
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

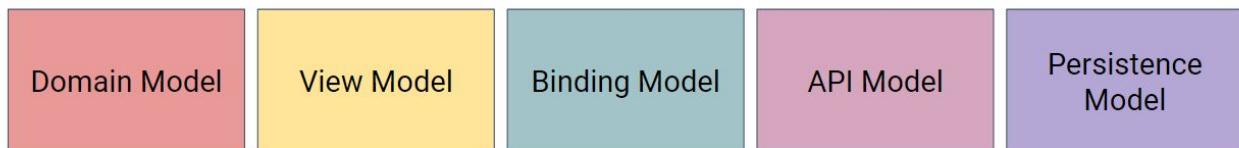
DoubtFire Submission

- 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)

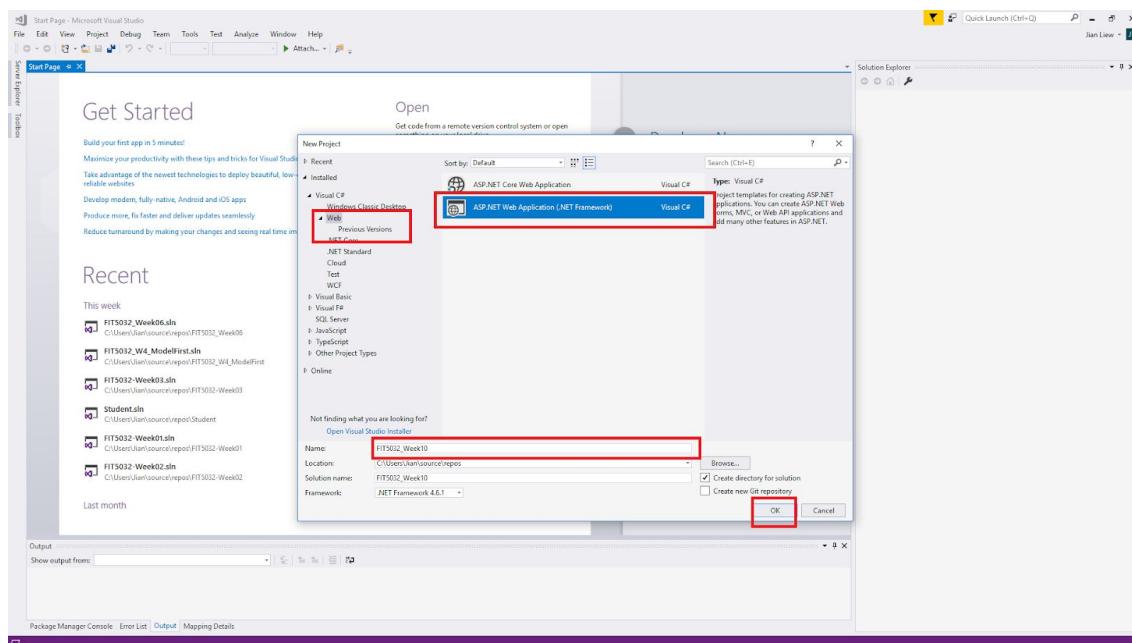
View Model & Binding Model covered on Week 6 (Validation)

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

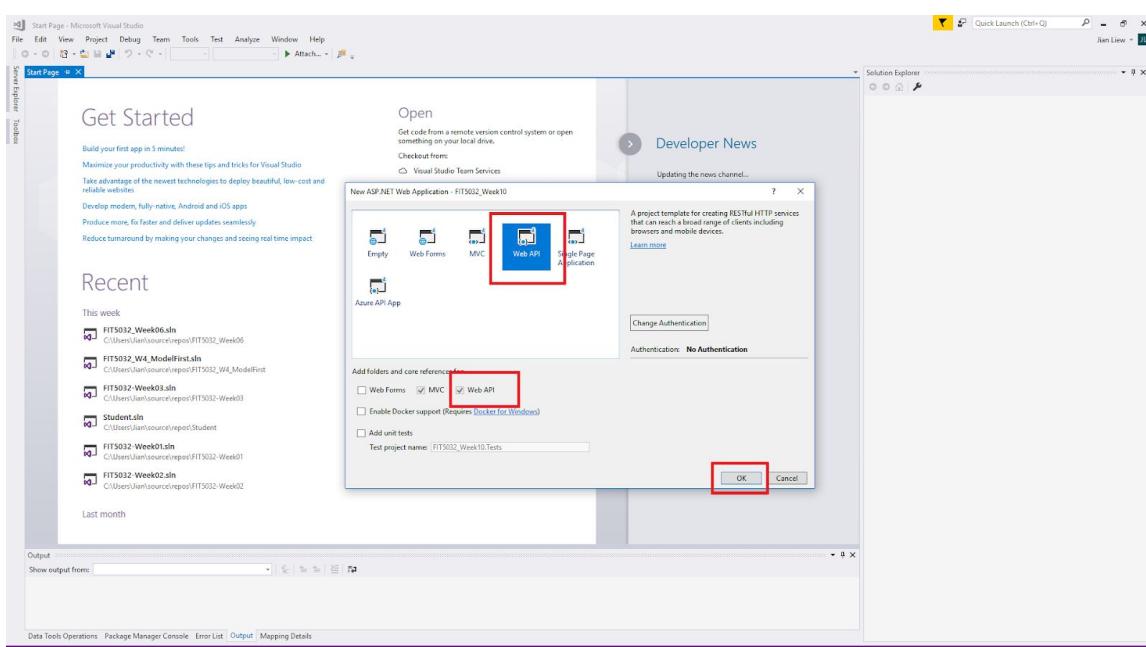
Creating the Web API

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

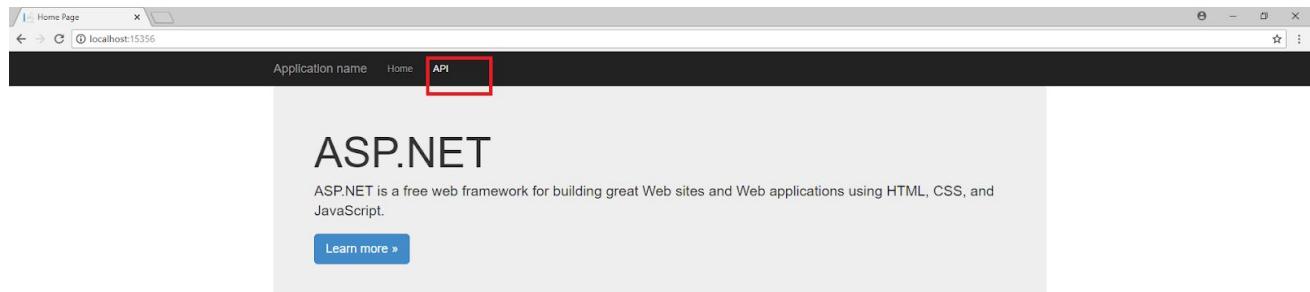


Step 2



Step 3

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



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.

[Learn more »](#)

Getting started

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

[Learn more »](#)

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

[Learn more »](#)

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

[Learn more »](#)

© 2018 - My ASP.NET Application



ASP.NET Web API Help Page

Introduction

Provide a general description of your APIs here.

Values

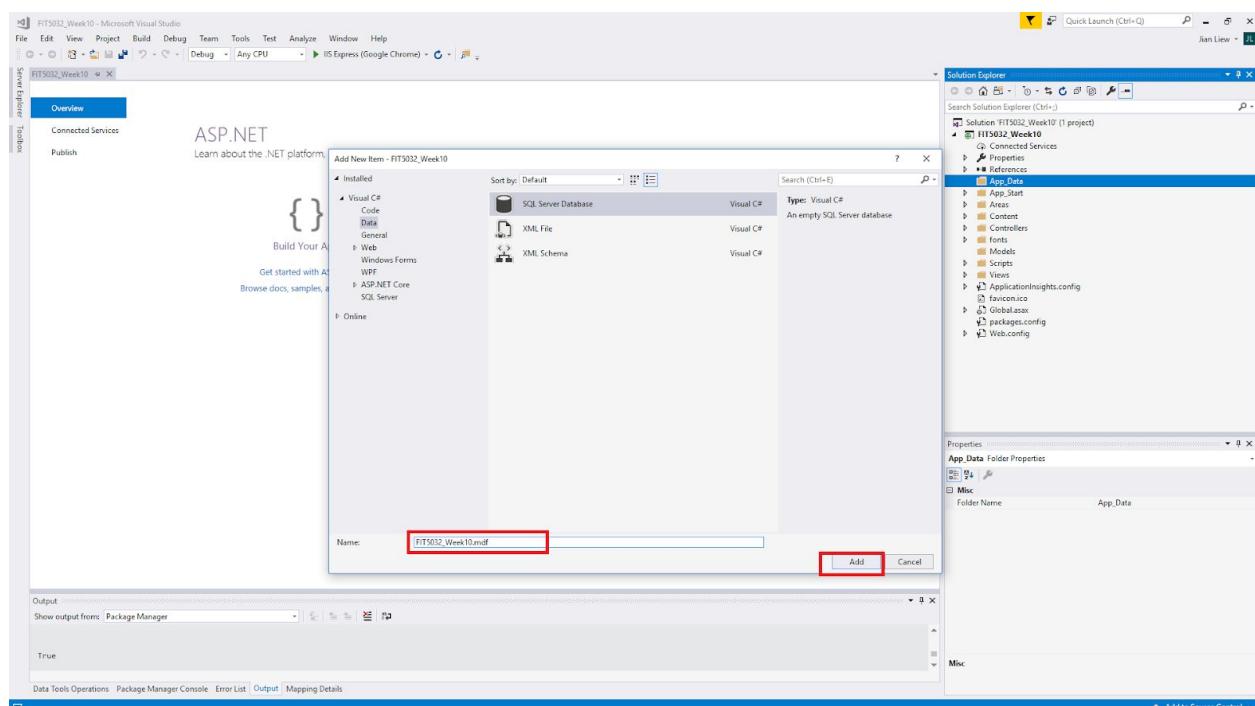
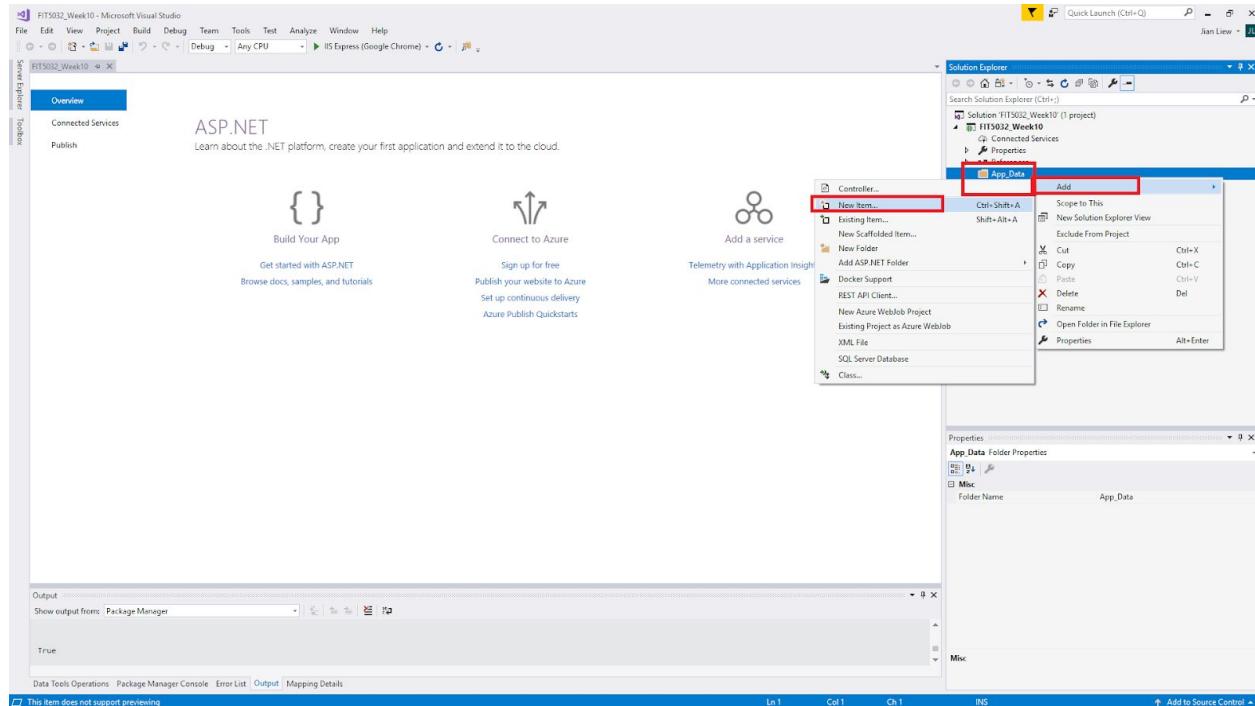
API	Description
GET api/values	No documentation available.
GET api/values/{id}	No documentation available.
POST api/values	No documentation available.
PUT api/values/{id}	No documentation available.
DELETE api/values/{id}	No documentation available.

© 2018 - My ASP.NET Application

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.

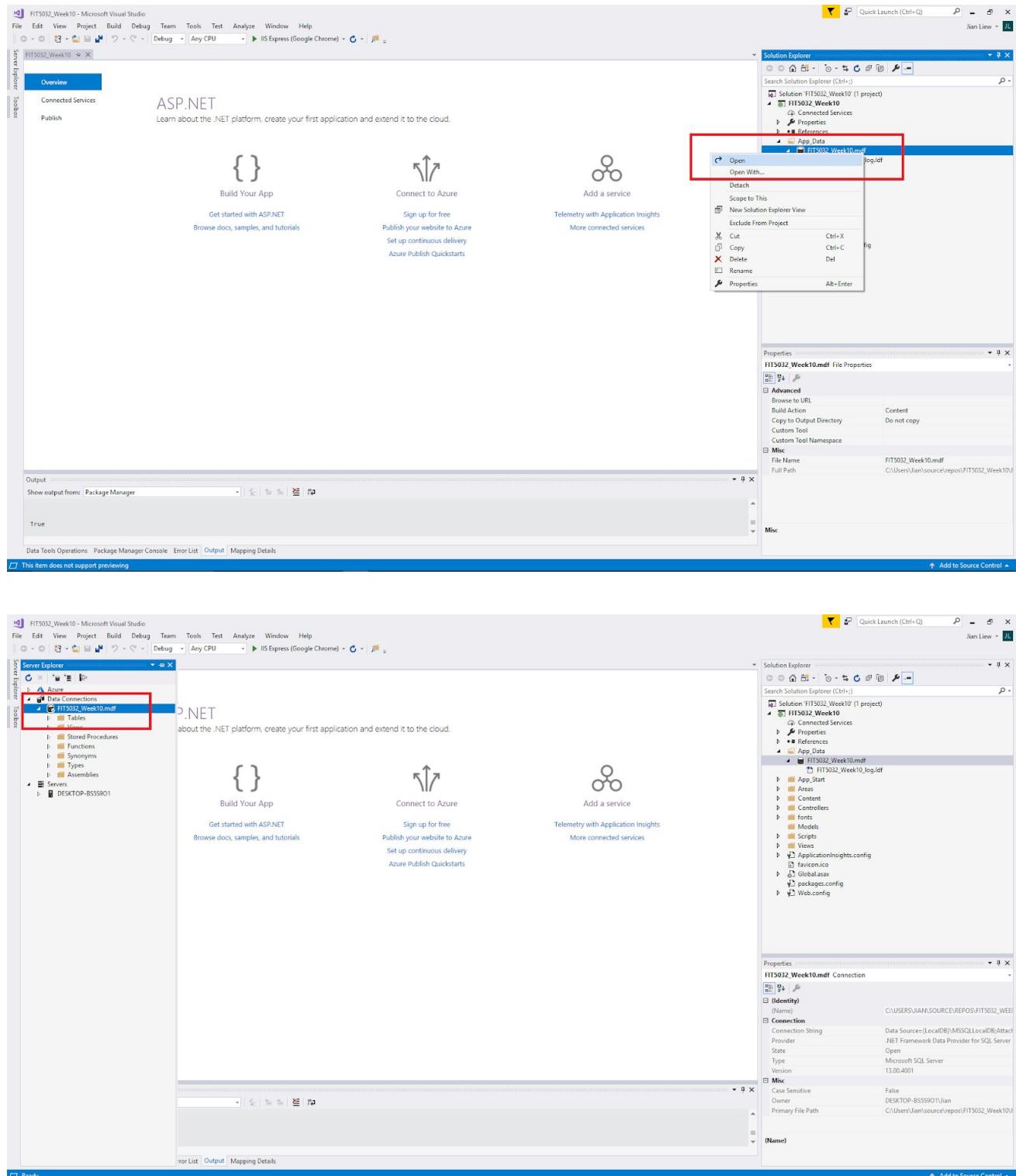
Step 4

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



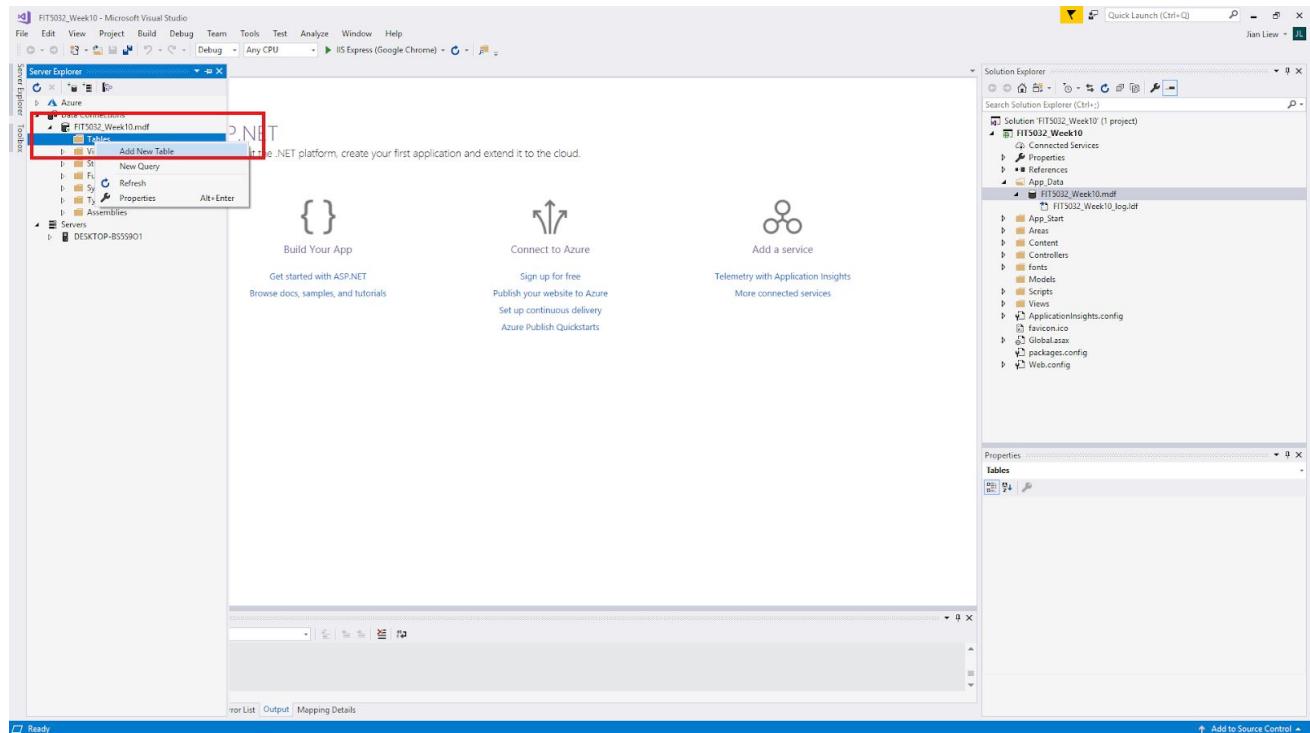
Step 5

Open the .mdf file.

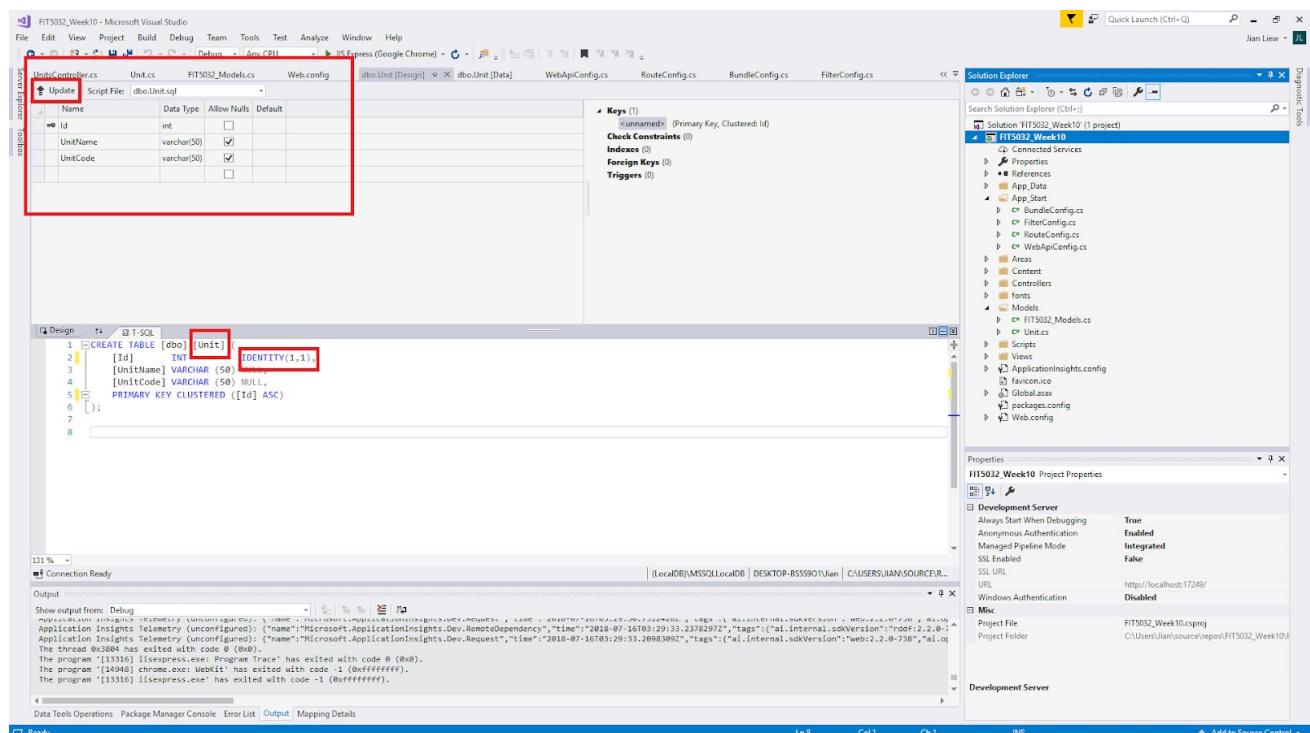


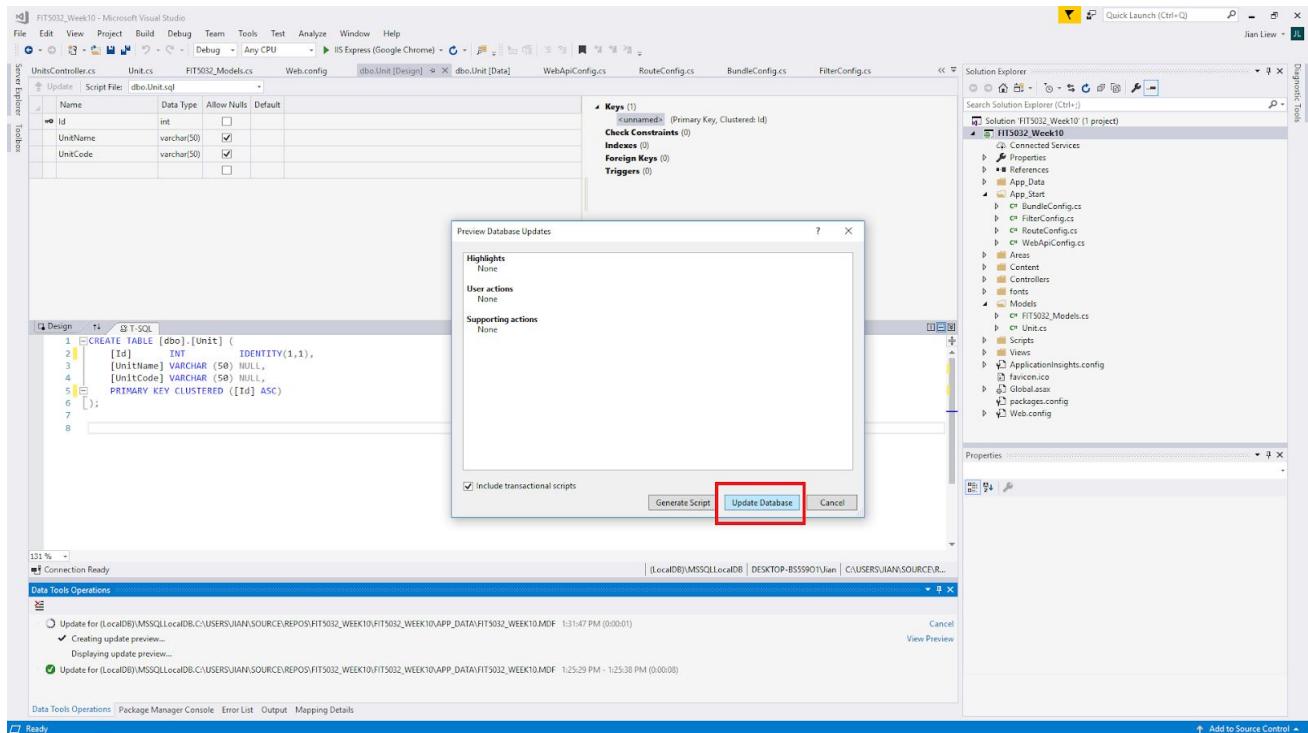
You should see the newly created .mdf file under the server explorer.

Step 6

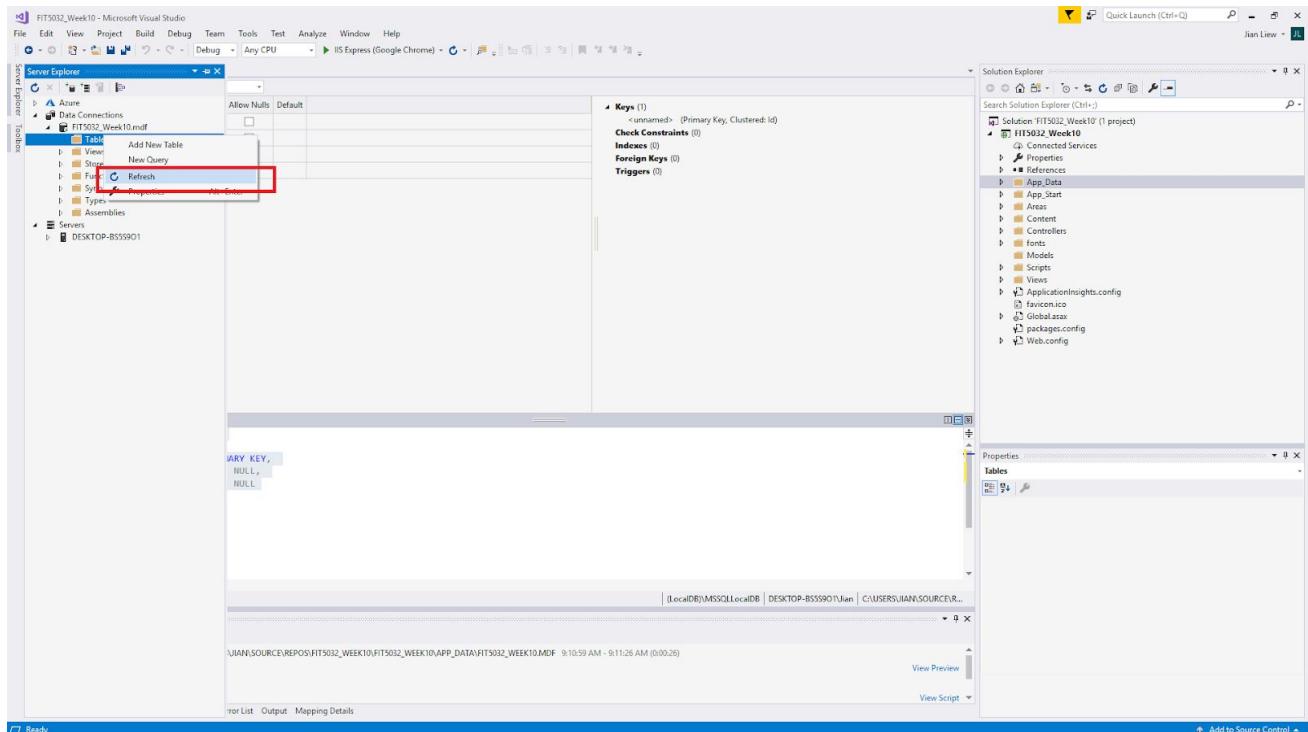


You can use the GUI to add new tables and update the database.



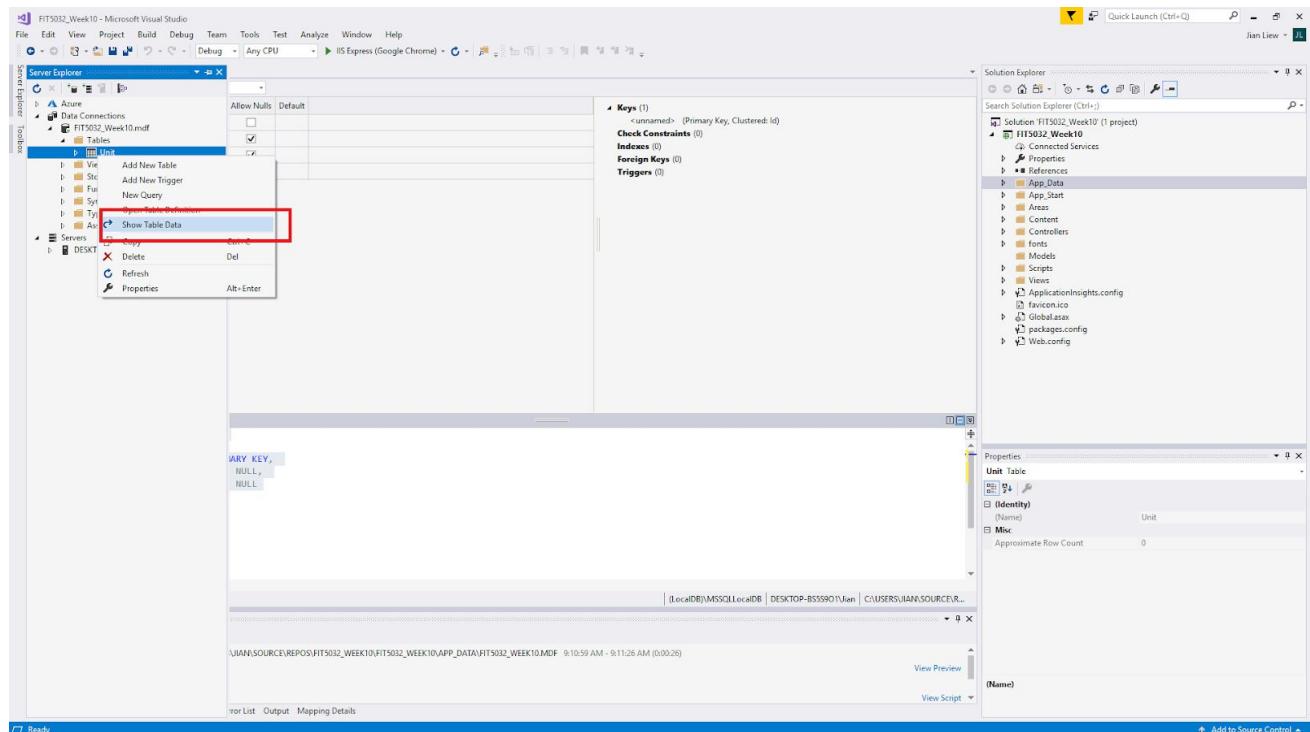


After that, the tables should be seen in the database. (After you hit the Refresh button)

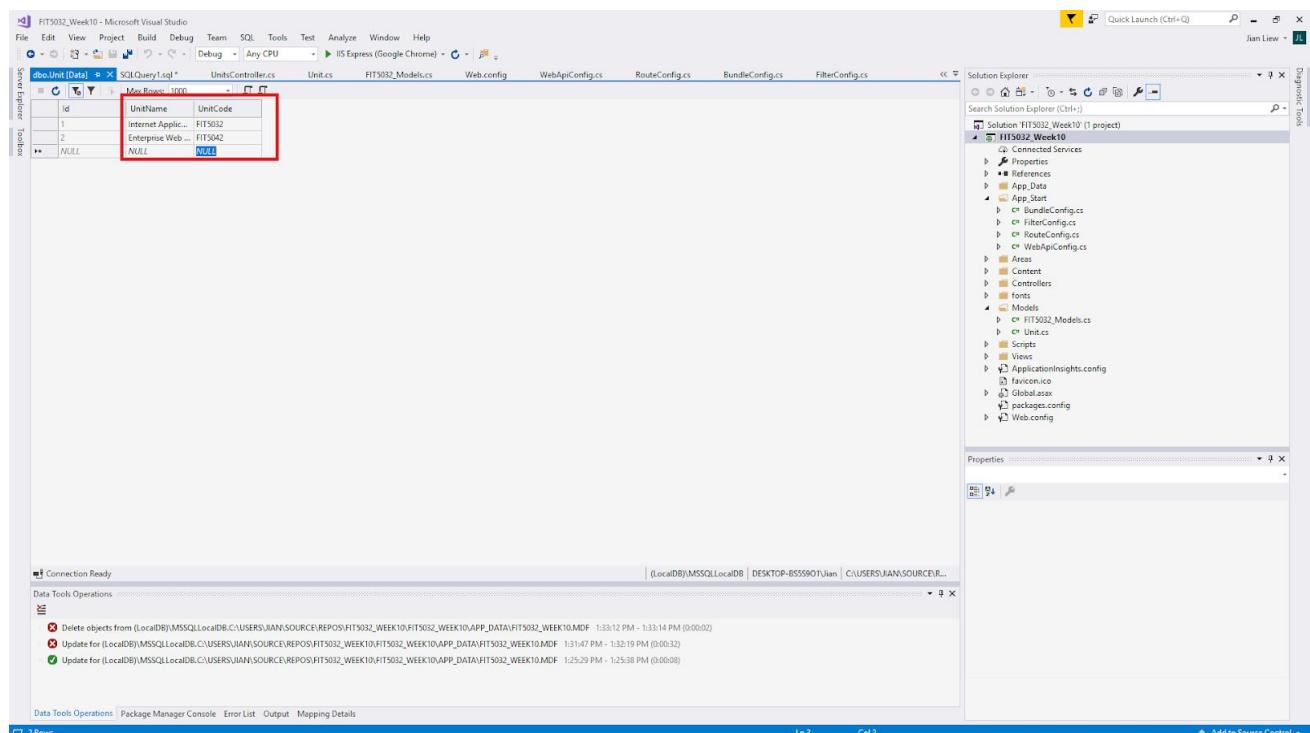


Step 7

I will add 2 rows of information into my table.



The screenshot shows the Microsoft Visual Studio interface with the 'Server Explorer' and 'Solution Explorer' panes. In the 'Server Explorer', under 'Data Connections' and 'FIT5032_Week10.mdf', the 'Tables' node is expanded, showing the 'Unit' table. A context menu is open over the 'Unit' table, with the 'Show Table Data' option highlighted and selected. The 'Properties' pane on the right shows the table's properties, including its primary key 'UnitId'.

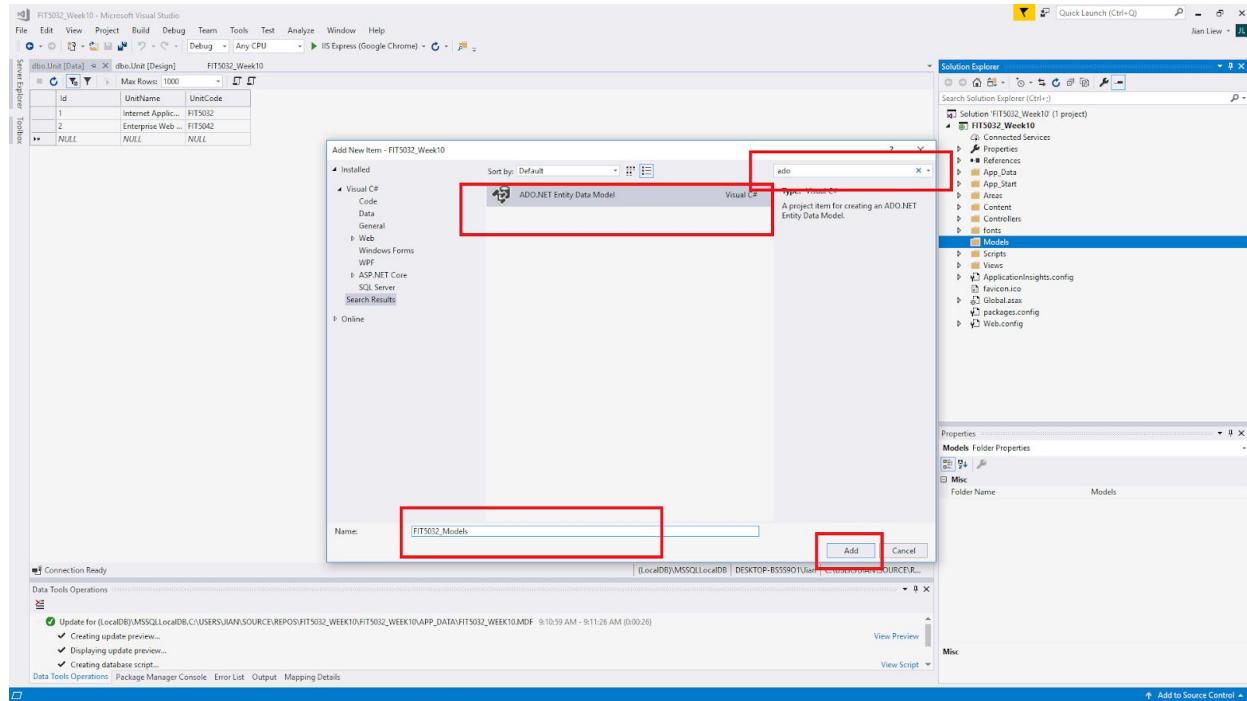


The screenshot shows the Microsoft Visual Studio interface with the 'Server Explorer' and 'Solution Explorer' panes. In the 'Server Explorer', under 'Data Connections' and 'FIT5032_Week10.mdf', the 'Tables' node is expanded, showing the 'Unit' table. The table now contains two rows of data:

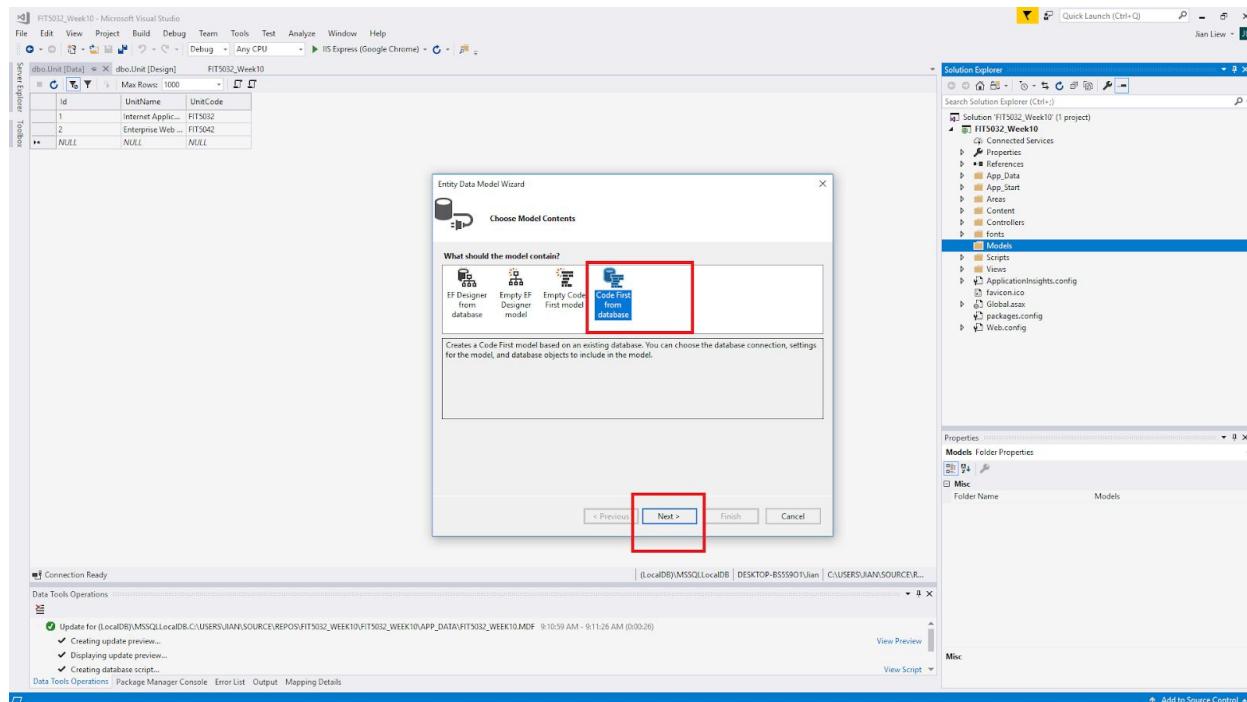
	UnitName	UnitCode
1	Internet Application	FIT5032
2	Enterprise Web Application	FIT5042

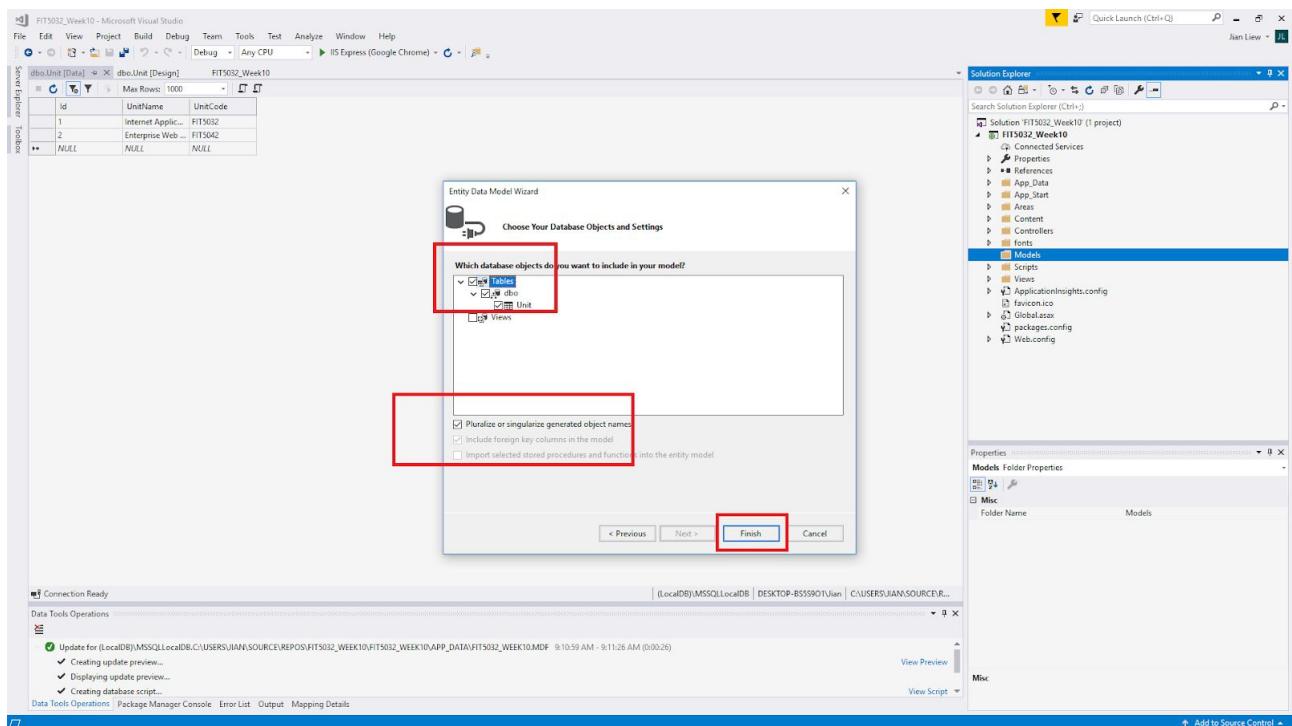
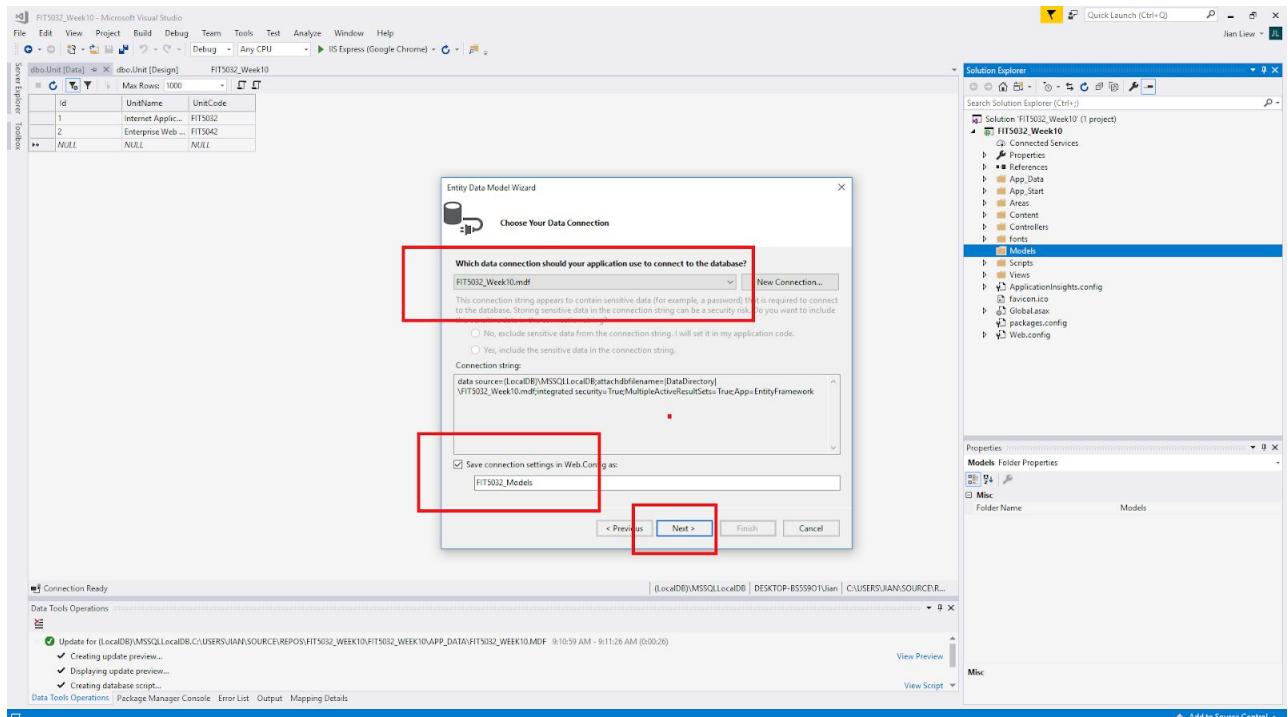
Step 8

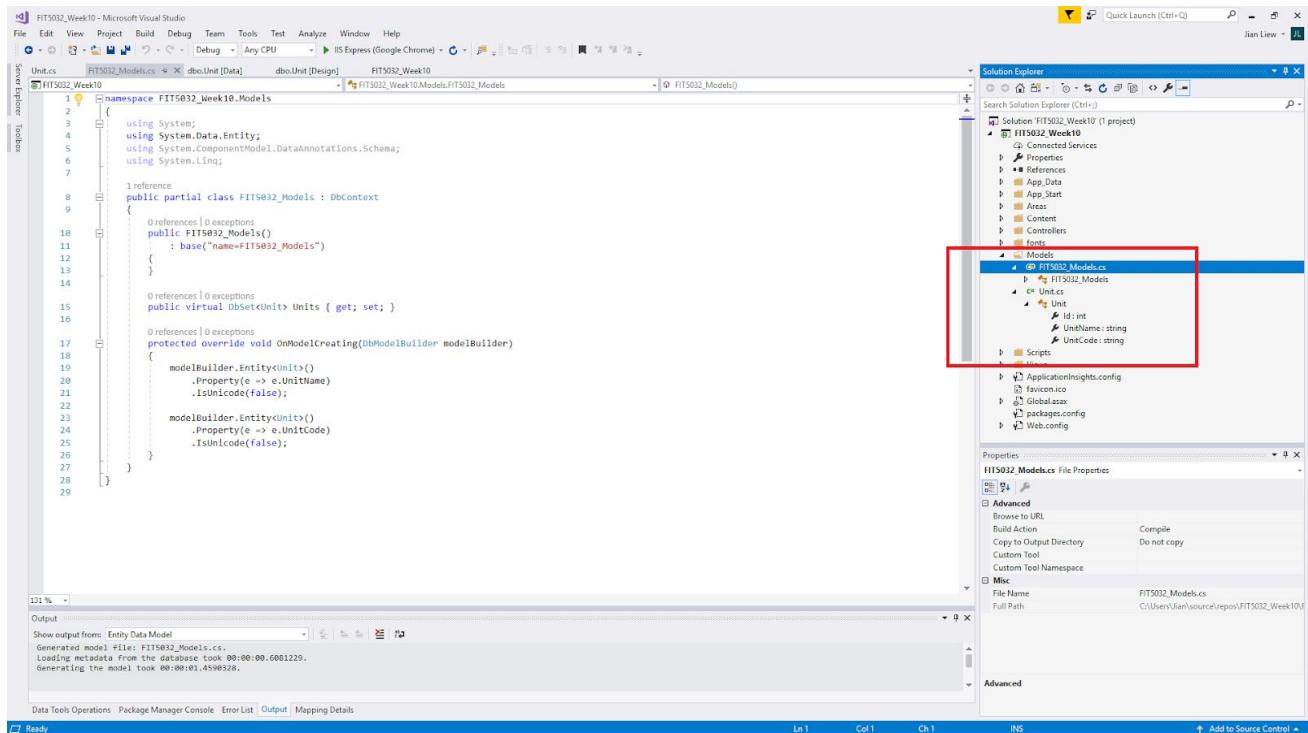
Now I will use the Database First approach to generate my models. (Right click on Models and Add New Item)



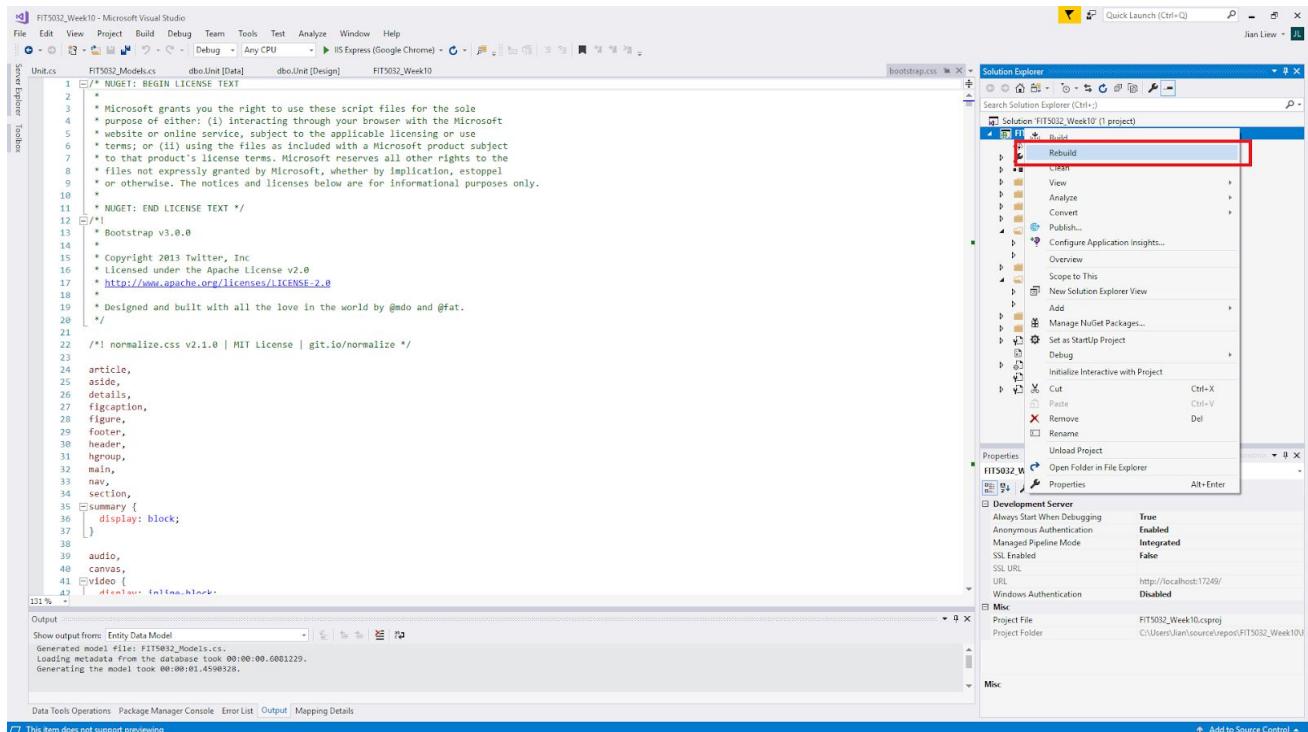
We can go either using the EF designer or the Code First when we generate the tables from an existing database. I will go with Code First as it is cleaner.





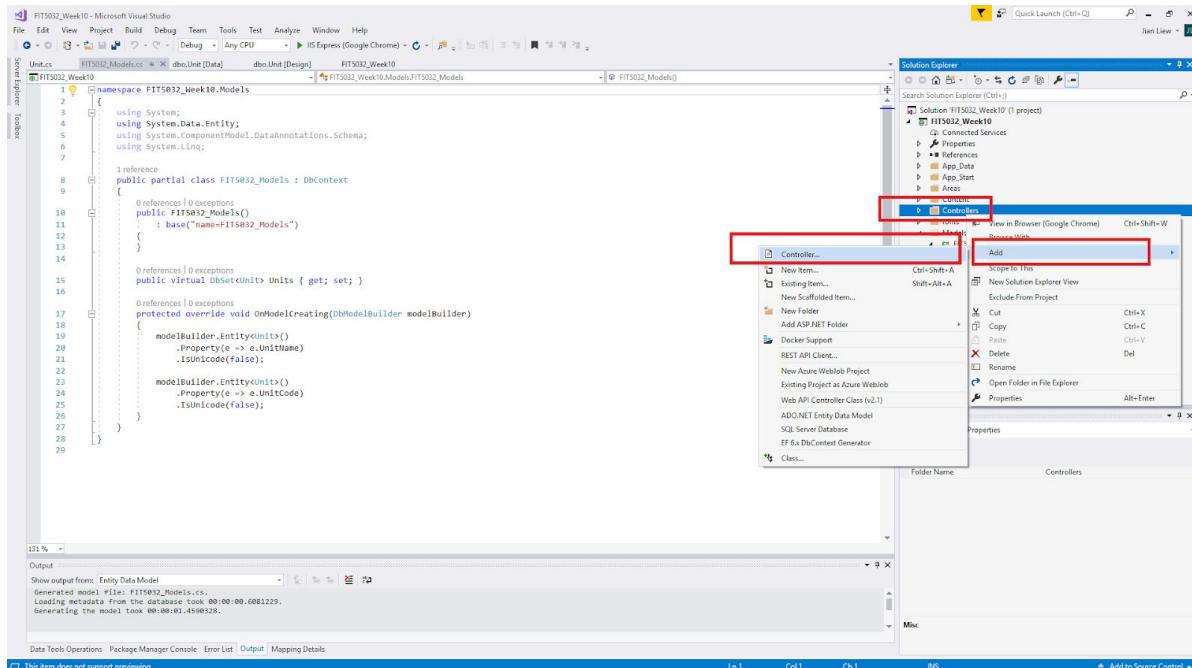


After that, I will rebuild the project. (If you do not do this, you will have troubles in the next step)

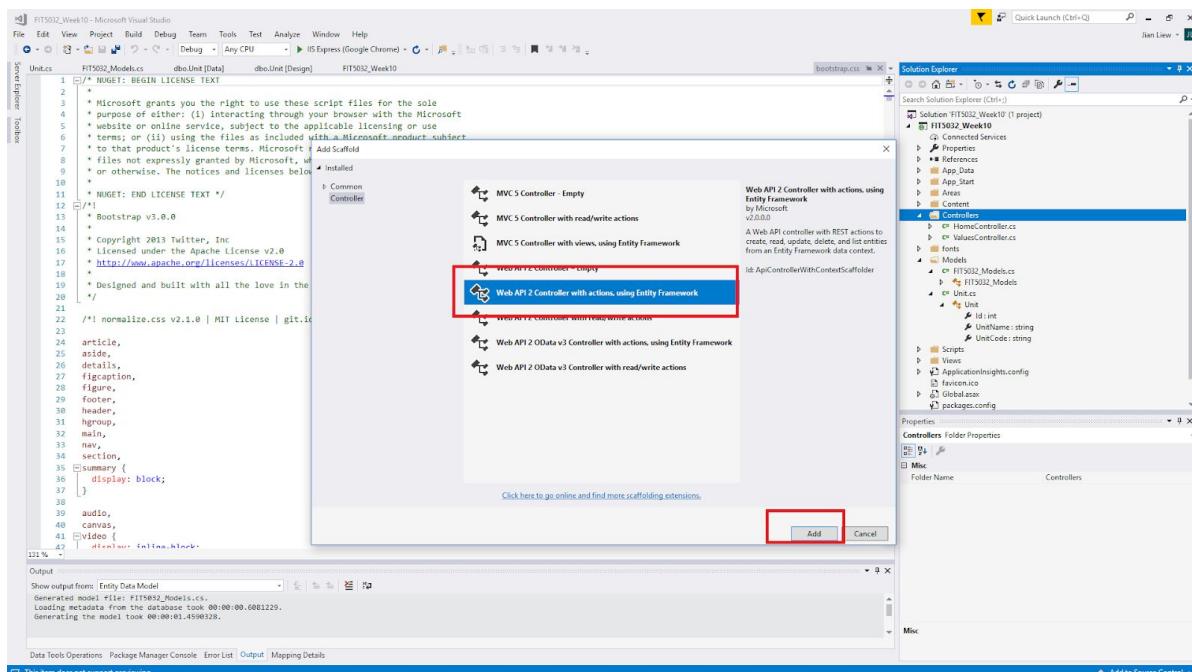


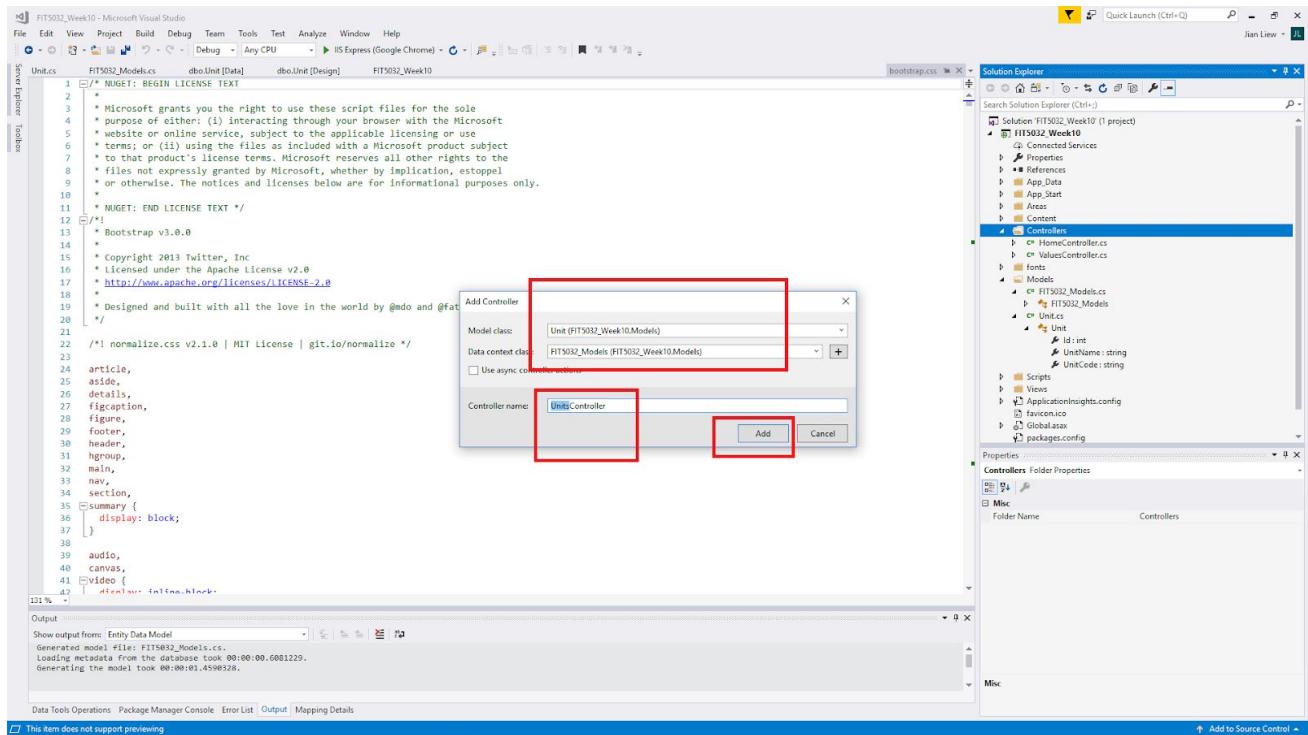
Step 8

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.

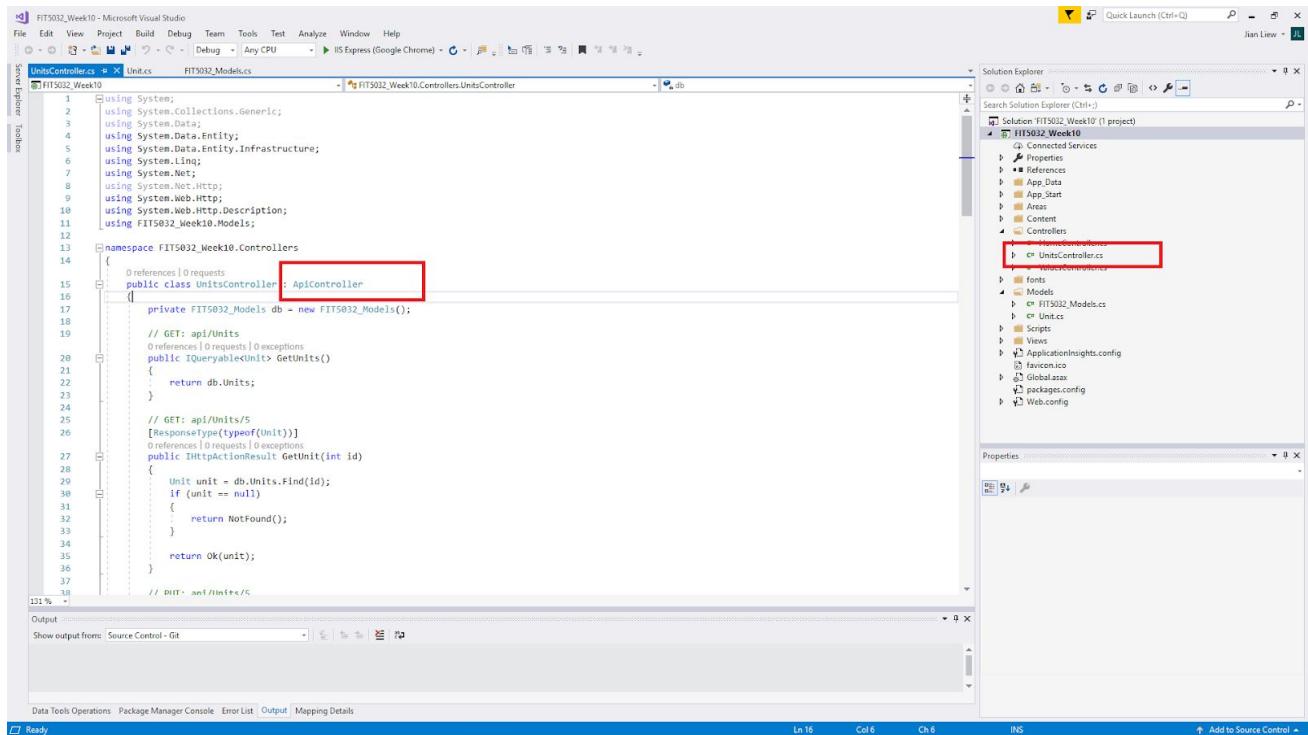


I will use the Web API 2 Controller option. (Refer screenshot for more details)





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.



This XML file does not appear to have any style information associated with it. The document tree is shown below:

```
<ArrayOfUnit xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/FIT5032_Week10.Models">
  <Unit>
    <Id>1</Id>
    <UnitCode>FIT5032</UnitCode>
    <UnitName>Internet Applications Development</UnitName>
  </Unit>
  <Unit>
    <Id>2</Id>
    <UnitCode>FIT5042</UnitCode>
    <UnitName>Enterprise Web Application</UnitName>
  </Unit>
</ArrayOfUnit>
```



This XML file does not appear to have any style information associated with it. The document tree is shown below:

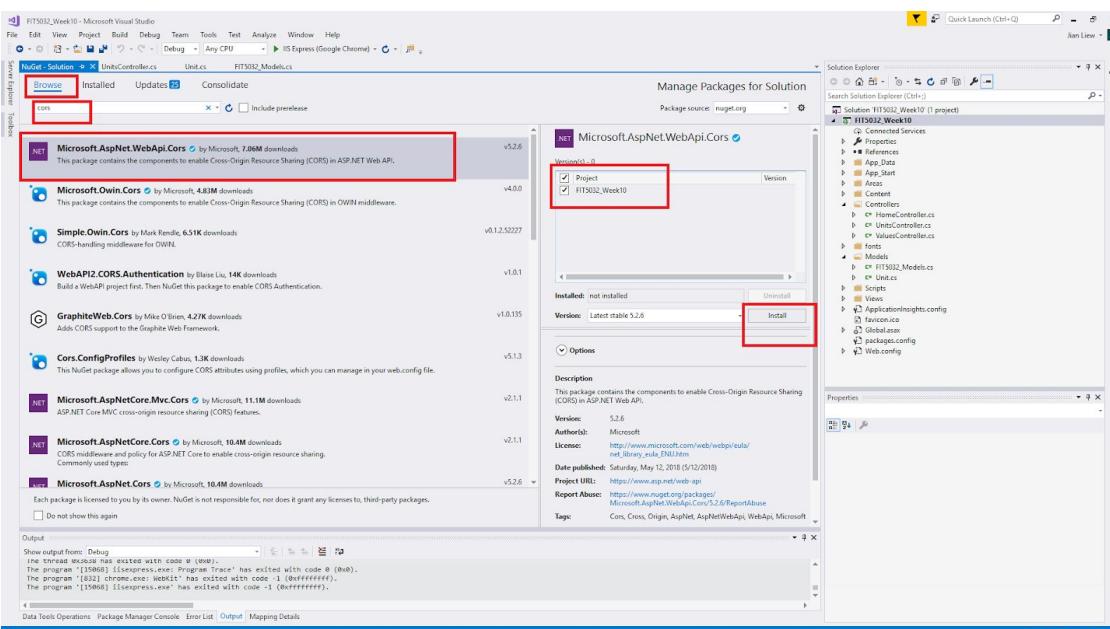
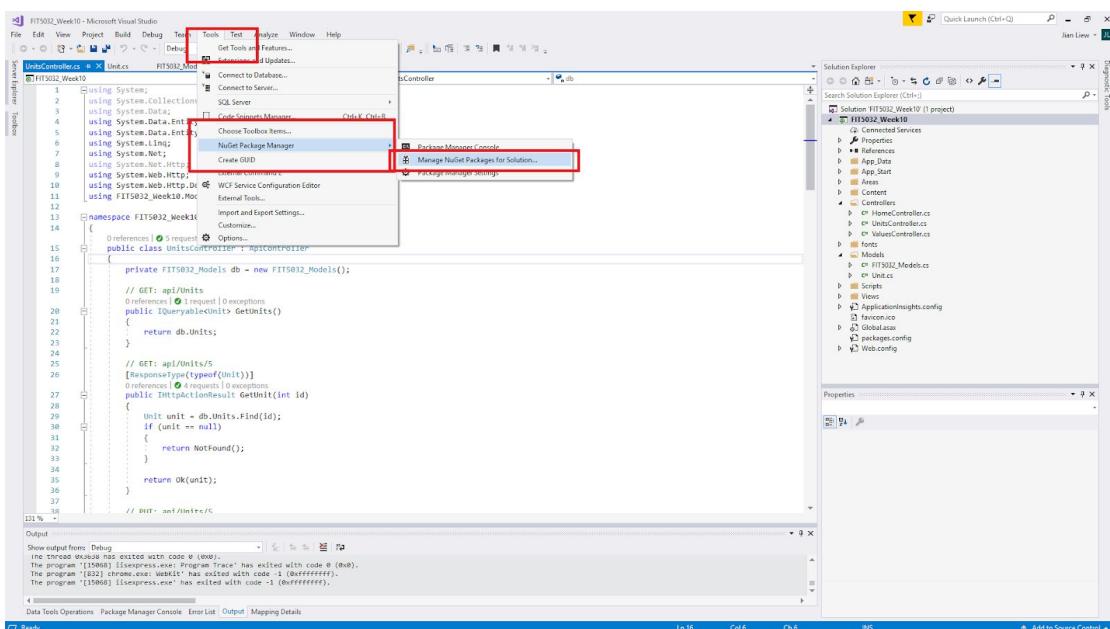
```
<Unit xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/FIT5032_Week10.Models">
  <Id>1</Id>
  <UnitCode>FIT5032</UnitCode>
  <UnitName>Internet Applications Development</UnitName>
</Unit>
```

Step 10

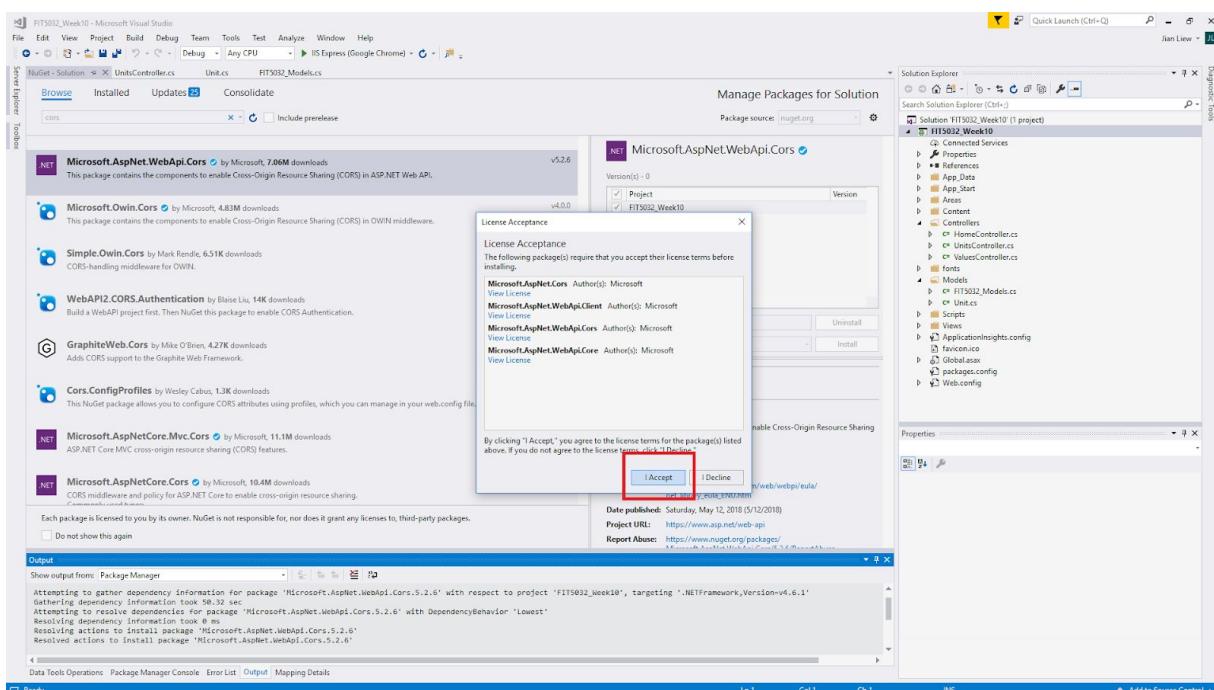
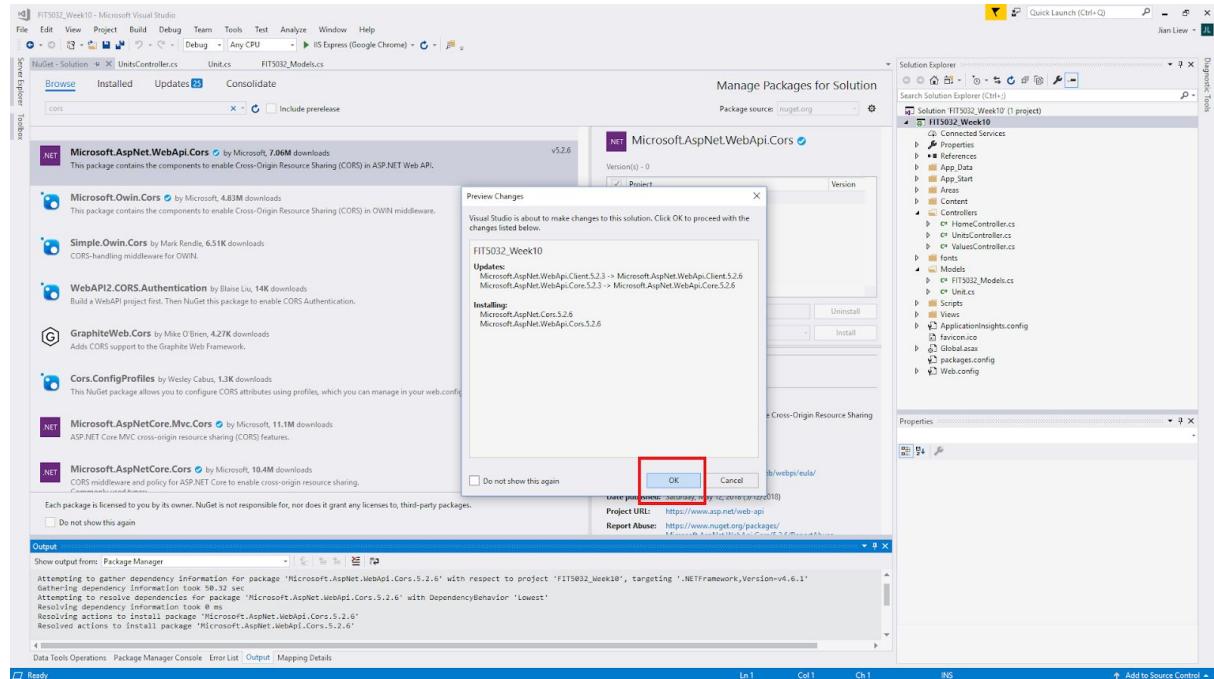
You can also install Postman to test out your end points. (Postman is not installed in the Monash lab computers). After you have created and test your endpoints, you should allow other consumers to consume the resource you have created.

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.

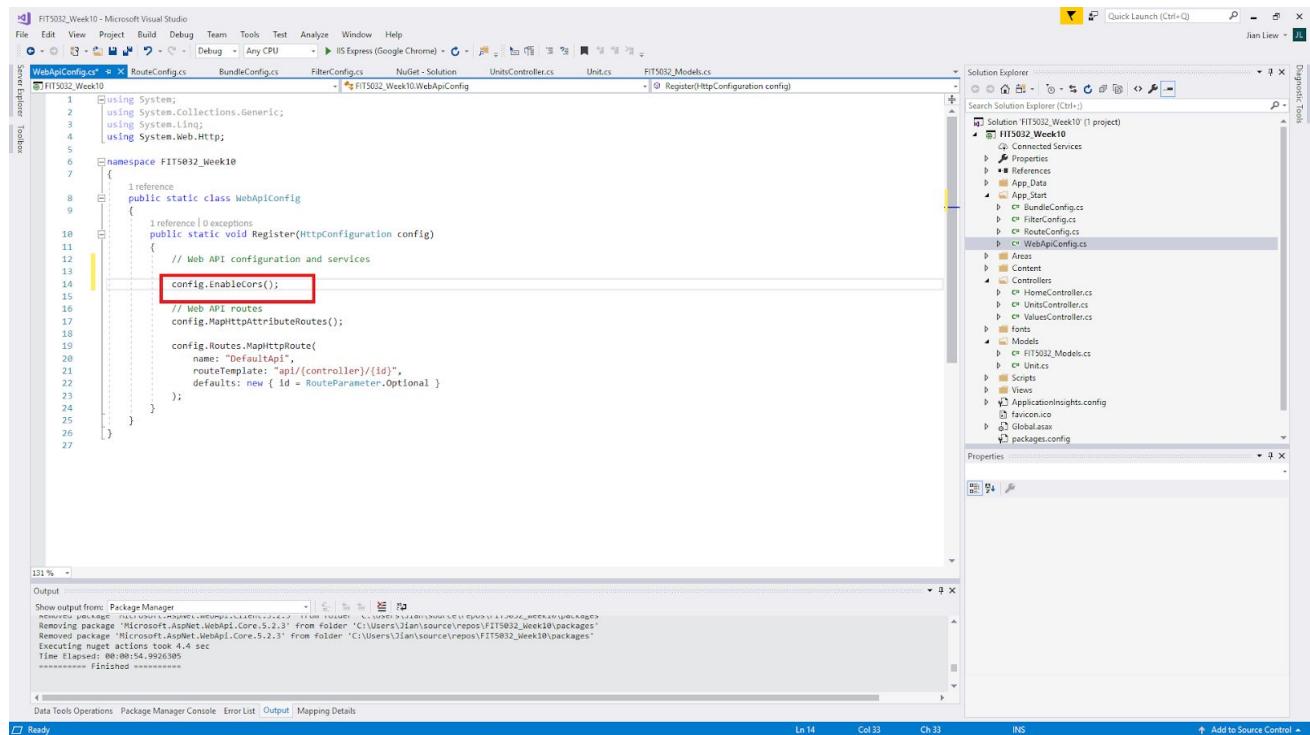


Step 11



Step 12

I will add the EnableCors line at the WebApiConfig.cs

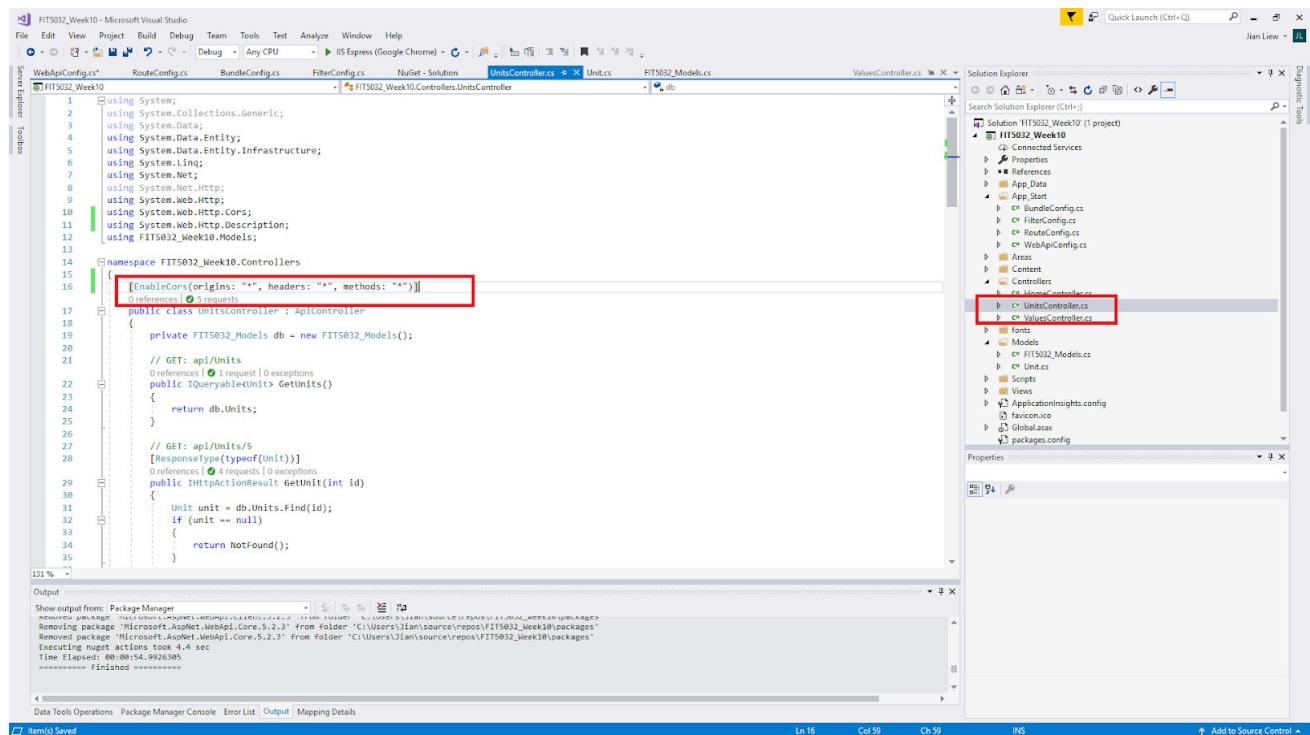


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web.Http;
5
6  namespace FIT5032_Week10
7  {
8      public static class WebApiConfig
9      {
10         public static void Register(HttpConfiguration config)
11         {
12             // Web API configuration and services
13
14             config.EnableCors();
15
16             // Web API routes
17             config.MapHttpAttributeRoutes();
18
19             config.Routes.MapHttpRoute(
20                 name: "DefaultApi",
21                 routeTemplate: "api/{controller}/{id}",
22                 defaults: new { id = RouteParameter.Optional }
23             );
24         }
25     }
26 }

```

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



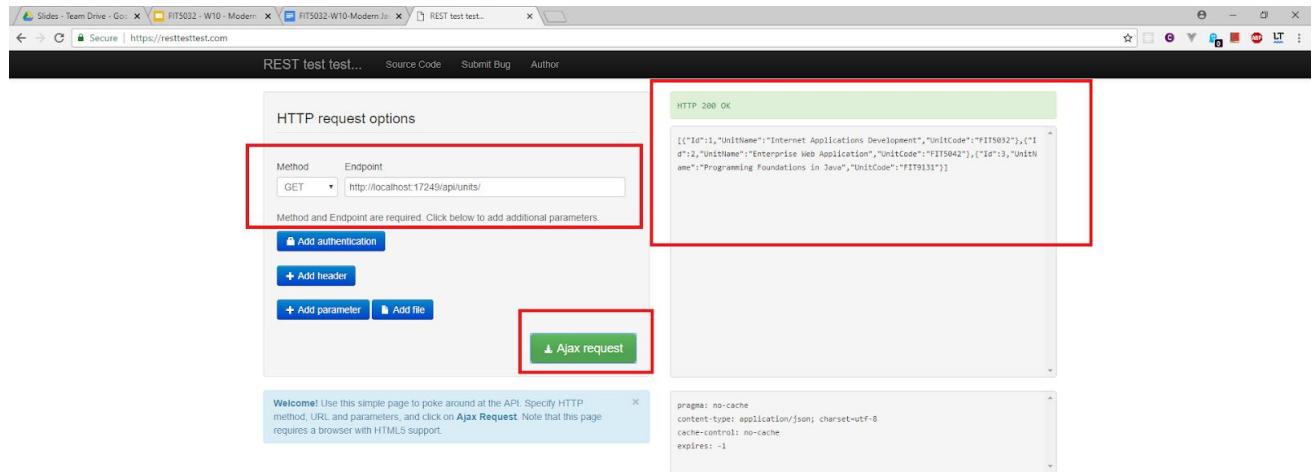
```

1  using System;
2  using System.Collections.Generic;
3  using System.Data.Entity;
4  using System.Data.Entity.Infrastructure;
5  using System.Linq;
6  using System.Net;
7  using System.Web.Http;
8  using System.Web.Http.Cors;
9  using System.Web.Http.Description;
10
11  using FIT5032_Week10.Models;
12
13
14  namespace FIT5032_Week10.Controllers
15  {
16      [EnableCors(origins: "*", headers: "*", methods: "*")]
17      public class UnitsController : ApiController
18      {
19          private FIT5032_Models db = new FIT5032_Models();
20
21          // GET: api/Units
22          [HttpGet]
23          public IQueryable<Unit> GetUnits()
24          {
25              return db.Units;
26          }
27
28          // GET: api/Units/
29          [ResponseType(typeof(Unit))]
30          [HttpGet]
31          public IHttpActionResult GetUnit(int id)
32          {
33              Unit unit = db.Units.Find(id);
34              if (unit == null)
35              {
36                  return NotFound();
37              }
38          }
39      }
40  }

```

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.

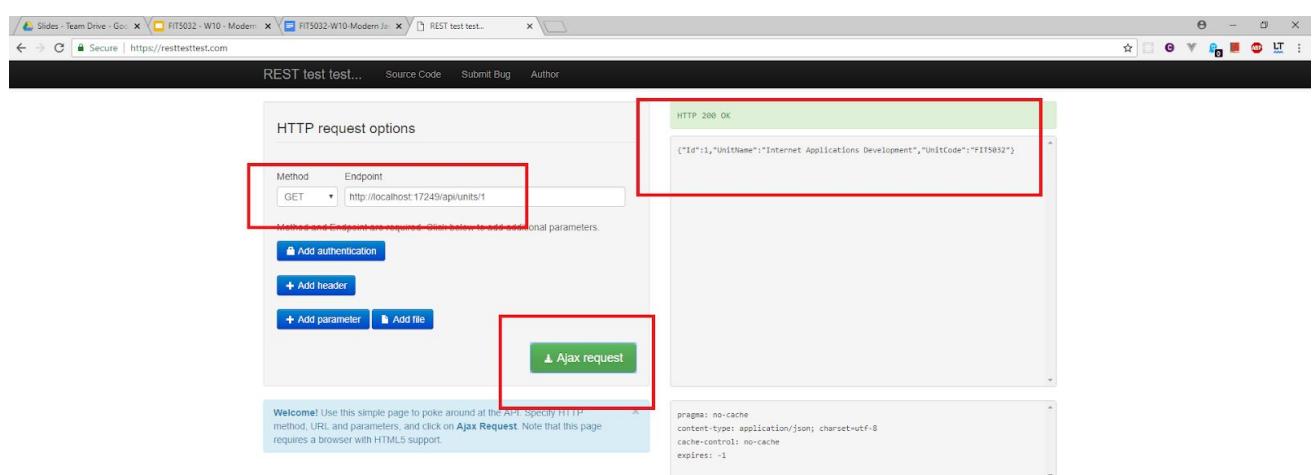


The screenshot shows a browser window with the URL <https://resttestest.com>. The page title is "REST test test...". On the left, there is a "HTTP request options" form with a red box around the "Method" dropdown set to "GET" and the "Endpoint" input field containing "http://localhost:17249/api/units/". Below the form are buttons for "Add authentication", "+ Add header", "+ Add parameter", and "+ Add file". A green "Ajax request" button is highlighted with a red box. On the right, a red box highlights the response area which shows "HTTP 200 OK" and a JSON array of unit records:

```
[{"Id":1,"UnitName":"Internet Applications Development","UnitCode":"FIT5832"}, {"Id":2,"UnitName":"Enterprise Web Application","UnitCode":"FIT5842"}, {"Id":3,"UnitName":"Programming Foundations in Java","UnitCode":"FIT9131"}]
```

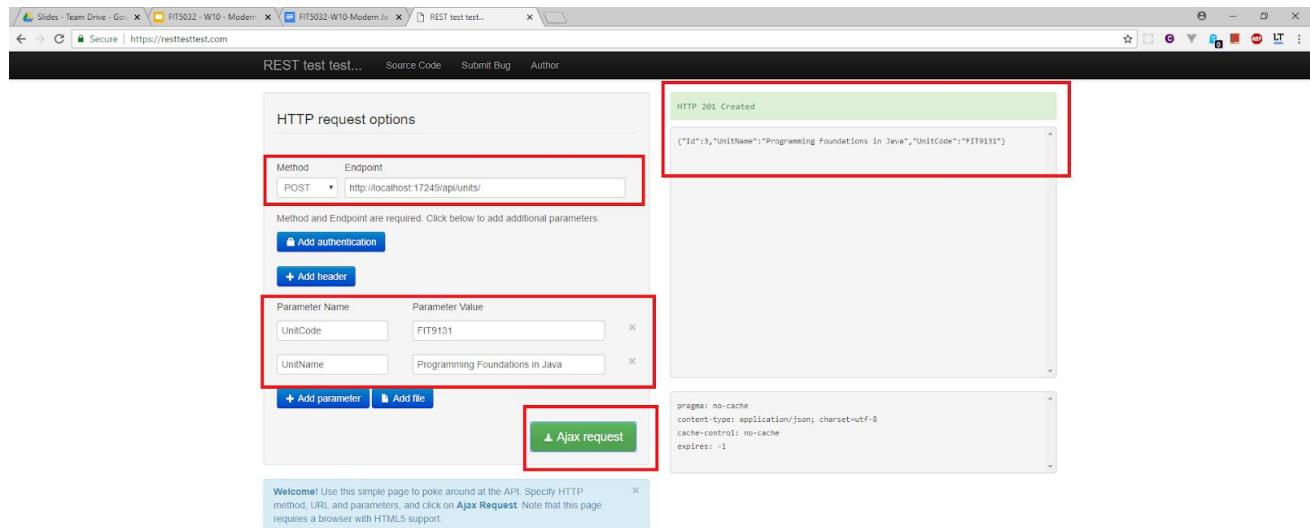
Below the response, the raw HTTP headers are shown:

```
pragma: no-cache
content-type: application/json; charset=utf-8
cache-control: no-cache
expires: -1
```



This screenshot is identical to the one above, showing the same "HTTP request options" form and the same JSON response. The "Method" is still "GET" and the "Endpoint" is "http://localhost:17249/api/units/". The "Ajax request" button is again highlighted with a red box. The response area shows the same array of unit records and the same raw HTTP headers.

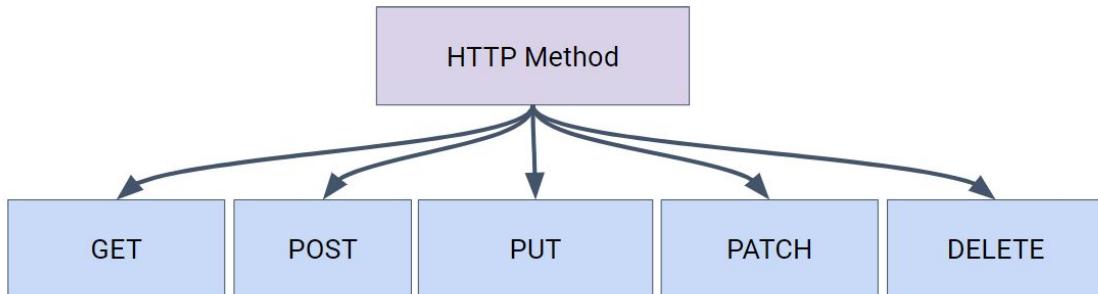
You can also try the POST request.



The screenshot shows a REST API testing interface. On the left, there's a form for "HTTP request options" with "Method" set to "POST" and "Endpoint" set to "http://localhost:17249/api/units/". Below this, there's a table for "Parameter Name" and "Parameter Value" with entries for "UnitCode" (value: FIT9131) and "UnitName" (value: Programming Foundations in Java). At the bottom of the form is a green button labeled "Ajax request". To the right of the form, a red box highlights the response area which shows "HTTP 201 Created" and the JSON response: {"Id":1,"UnitName":"Programming Foundations in Java","UnitCode":"FIT9131"}. Below the response, there's some header information: pragma: no-cache, content-type: application/json; charset=utf-8, cache-control: no-cache, and expires: -1.

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 requests. We have tested out the GET and POST so far. Please try the other methods and see how they work.

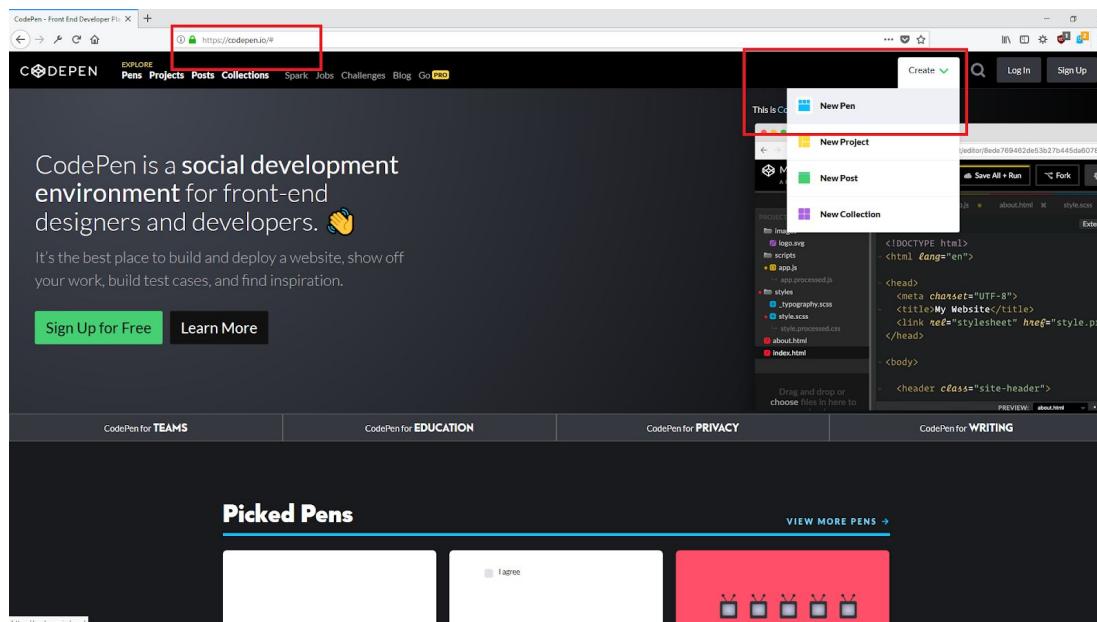
Creating a Simple AngularJS Application

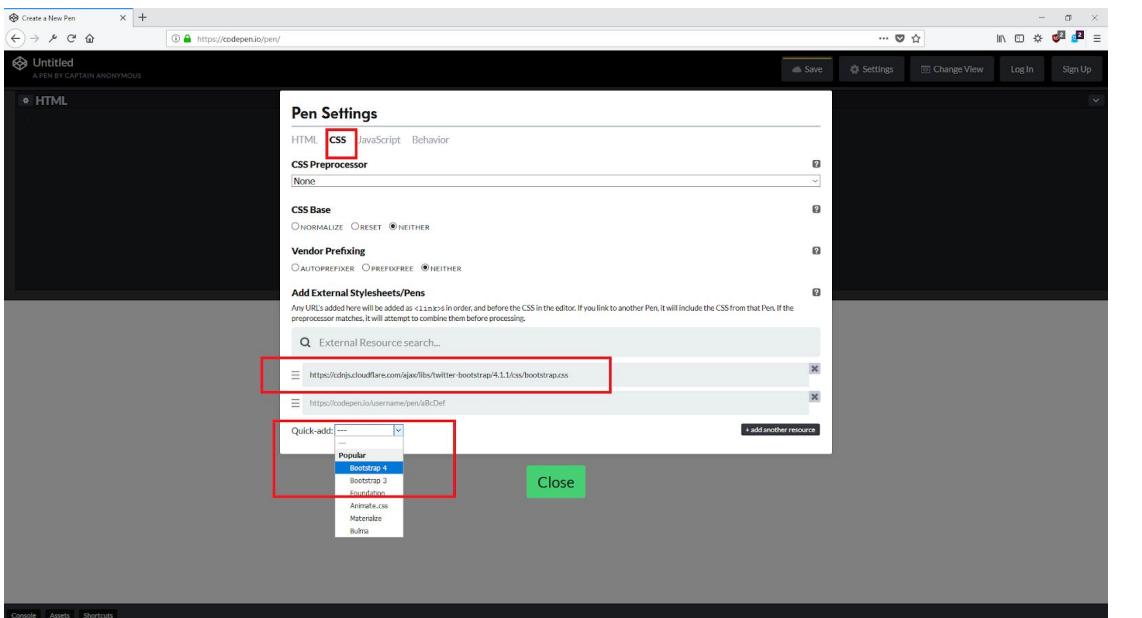
Once we have created the RESTful endpoints, we will now create a simple JavaScript application. The reason we went about creating a RESTful service first is because it is a very common scenario to consume a RESTful service or WEB API. Most online tutorials tend to "over-engineer" this however, it is 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)

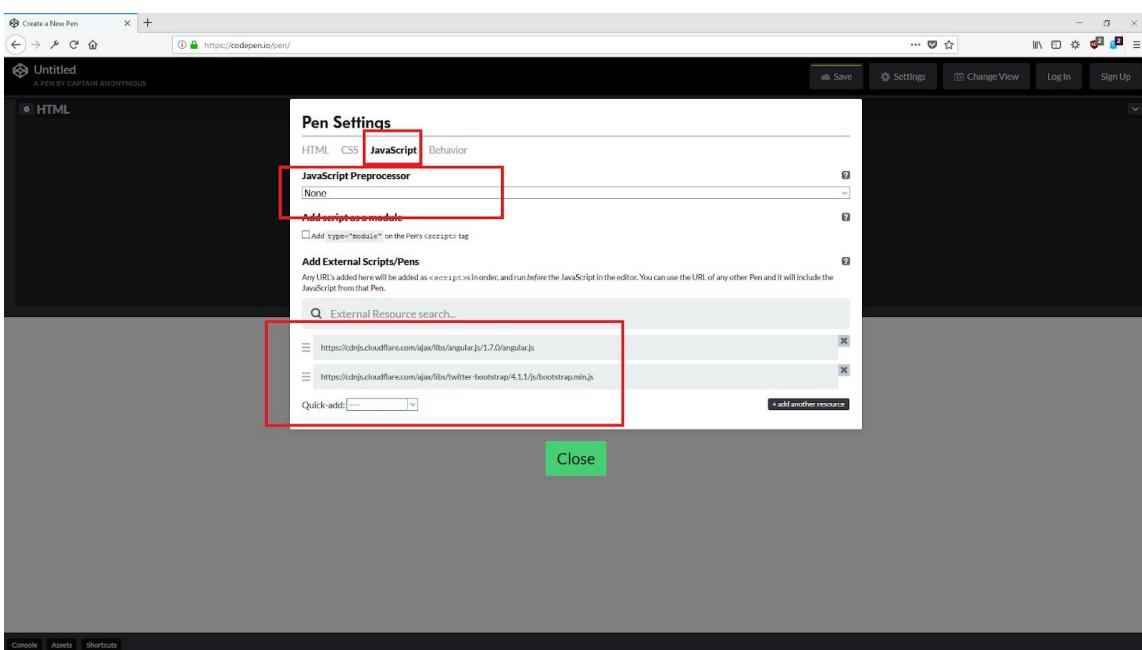
The final product can be found here - <https://codepen.io/jianloong/pen/bjpjoK>

Following the following steps are highly optional





The screenshot shows the CodePen interface with the 'CSS' tab selected in the 'Pen Settings' section. The 'Add External Stylesheets/Pens' input field contains two URLs: 'https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.1/css/bootstrap.css' and 'https://codepen.io/username/pen/aBcDef'. A dropdown menu titled 'Quick-add:' is open, showing a list of popular CSS frameworks, with 'Bootstrap 4' highlighted. A red box highlights the 'CSS' tab in the settings, and another red box highlights the 'External Resource search...' input field.



The screenshot shows the CodePen interface with the 'JavaScript' tab selected in the 'Pen Settings' section. The 'Add External Scripts/Pens' input field contains two URLs: 'https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.7.0/angular.js' and 'https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.1.1/js/bootstrap.min.js'. A red box highlights the 'JavaScript' tab in the settings, and another red box highlights the 'External Resource search...' input field.

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>
```

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)