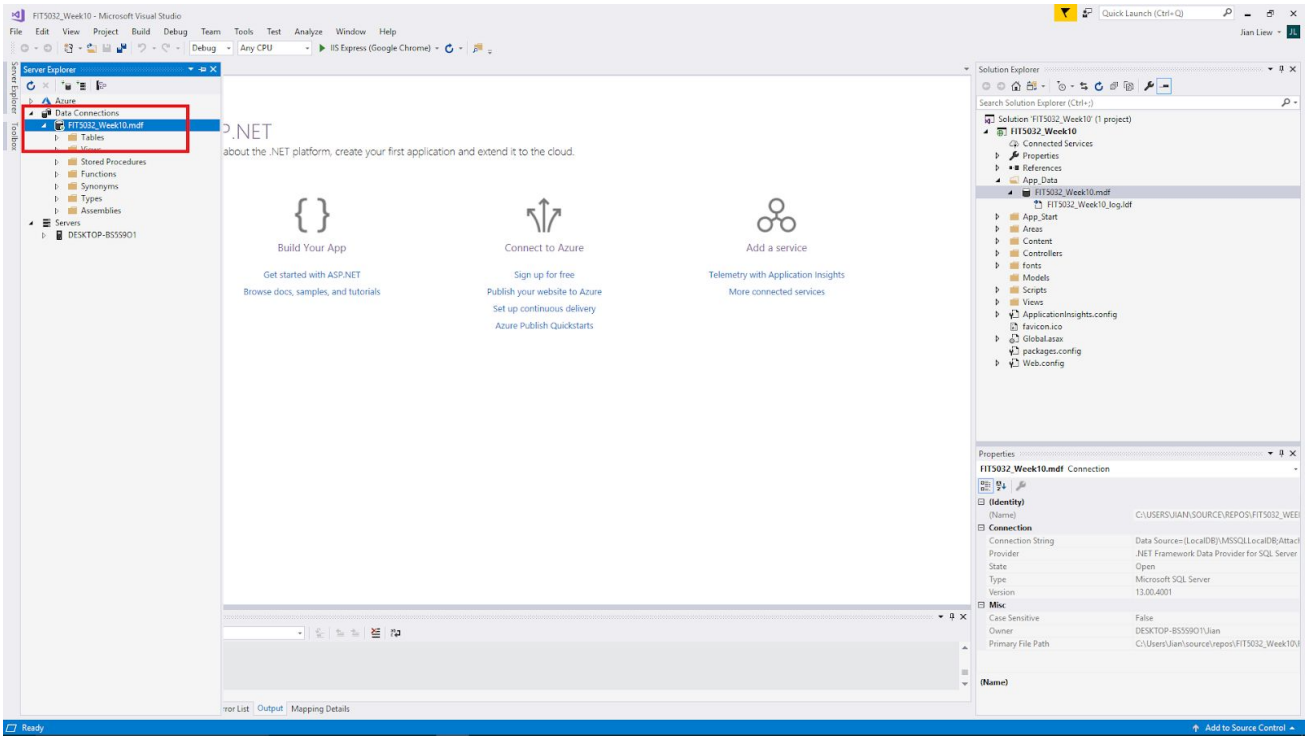




Step 5

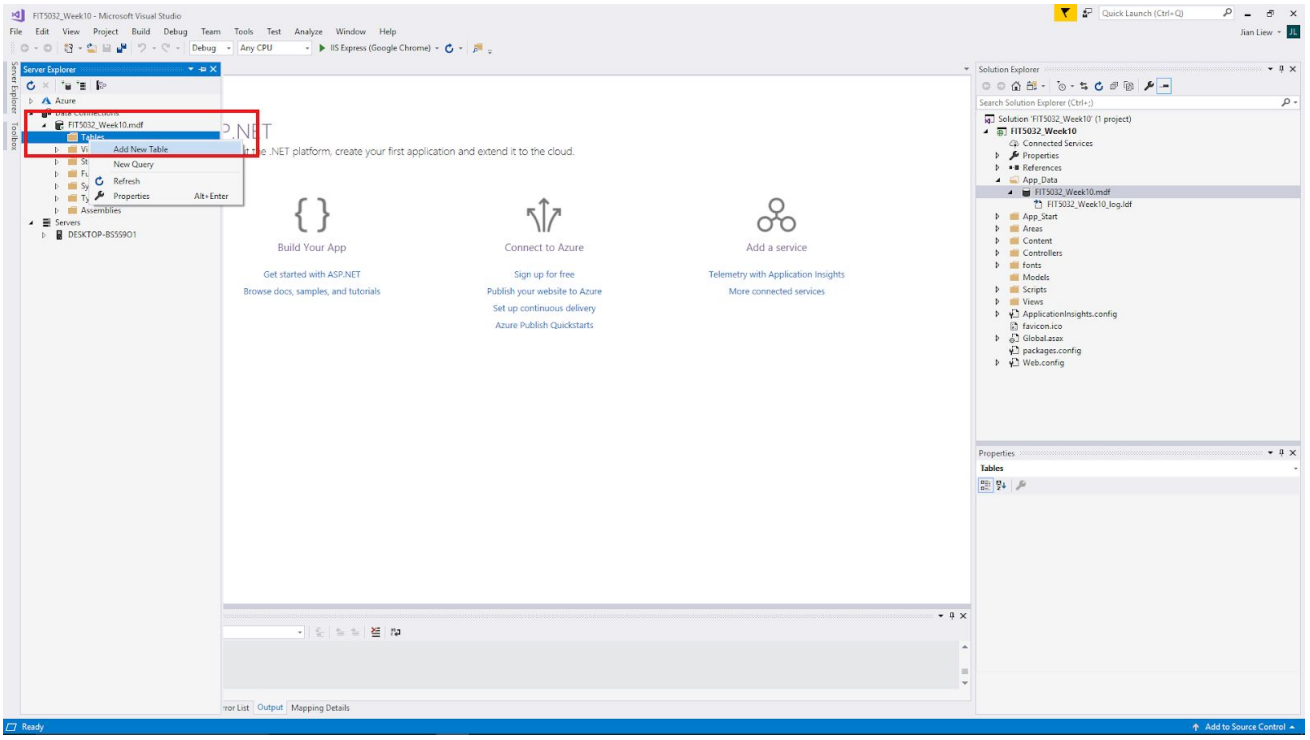
Open the .mdf file.



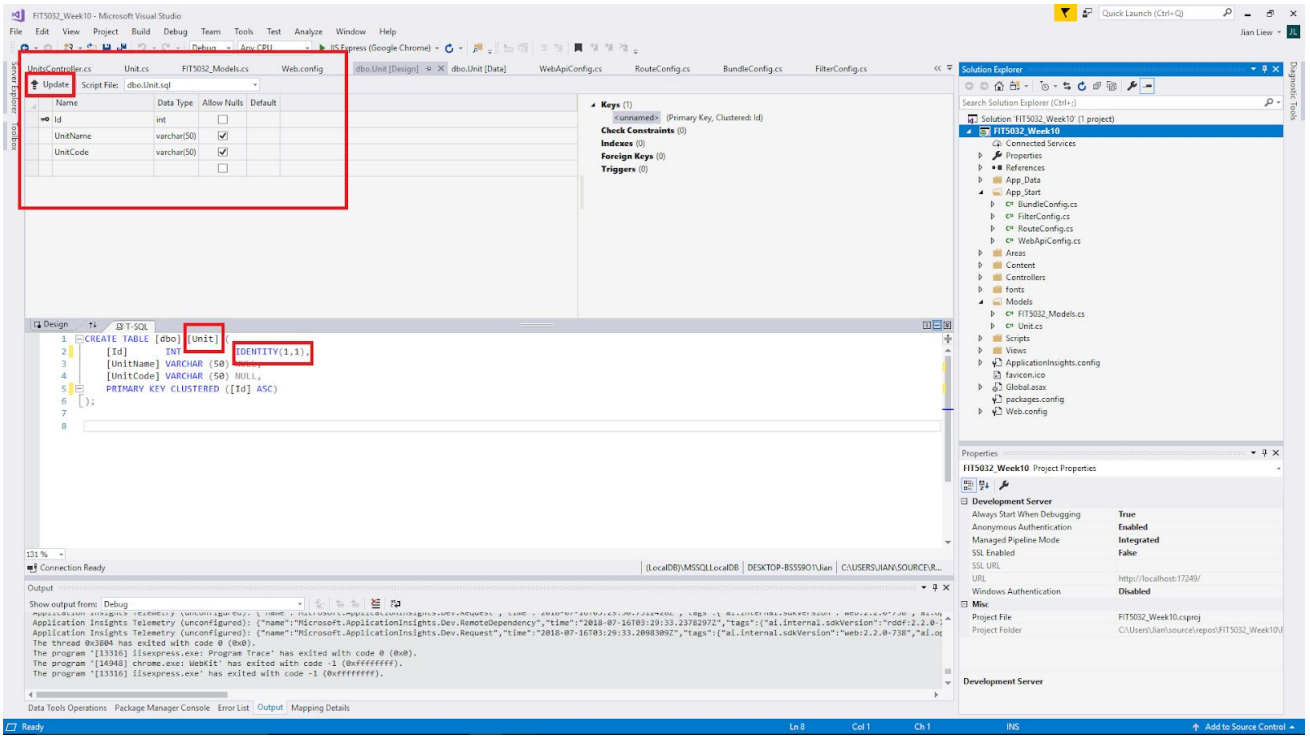


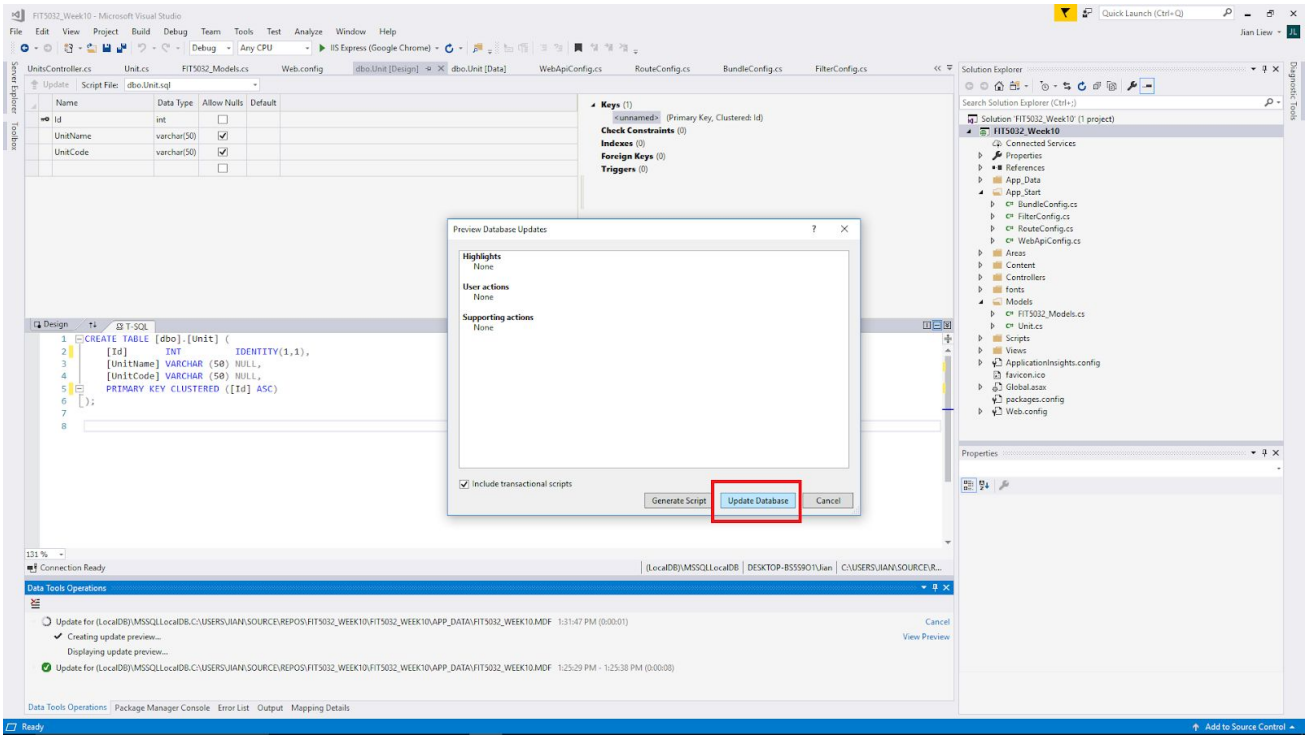
You should see the newly created .mdf file under the server explorer.

Step 6

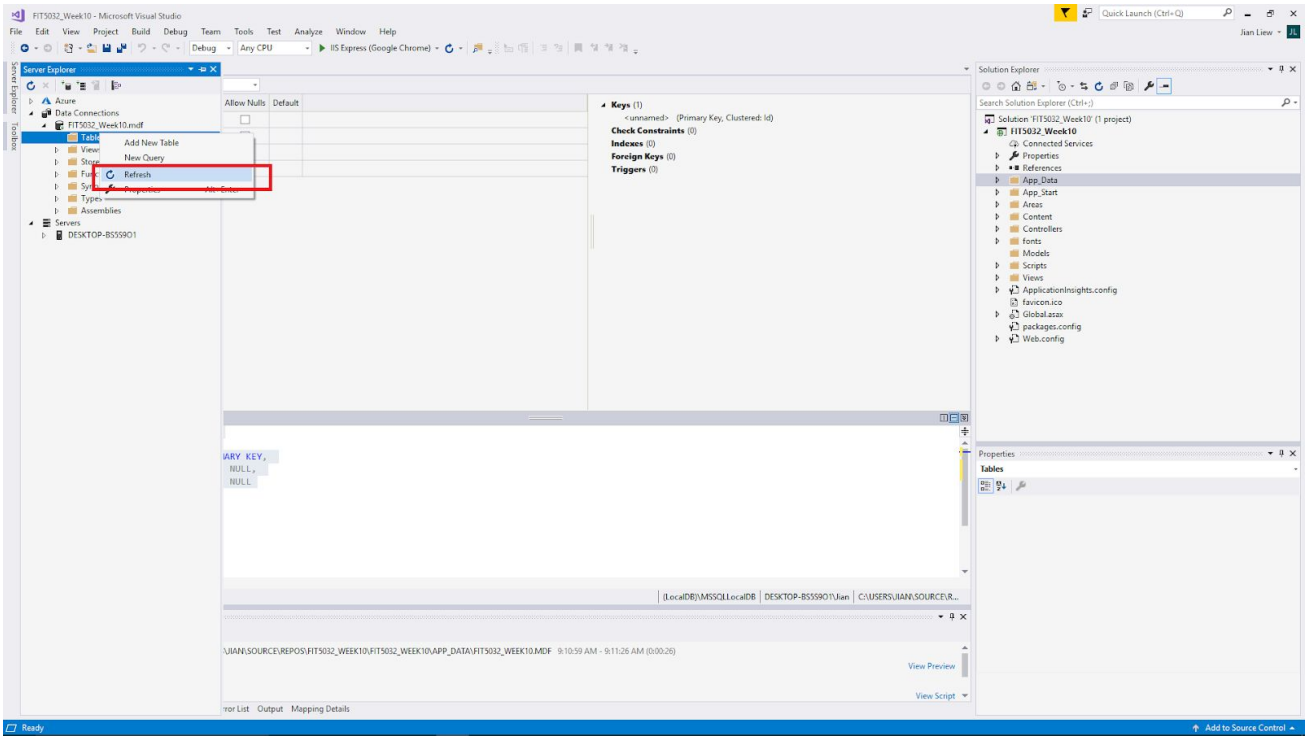


You can use the GUI to add new tables and update the database.



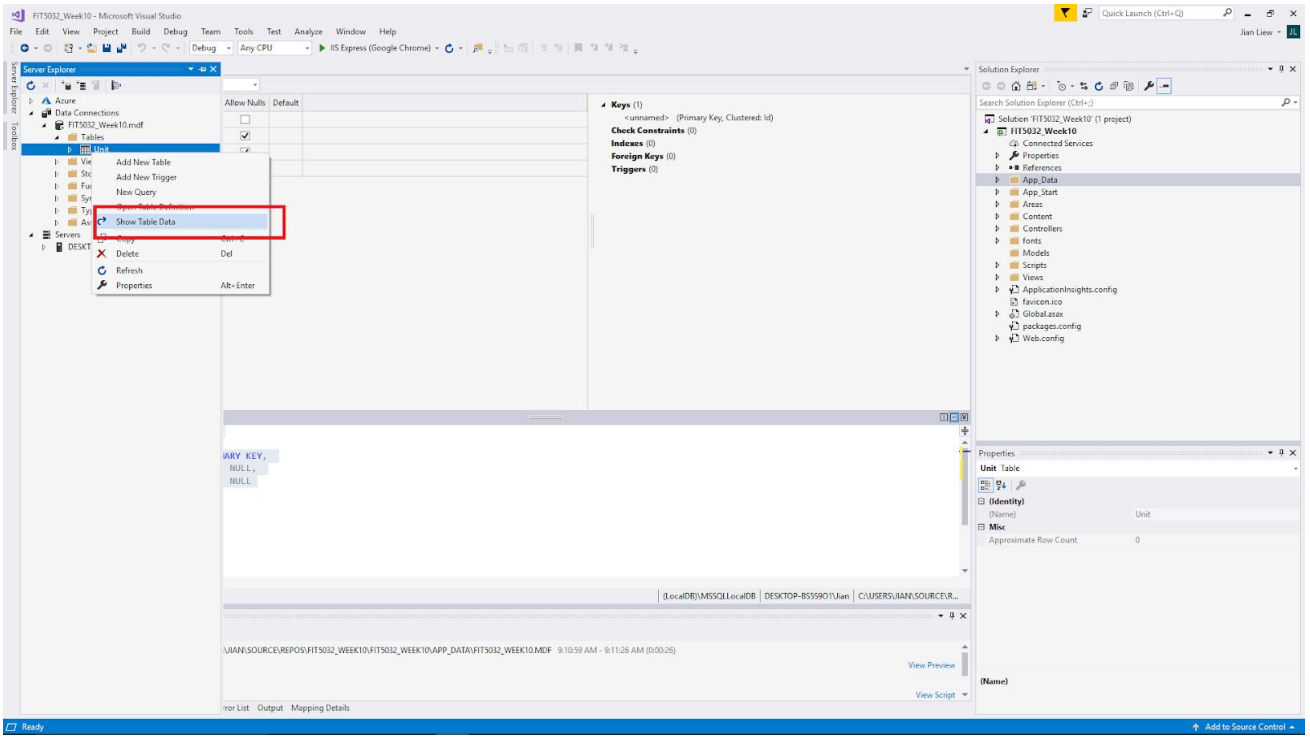


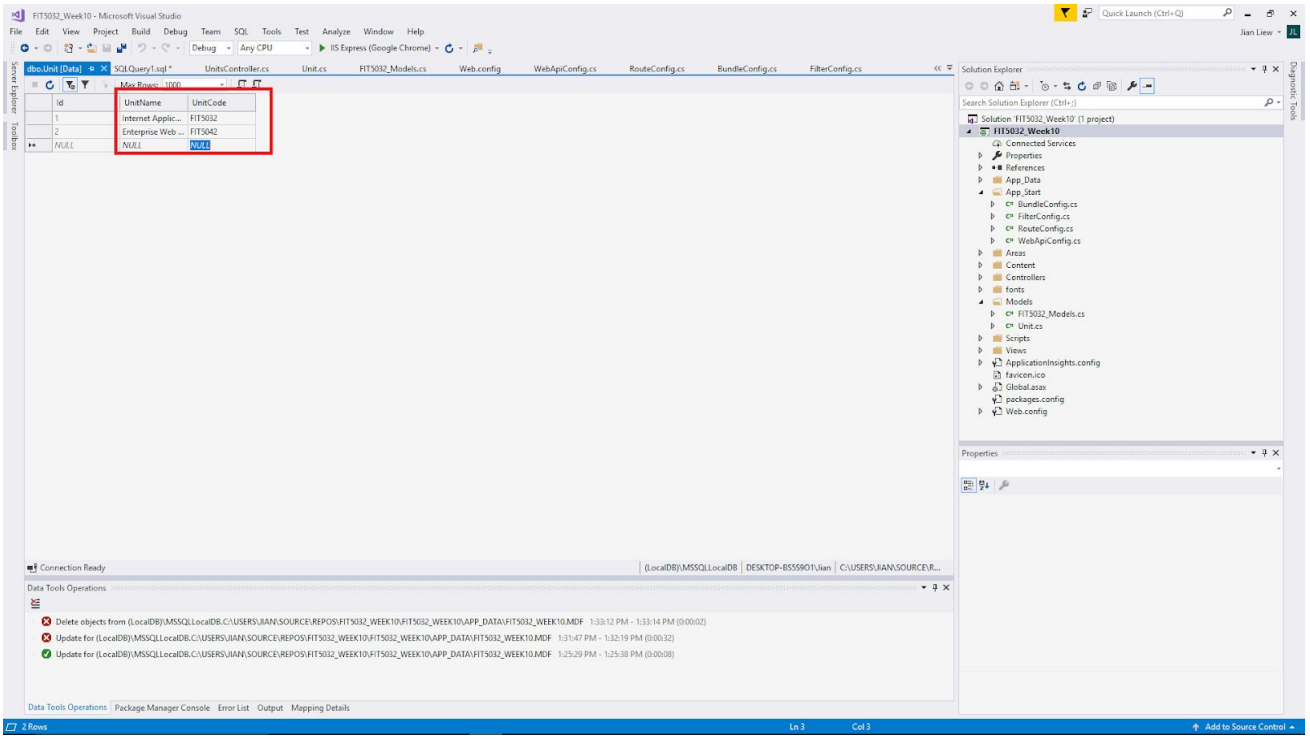
After that, the tables should be seen in the database. (After you hit the Refresh button)



Step 7

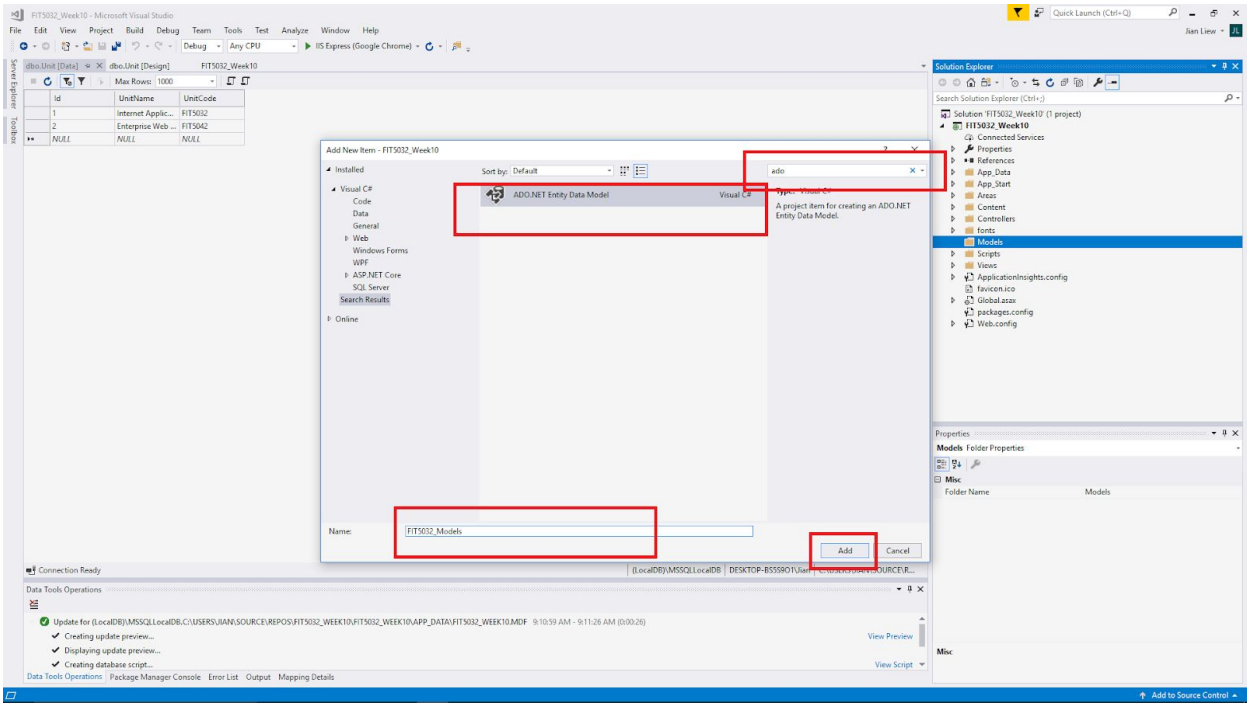
I will add 2 rows of information into my table.



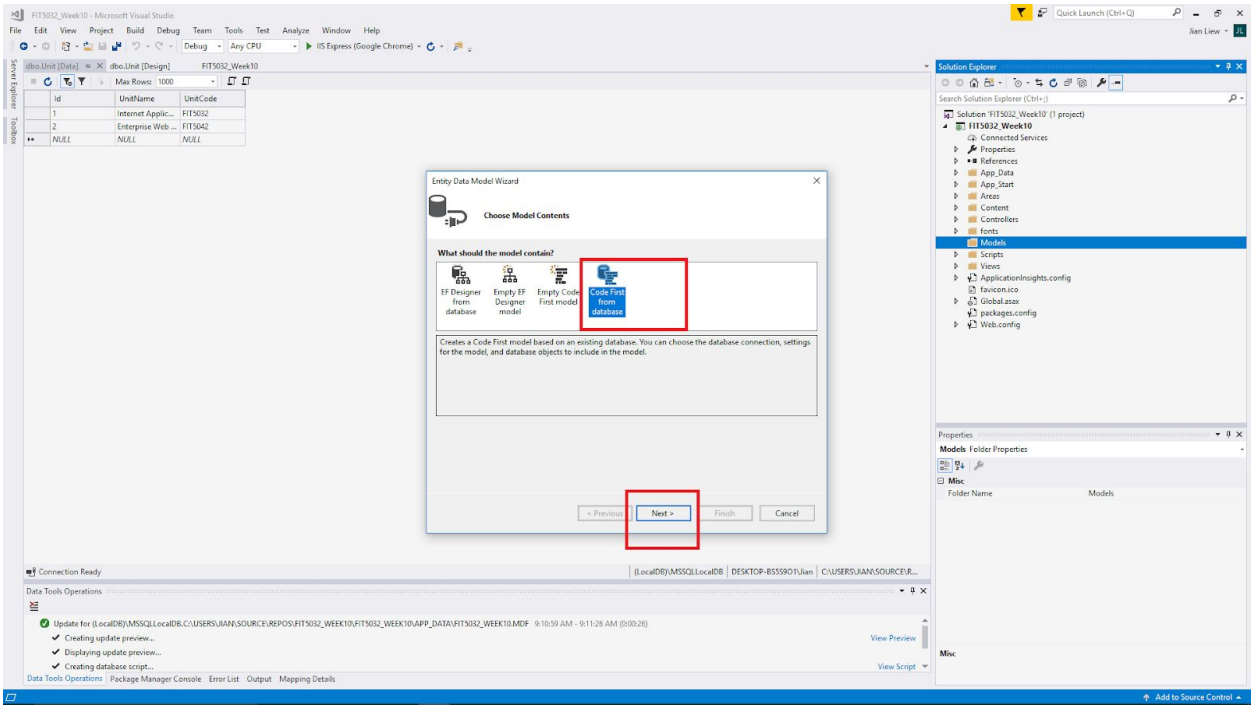


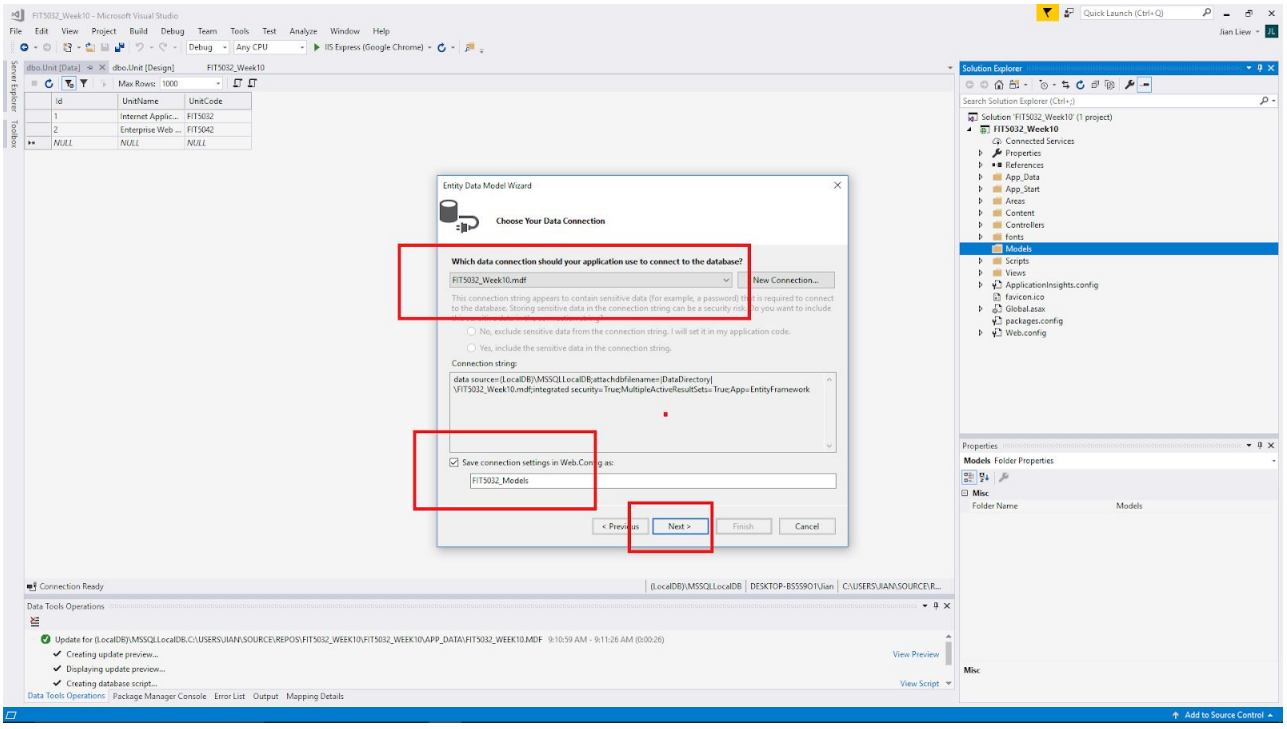
Step 8

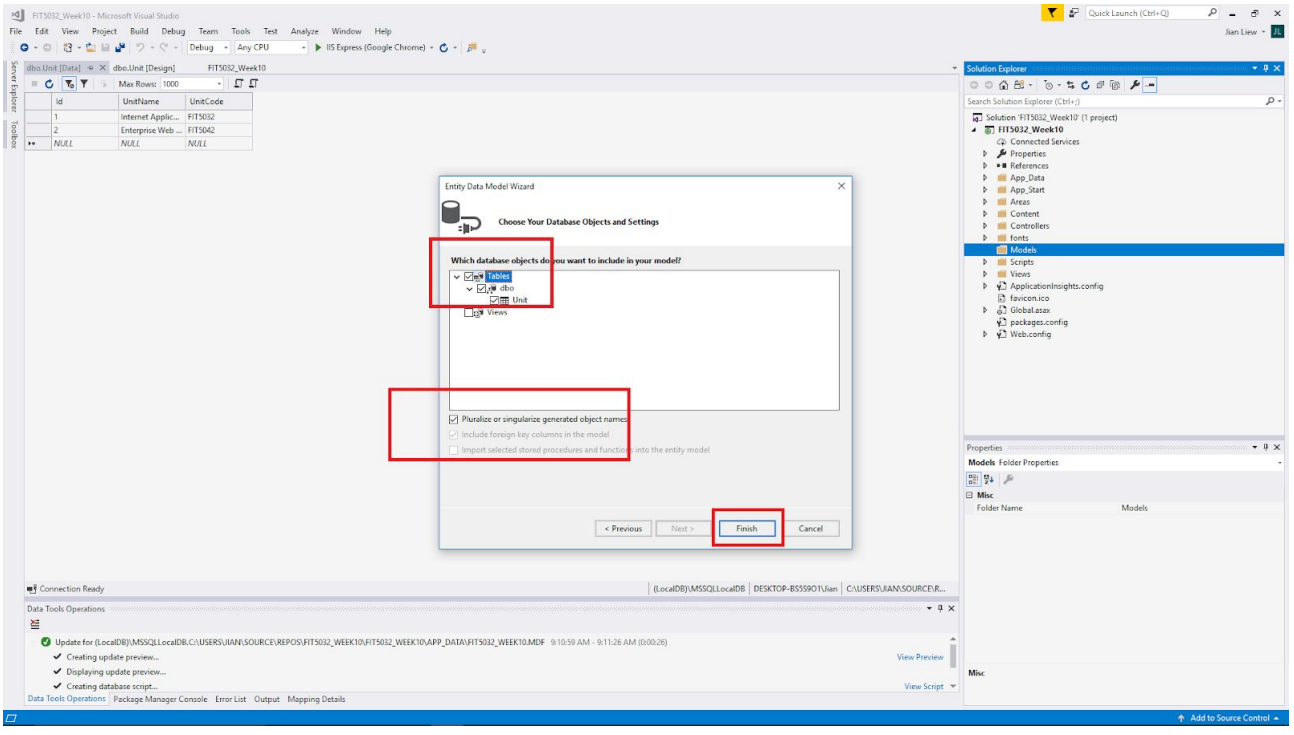
Now I will use the Database First approach to generate my models. (Right click on Models and Add New Item)

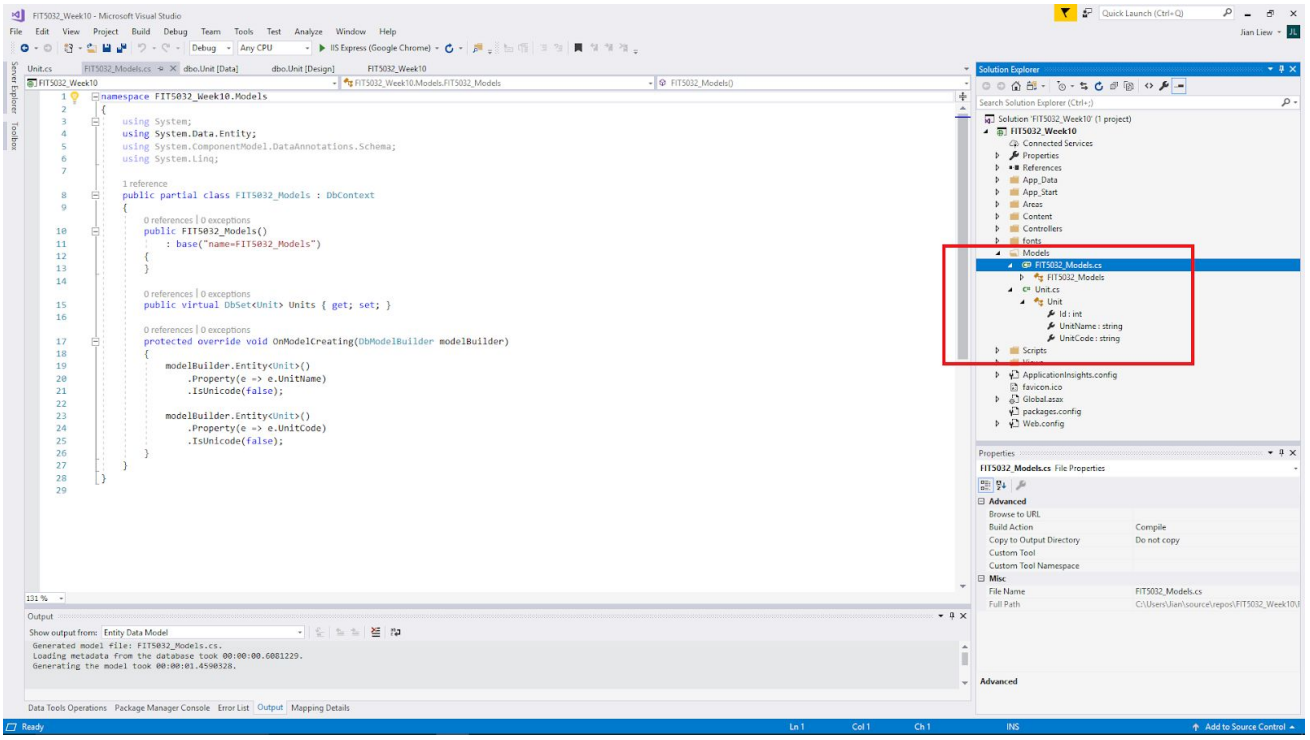


We can go either using the EF designer or the Code First when we generate the tables from an existing database. I will go with Code First as it is cleaner.

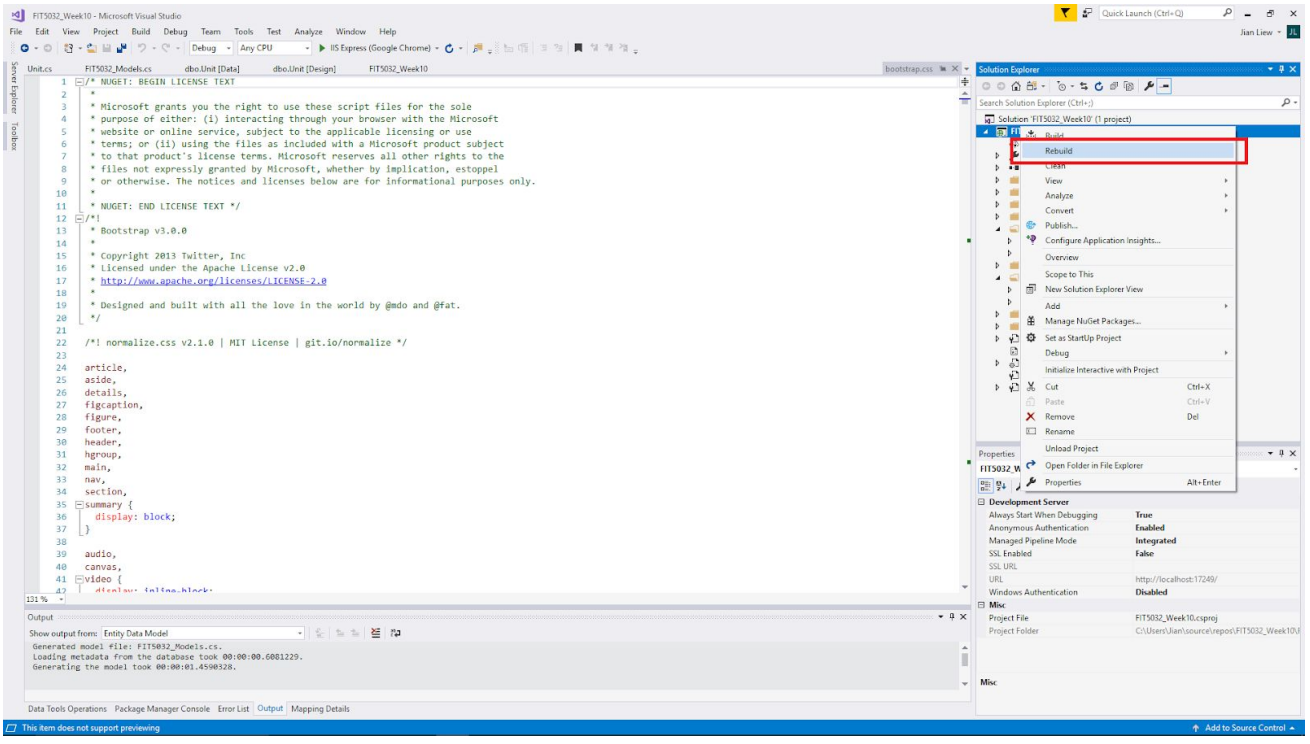



Entity Data Model Wizard
Choose Your Data Connection
Which data connection should your application use to connect to the database?
FIT5032_Week10.mdf
New Connection...
Connection string:
Save connection settings in Web.Config as:
FIT5032_Models
< Previous
Next >
Finish
Cancel


Entity Data Model Wizard
Choose Your Database Objects and Settings
Which database objects do you want to include in your model?
Tables
dbo
Unit
Views
Pluralize or singularize generated object names
Include foreign key columns in the model
< Previous
Next >
Finish
Cancel

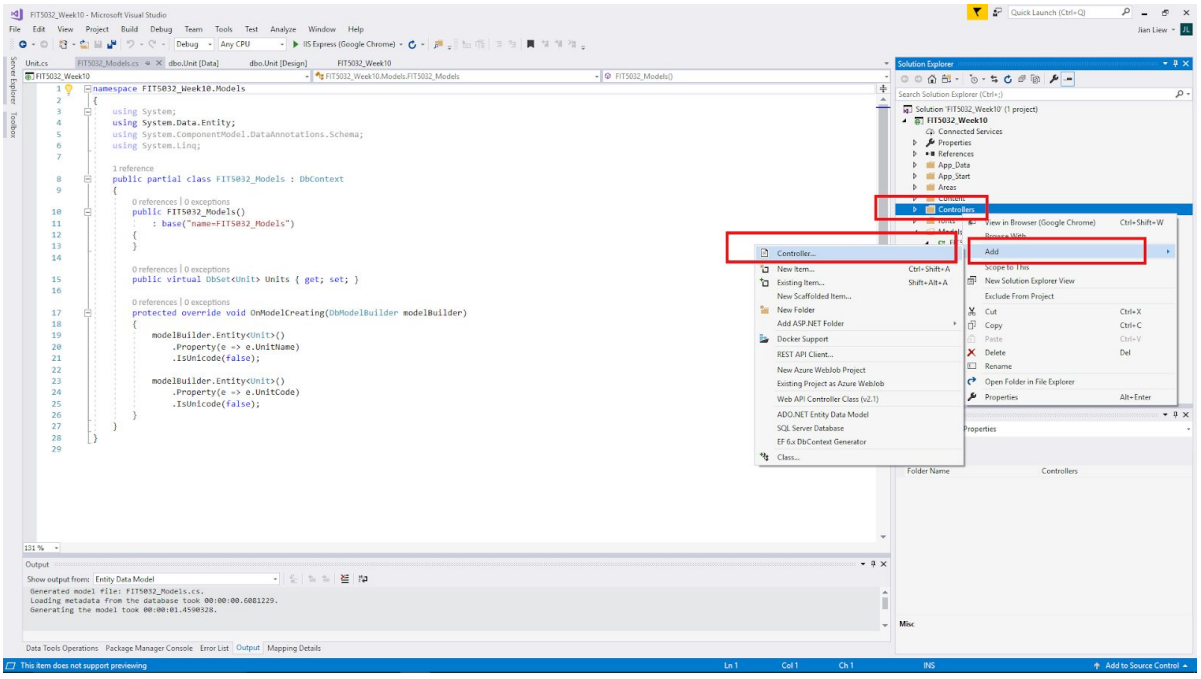


After that, I will rebuild the project. (If you do not do this, you will have troubles in the next step)

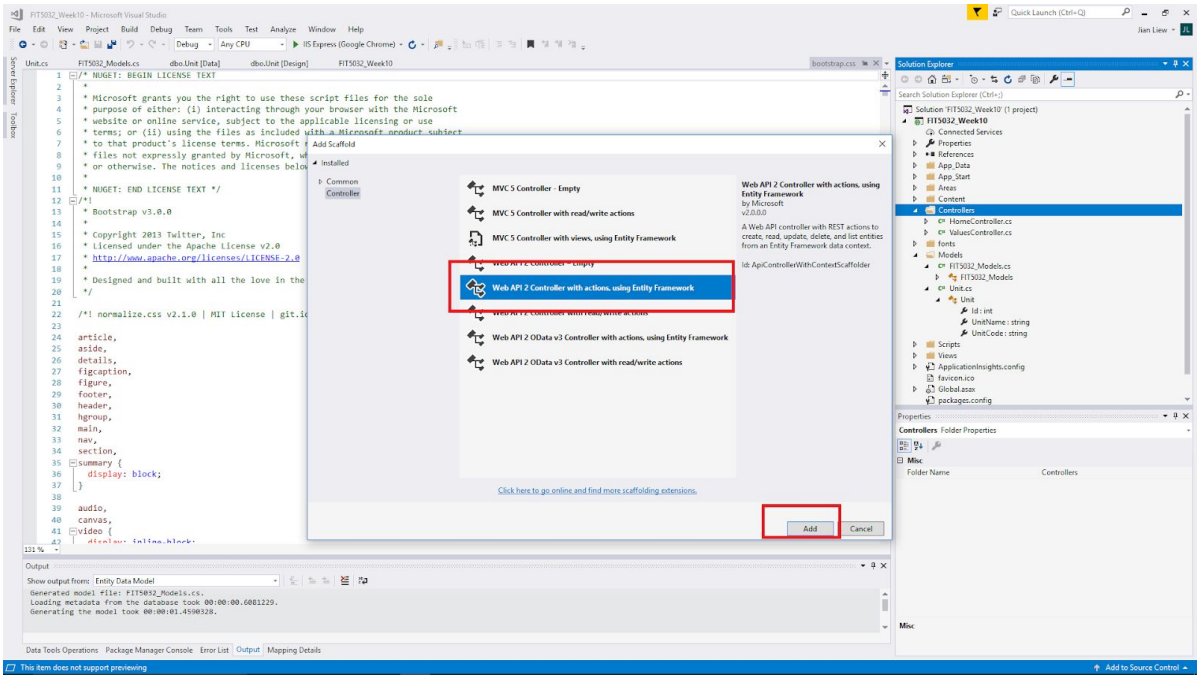


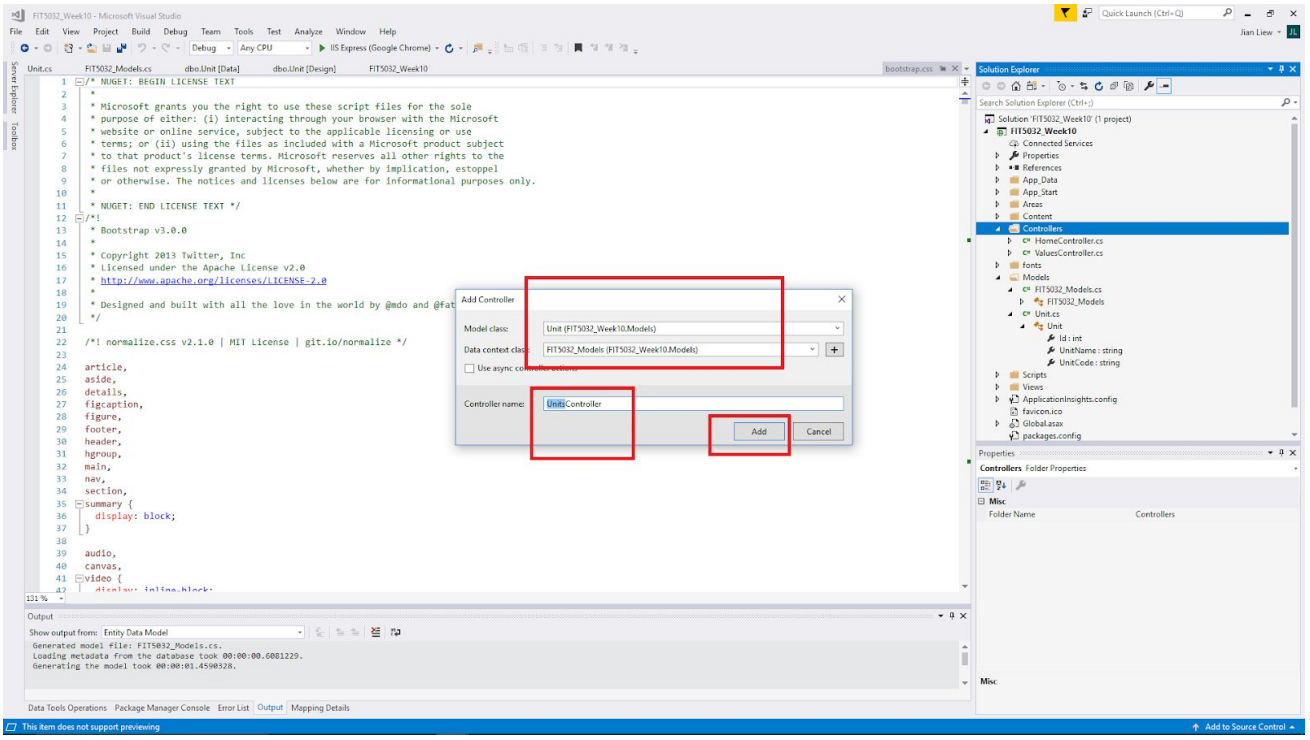
Step 8

Now I will introduce RESTful endpoints. (Done by creating Controllers). This can be done as long as you have the models. So, you can use other approaches like Model or Code.

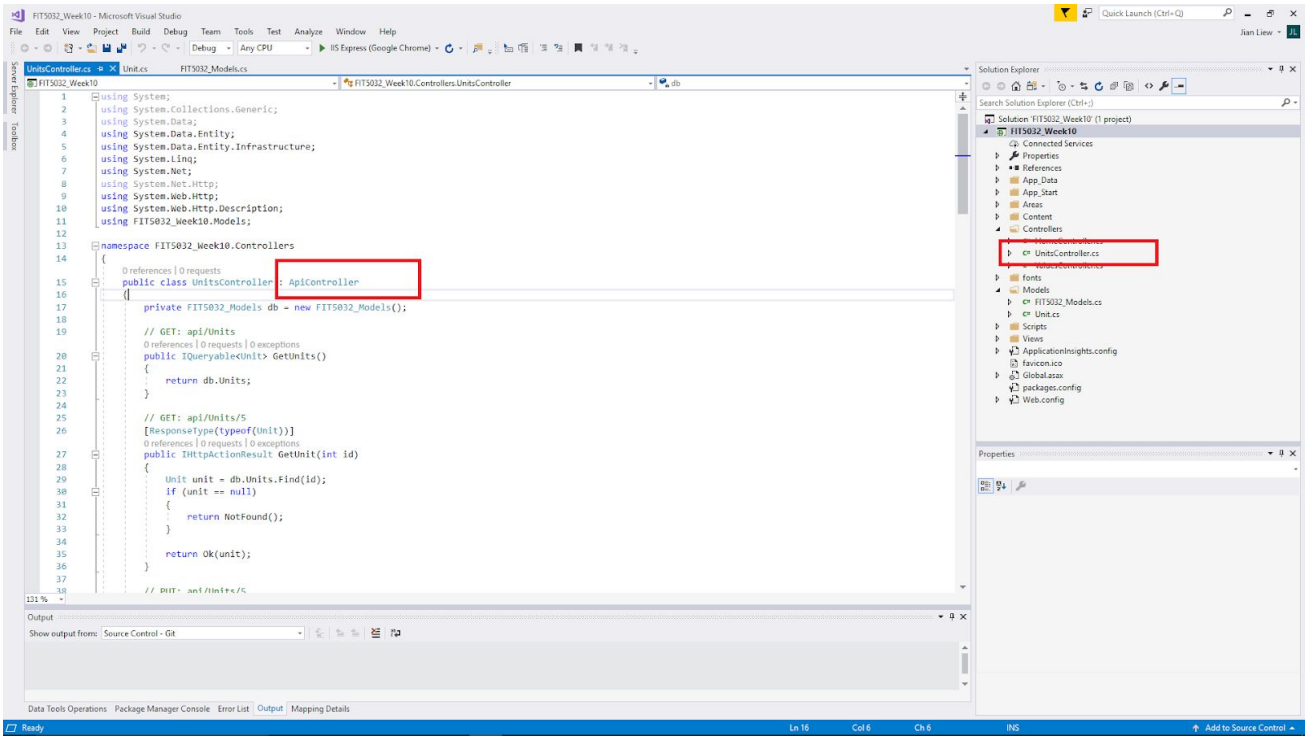


I will use the Web API 2 Controller option. (Refer screenshot for more details)





Visual Studio will automatically generate the codes needed.



You will notice that the UnitsController inherits the ApiController instead of Controller.

Step 9

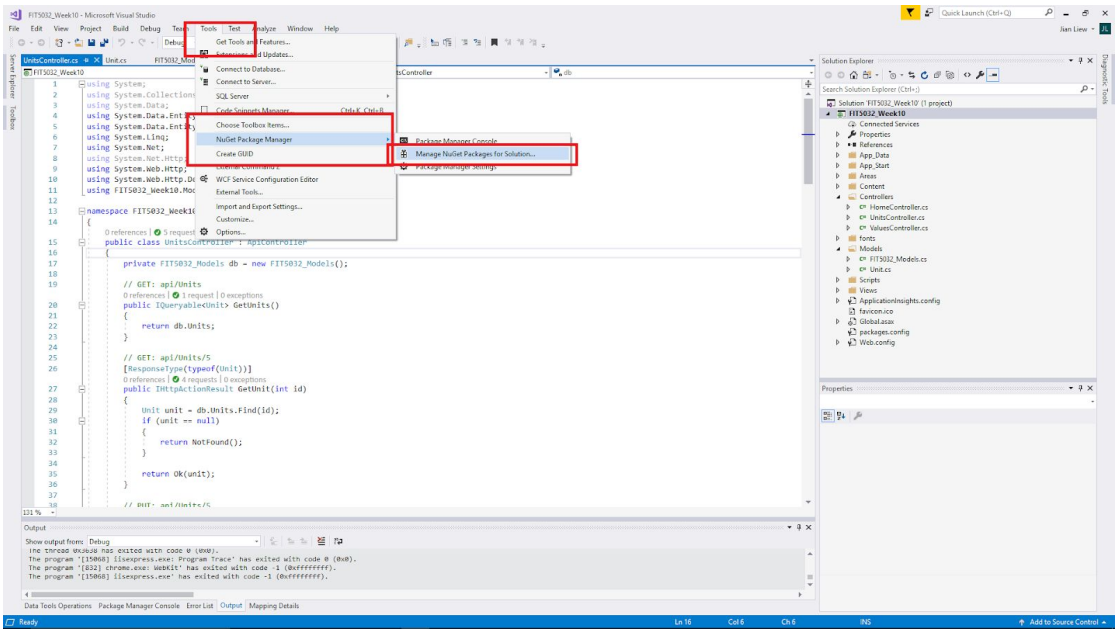
In order to test if the endpoints are working, you should first run the solution. I will test it on my browser.

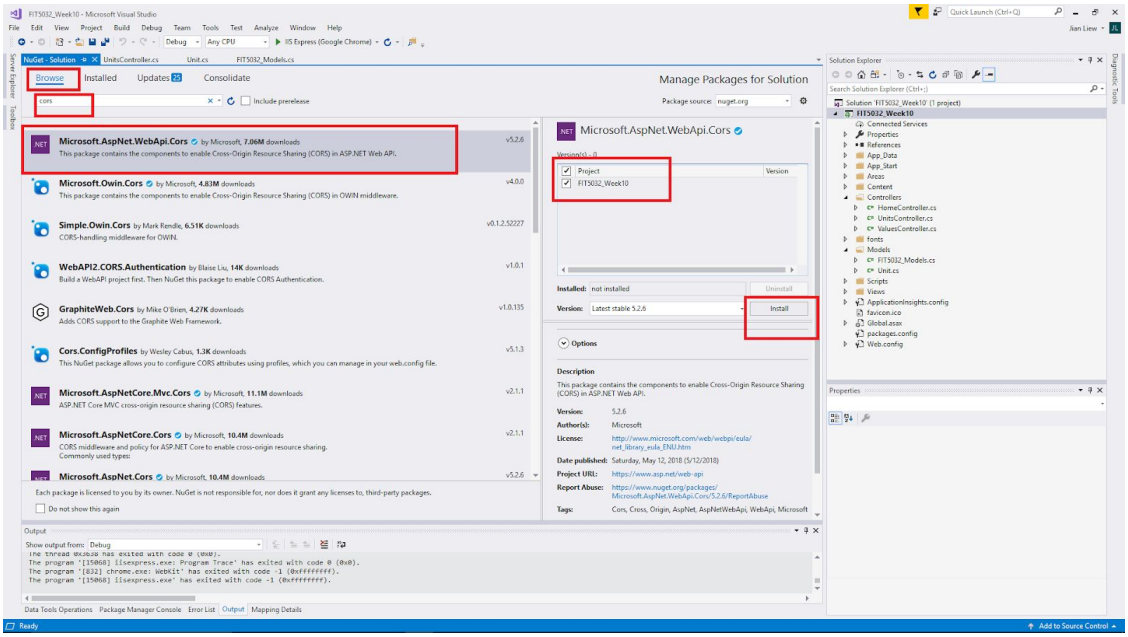
Step 10

You can also install Postman to test out your end points. (Postman is not installed in the Monash lab computers). After you have created and test your endpoints, you should allow other consumers to consume the resource you have created.

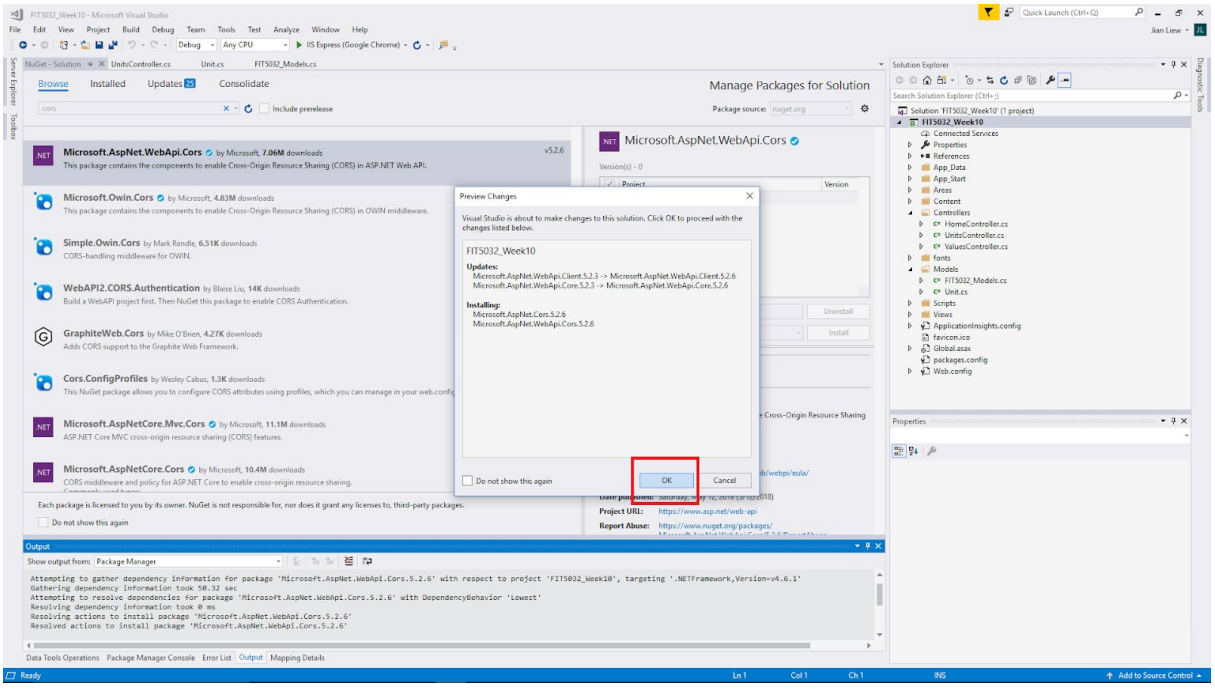
I will enable CORS on my API server. Please read more about CORS here https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

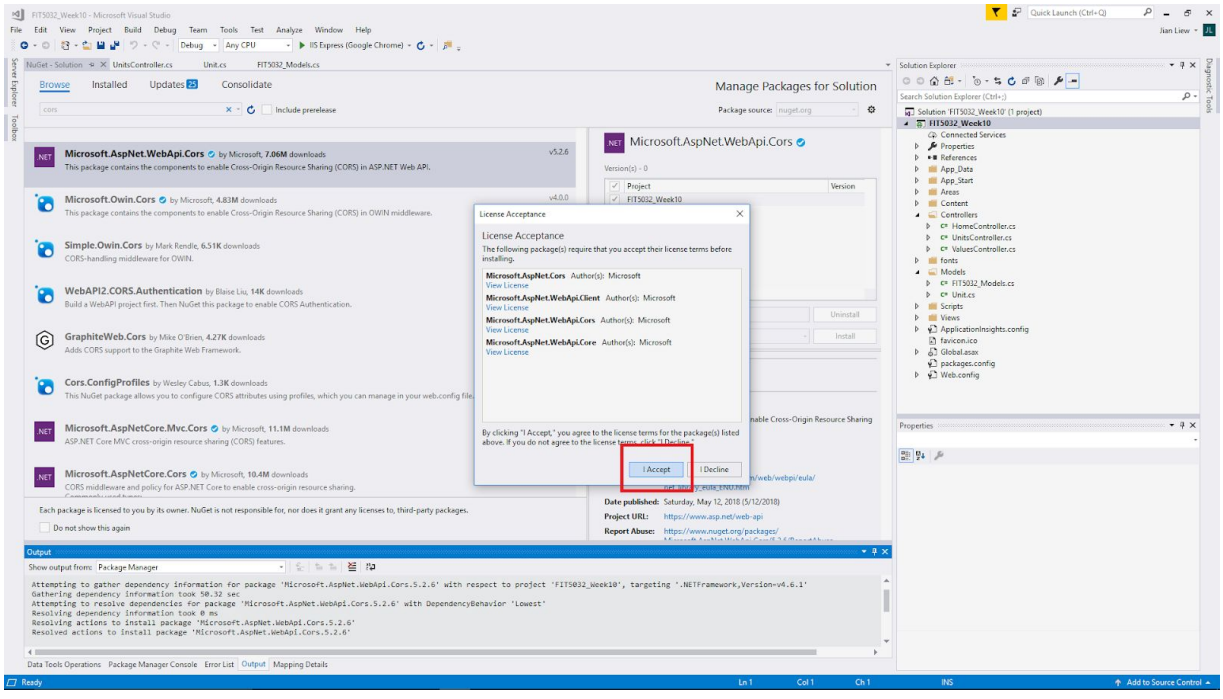
This will be done by installing the NuGet package.





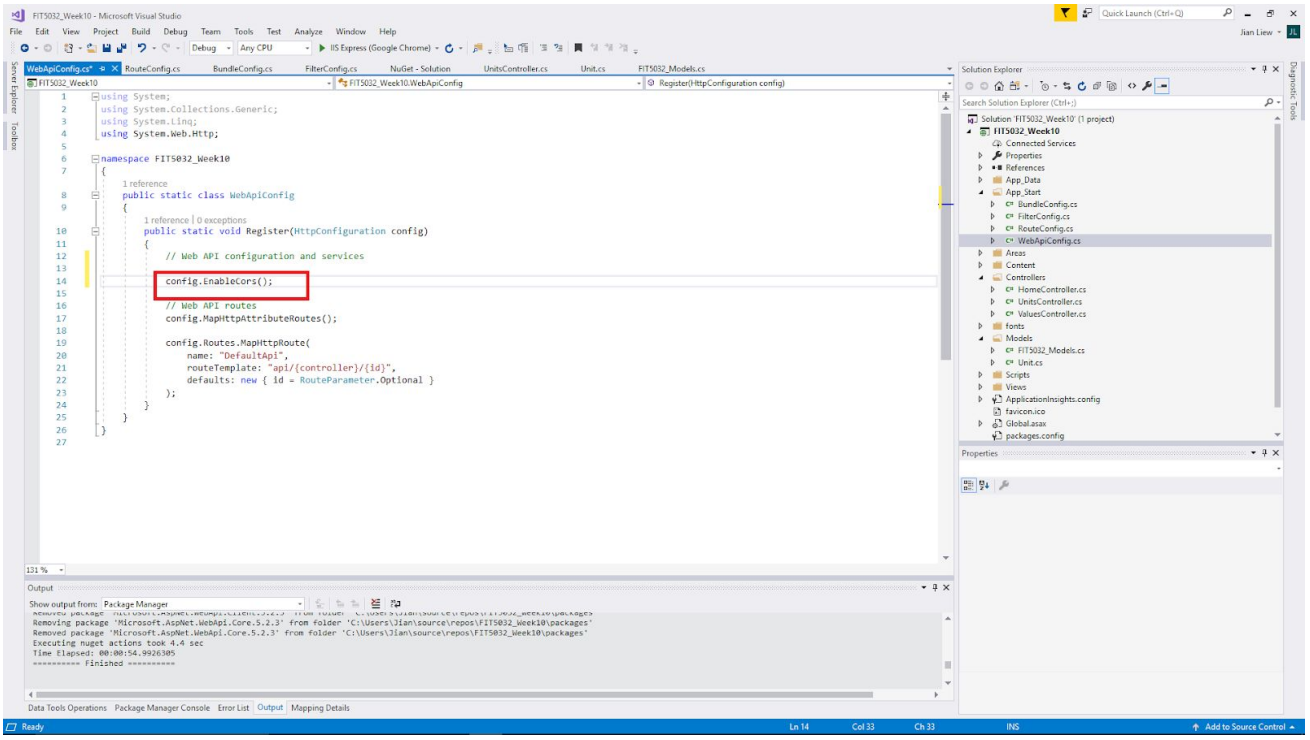
Step 11


Preview Changes
Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.
FIT5032_Week10
Updates:
Microsoft.AspNet.WebApi.Client.5.2.3 -> Microsoft.AspNet.WebApi.Client.5.2.6
Microsoft.AspNet.WebApi.Core.5.2.3 -> Microsoft.AspNet.WebApi.Core.5.2.6
Installing:
Microsoft.AspNet.Cors.5.2.6
Microsoft.AspNet.WebApi.Cors.5.2.6
Do not show this again
OK
Cancel

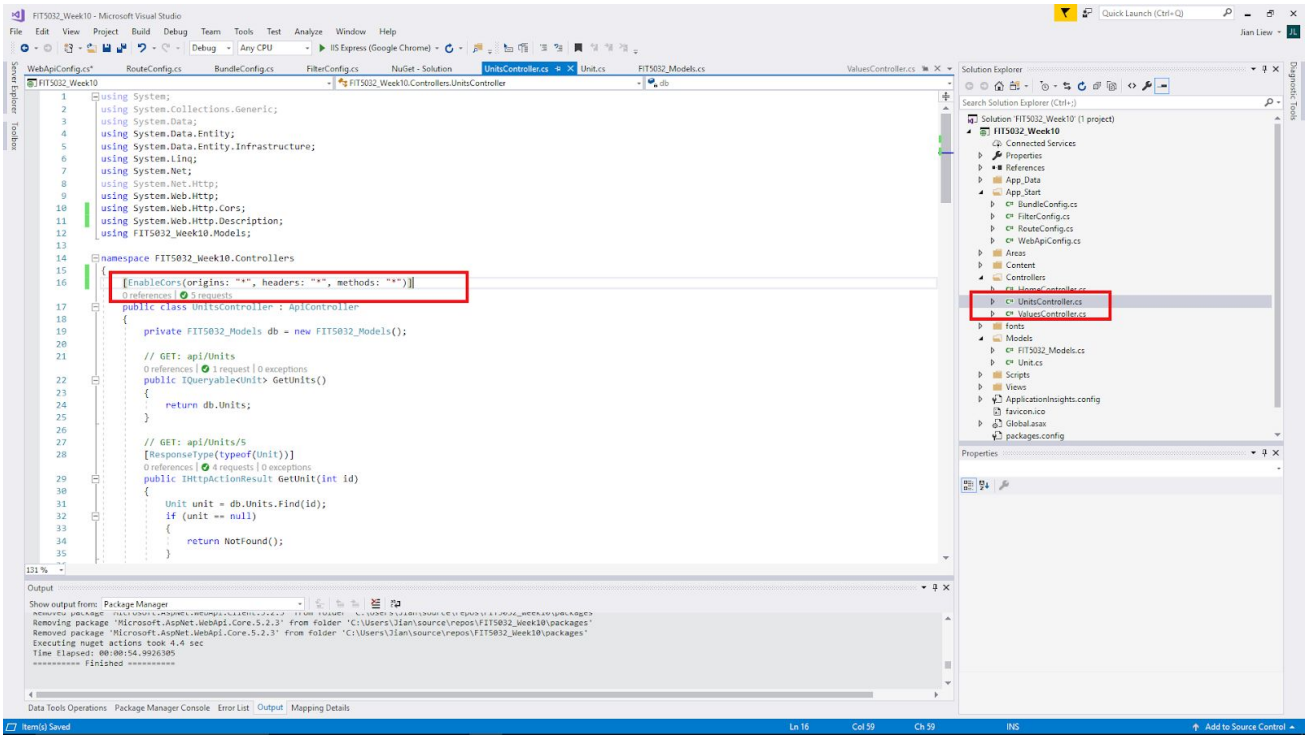

License Acceptance
License Acceptance
The following package(s) require that you accept their license terms before installing.
Microsoft.AspNet.Cors Author(s): Microsoft
View License
Microsoft.AspNet.WebApi.Client Author(s): Microsoft
View License
Microsoft.AspNet.WebApi.Cors Author(s): Microsoft
View License
Microsoft.AspNet.WebApi.Core Author(s): Microsoft
View License
By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."
I Accept
I Decline

Step 12

I will add the EnableCors line at the WebApiConfig.cs

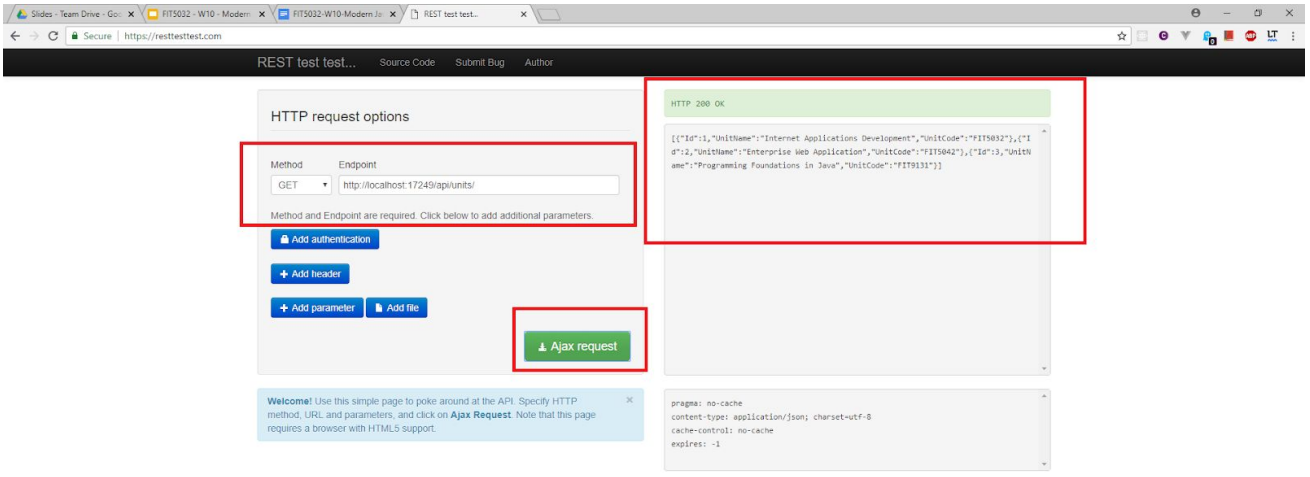


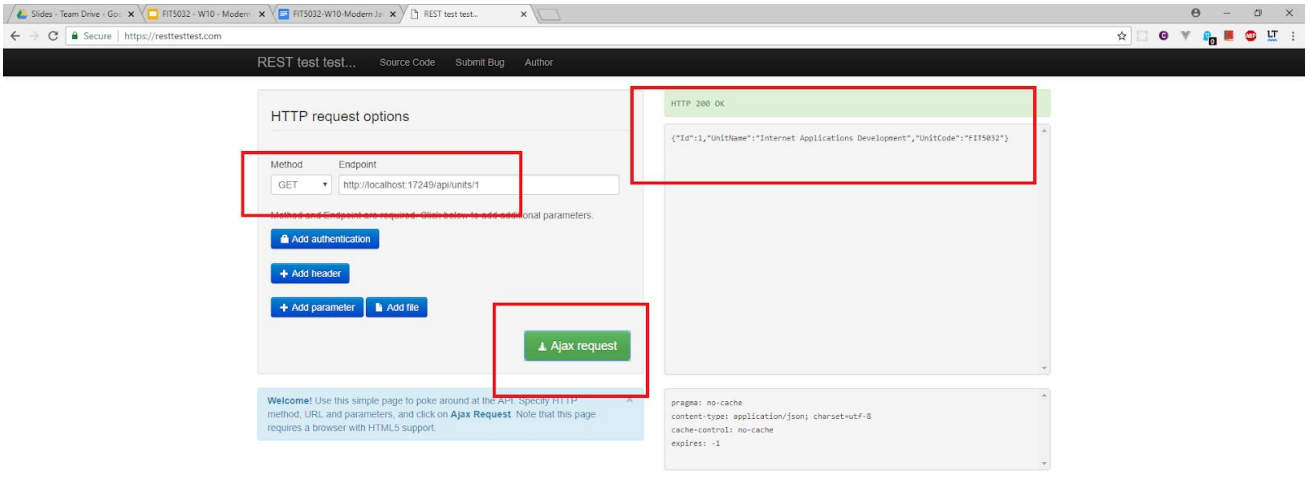
I will also use Attribute settings at the UnitControllers to allow Cors from any url. It is pretty self explanatory.



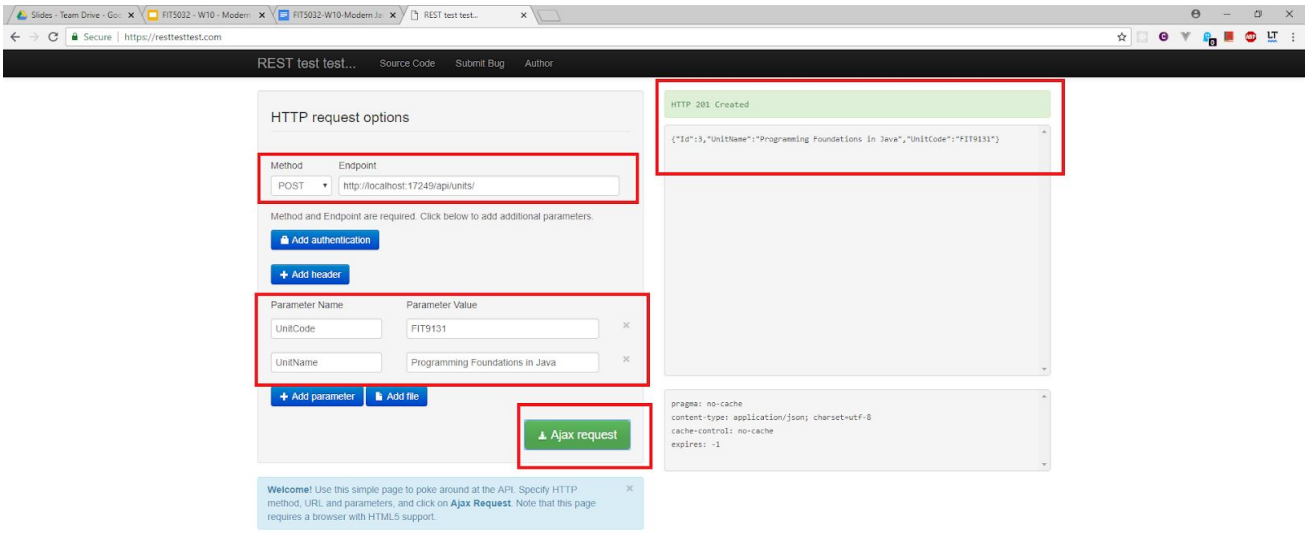
Step 13

We can now try to test it out. (I am going to use the web site resttestest.com). If you have POSTMAN installed you can use it. Or you can use command line tools like cURL or wget.



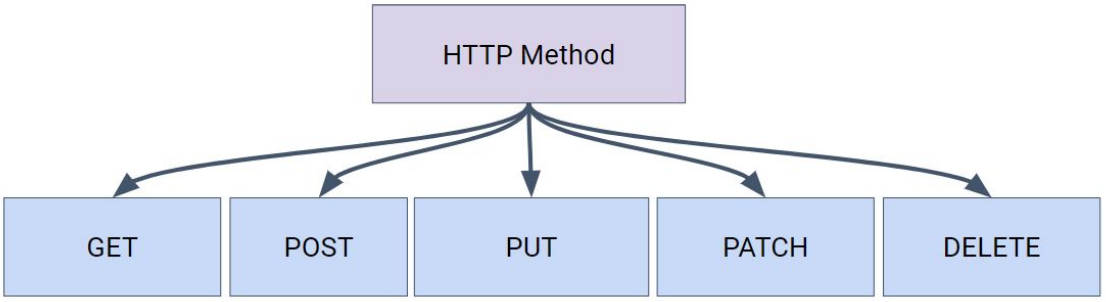


You can also try the POST request.



Upon the completion of this task you have completed the first part of this lab which involves the creation of a WEB API using Visual Studio.

You can pry into the source codes and see how it handles different request like PUT and etc.



Normally a good RESTful API should define the various actions that will happen upon these request. We have tested out the GET and POST so far. Please try the other methods and see how they work.

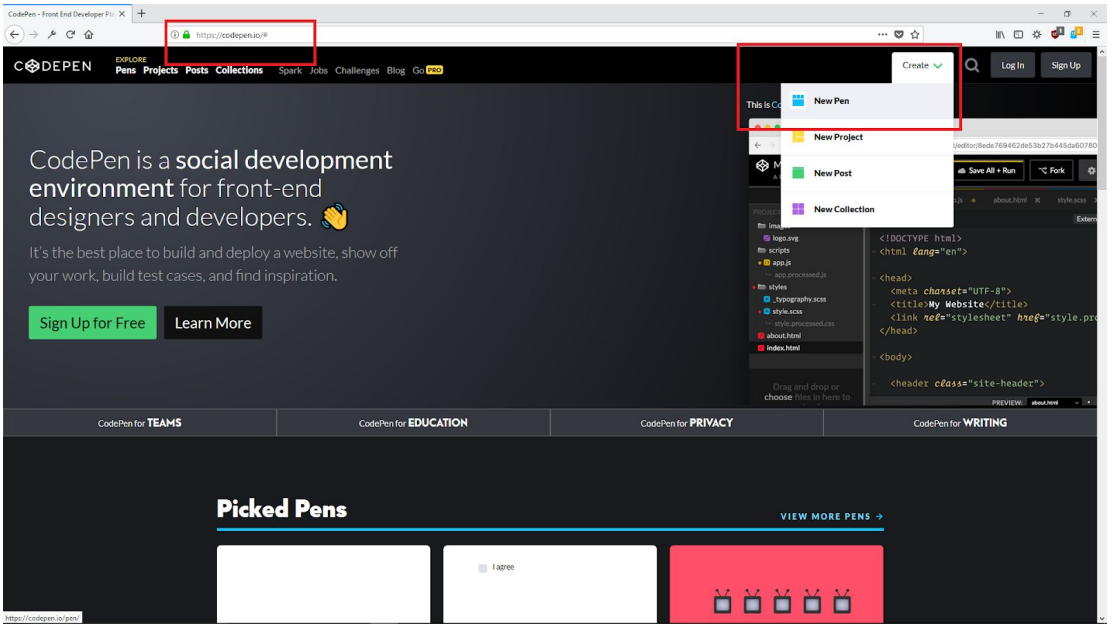
Creating a Simple AngularJS Application

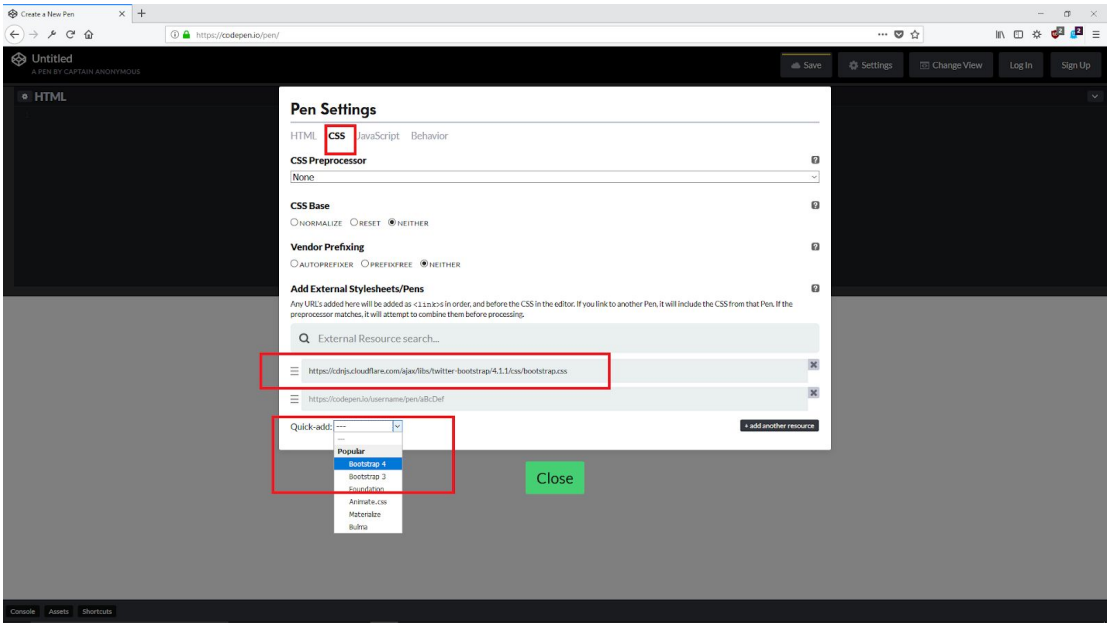
Once we have created the RESTful endpoints, we will now create a simple JavaScript application. The reason we went about creating a RESTful service first is because it is a very common scenario to consume a RESTful service or WEB API. Most online tutorials tend to "over-engineer" this however, it is quite easy to demonstrate this in a simple manner. (The RESTful endpoints must be working for the AngularJS application to work)

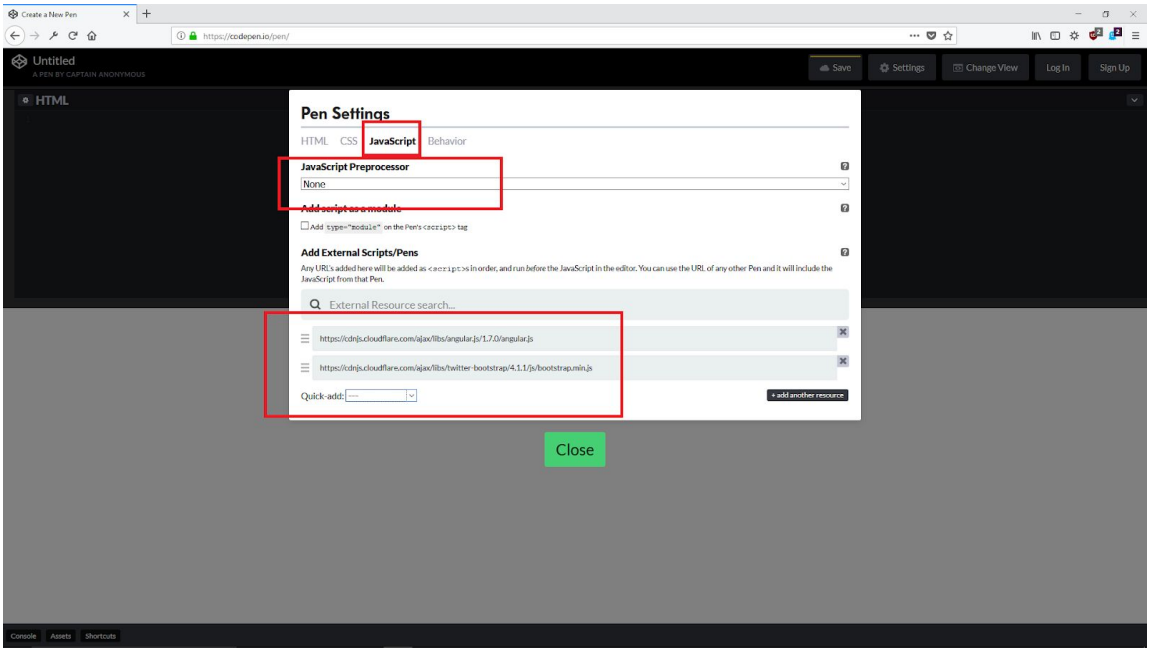
In order to simplify the development of it, I will use an online tool called CodePen. (It is not needed to use the IDE to test it as it is pointless for it to be demonstrated it using Visual Studio)

The final product can be found here - https://codepen.io/jianloong/pen/bjpjoK

Following the following steps are highly optional







Here, I am using CodePen inbuilt features so that it will load the needed resources. It is similar to loading it with src tags.

HTML

```
<div ng-app="demoModule" id="body" class="container">
  <div ng-controller="demoCtrl">
    <h2>FIT5032 Simple AngularJS Demo</h2>
    <div ng-if="noResult" class="alert alert-info" role="alert">
      Failed to retrieve information from the API. Please change the API URL.
    </div>
    <div ng-if="!noResult">
      <h3>List of Units</h3>
      <table ng-cloak class="table">
        <thead>
          <tr>
            <th style="display: none;">ID</th>
            <th>Unit Code</th>
            <th>Unit Name</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr ng-repeat="items in unitData">
            <td hidden="hidden">{{items.Id}}</td>
            <td>{{items.UnitCode}}</td>
            <td>{{items.UnitName}}</td>
            <td>
              <button class="btn btn-primary"
                      ng-model="$scope.Unit"
                      ng-click="edit($index, unitData[$index])">
                Edit
              </button>
              <button class="btn btn-danger"
                      ng-click="delete($index)">
                Delete
              </button>
            </td>
          </tr>
        </tbody>
        <tfoot>
        </tfoot>
      </table>
      <div ng-if="deleteResult" class="alert alert-success" role="alert">
        Unit has been deleted.
      </div>
      <button class="btn btn-primary" data-ng-click="addButton()">Add a unit</button>
      <br >
      <div ng-if="showAdd">
        <div>
          <br >
          <h2>Add A Unit</h2>
        </div>
        <div class="form-group">
          <label for="name">Unit Code</label>
          <input class="form-control" type="text" data-ng-model="Unit.UnitCode" />
        </div>
        <div class="form-group">
          <label for="category">Unit Name</label>
          <input class="form-control" type="text" data-ng-model="Unit.UnitName" />
        </div>
        <div ng-if="addResult" class="alert alert-success" role="alert">
```

```
            Add Successful.
          </div>
          <div>
            <button class="btn btn-primary" data-ng-click="save()">Save</button>
            <button class="btn btn-default" data-ng-click="back()">Back</button>
          </div>
        </div>
        <br >
        <div ng-show="showEdit">
          <div ng-if="updateResult" class="alert alert-success" role="alert">
            Update Successful.
          </div>
          <div>
            <h2>Update Unit</h2>
          </div>
          <div hidden="hidden">
            <label for="id">Id</label>
            <input type="text" data-ng-model="Unit.Id" />
          </div>
          <div class="form-group">
            <label for="name">Unit Code</label>
            <input class="form-control" type="text" data-ng-model="Unit.UnitCode" />
          </div>
          <div class="form-group">
            <label for="category">Unit Name</label>
            <input class="form-control" type="text" data-ng-model="Unit.UnitName" />
          </div>
          <br />
          <div>
            <button class="btn btn-primary" data-ng-click="update()">Update</button>
            <button class="btn btn-default" data-ng-click="cancel()">Cancel</button>
          </div>
        </div>
      </div>
    </div>
  </div>
```

JavaScript

```
// Defining angularjs module
var app = angular.module('demoModule', []);
// You will need to use your newly created Web API here (This one here is mine but your port will be
different)
var apiURL = "http://localhost:17249/api/";

app.controller('demoCtrl', function ($scope, $http, UnitService) {
  $scope.showAdd = false;
  $scope.showEdit = false;
  $scope.unitData = "";
  $scope.noResult = false;

  UnitService.GetAllRecords().then(function (d) {
    $scope.unitData = d.data; // Success
    console.log($scope.unitData);
  }, function () {
    $scope.noResult = true;
  });

  $scope.Unit = {
    Id : '',
    UnitName : '',
    UnitCode : ''
  };

  // Clear
  $scope.back = function () {
    $scope.showEdit = false;
    $scope.showAdd = false;
  }

  $scope.addButton = function () {
    $scope.showAdd = true;
    $scope.showEdit = false;
  }

  //Add New Unit
  $scope.save = function () {
    if ($scope.Unit.UnitCode != "" && $scope.Unit.UnitName != "") {
      $scope.Unit.Id = 0;
      $http({
        method: 'POST',
        url: apiURL + "units",
        data: $scope.Unit
      }).then(function successCallback(response) {
        $scope.unitData.push(response.data);
        $scope.addResult = true;
        $scope.clear();
        alert("Unit Added Successfully");
      }, function errorCallback(response) {

        $scope.noResult = true;
      });
    }
    else {
      alert('Please enter both the unit code and unit name.');
    }
  };
```

```
  $scope.edit = function (index, data) {
    $scope.updateIndex = index;
    $scope.showAdd = false;
    $scope.showEdit = true;
    $scope.Unit = { Id: data.Id, UnitCode: data.UnitCode, UnitName : data.UnitName}
  }

  // Cancel unit details
  $scope.cancel = function () {
    $scope.showEdit = false;
  }

  // Update product details
  $scope.update = function () {
    if ($scope.Unit.UnitTitle != "" && $scope.Unit.UnitName != "") {

      $http({
        method: 'PUT',
        url: apiURL + "units/"+ $scope.Unit.Id,
        data: $scope.Unit
      }).then(function successCallback(response) {
        var id = $scope.Unit.Id;
        //console.log(id);
        //$scope.unitData.splice($scope.updateIndex, 1);
        var unit = $scope.Unit;
        $scope.unitData[$scope.updateIndex] = unit;
        $scope.updateResult = true;
        //$scope.clear();
      }, function errorCallback(response) {
        alert("Error : " + response.data.ExceptionMessage);
      });
    }
    else {
      alert('Please enter the required values.');
    }
  };

  // Delete unit
  $scope.delete = function (index) {
    $http({
      method: 'DELETE',
      url: apiURL + "units/" + $scope.unitData[index].Id

    }).then(function successCallback(response) {
      $scope.deleteResult = true;
    }, function errorCallback(response) {
      alert("Error : " + response.data.ExceptionMessage);
    });
  };

});

app.factory('UnitService', function ($http) {
  var fac = {};
  fac.GetAllRecords = function () {
    return $http.get(apiURL + "units");
  }
  return fac;
});
```

DoubtFire Submission

- A document detailing if the automatically generated web API from Visual Studio follows the best practices
- OR
- A Document showing the screen shots of your web api application running (in PDF document)