5.5 URI Customization and Entity Framework Integration



This section will guide you to:

* Integrate and customize URI and Entity framework

This guide has ten subsections, namely:

5.5.1 Creating a database and populate it with data

5.5.2 Creating an ASP.NET MVC Web API project

5.5.3 Installing Entity Framework using NuGet

5.5.4 Creating a Student Entity Data Model using EF

5.5.5 Creating a StudentViewModel

5.5.6 Creating a Student Controller

5.5.7 Adding a custom route to WebApiConfig

5.5.8 Building the project

5.5.9 Publishing and running the project

5.5.10 Pushing the code to your GitHub repositories

**Step 5.5.1:** Creating a database and populate it with data

* MSSQL is already installed in your practice lab. Refer **DotNet Lab guide: Phase 3** for more information.
* Open SQL Server Management Studio. In the login screen, make a note of the **Server Name** value as you will need to put in your ASP.NET application.
* In **Object Explorer**,right click **Databases** and choose **New Database**.
* Enter **Database name** as **School1** and click **Ok**.
* In **Object Explorer**,expand **School1->Tables.** Right click **Tables** and choose **New->Table**.
* For the first row enter ID as **Column Name**,int as **Data Type**, anduncheck **Allow Nulls**.
* In **Column Properties,** go to **Identity Specification** and expand it. Double click **Is Identity** to make it **Yes**.
* For the next row enter Name as **Column Name**,varchar(100) as **Data Type**, anduncheck **Allow Nulls**.
* For the next row enter Address as **Column Name**,varchar(100) as **Data Type**,and uncheck **Allow Nulls**.
* For the next row enter Email as **Column Name**,varchar(75) as **Data Type**, anduncheck **Allow Nulls**.
* For the next row enter Class as **Column Name**,varchar(5) as **Data Type**, anduncheck **Allow Nulls**.
* Click on the x icon to close the table grid tab. Click **Yes** on the save dialog.
* For **Enter a name** put Student and press **Ok**.
* In **Object Explorer**,expand **School1->Tables->Student.** Right click **Tables** and choose **Edit Top 200 Rows**.
* Add in a few rows of data with random values.
* Close the Management Studio.

**Step 5.5.2:** Creating an ASP.NET MVC Web API project

* From the top menu in the Visual Studio, select **File->New->Project**.
* In **Create A New Project** screen, select **ASP.NET Web Application (.NET Framework)** from the list of available project types and click on **Next**.
* Enter **Project Name** as **Phase3Section5.9** and click on **Create**.
* From the list of project sub-types, choose **Web API** and uncheck **Configure for HTTPS.** Click on **Create**.
* This will create the files for an ASP.NET MVC Web API project.

**Step 5.5.3:** Installing Entity Framework using NuGet

* In the top menu, go to **Tools->Nuget Package Manager->Package Manager Console**.
* In the console, type install-package entityframework and press Enter.

**Step 5.5.4:** Create a Student Entity Data Model using EF

* In the **Solution Explorer**,right click **Phase3Section5.7** and choose **Add->New Item**.
* Under **Visual C#,** select **Data** and choose **ADO.NET Entity Data Model.** Enter **Name** as Student and click **Add**.
* In the next screen, click on **EF Designer from Database** and click **Next**.
* In the next screen, click on **New Connection**.
* This will open a popup window. In the **Server Name,** put the server name from the SQL Management Studio login screen.
* In **Select or enter a Database name,** choose **School1** and click **Ok**.
* This will close the popup window. In the current window, make sure **Save Connection settings in web.config** is checked and make a note of the value in the textbox eg.School1Entities.
* Click on **Next**.
* In the next screen, expand **Tables->dbo->Student** and check **Student.** Click **Finish**.
* This will create all the entity files for the Student table.

**Step 5.5.5:** Creating a StudentViewModel

* In the **Solution Explorer**,right click the **Models** folder and choose **Add->Class**.
* Enter **Name** as StudentViewModel.cs and click **Add**.
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Web;

**namespace** Phase3Section5.\_9.Models

{

**public** **class** StudentViewModel

{

**public** **int** Id { **get**; **set**; }

**public** **string** Name { **get**; **set**; }

**public** **string** Address{ **get**; **set**; }

**public** **string** Class { **get**; **set**; }

**public** **string** Email { **get**; **set**; }

}

}

**Step 5.5.6:** Creating a Student Controller

* In the **Solution Explorer**,right click the **Controllers** folder and choose **Add->Controller**.
* From the list of controller types, choose **Web API 2.0 Controller – Empty** and click **Add**.
* Put **Controller Name** as StudentController.
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Net;

**using** System.Net.Http;

**using** System.Web.Http;

**using** Phase3Section5.\_7.Models;

**namespace** Phase3Section5.\_9.Controllers

{

**public** **class** StudentController : ApiController

{

[Route("api/student/names")]

**public** IHttpActionResult Get()

{

IList<StudentViewModel> students = **null**;

**using** (**var** ctx = **new** School1Entities1())

{

students = ctx.Students

.Select(s => **new** StudentViewModel()

{

Name = s.Name,

}).ToList<StudentViewModel>();

}

**if** (students.Count == 0)

{

**return** NotFound();

}

**return** Ok(students);

}

**public** IHttpActionResult GetAllStudents()

{

IList<StudentViewModel> students = **null**;

**using** (**var** ctx = **new** School1Entities1())

{

students = ctx.Students

.Select(s => **new** StudentViewModel()

{

Id = s.ID,

Name = s.Name,

Email = s.Email,

Class = s.Class,

Address = s.Address

}).ToList<StudentViewModel>();

}

**if** (students.Count == 0)

{

**return** NotFound();

}

**return** Ok(students);

}

**public** IHttpActionResult PostNewStudent(StudentViewModel student)

{

**if** (!ModelState.IsValid)

**return** BadRequest("Invalid data.");

**using** (**var** ctx = **new** School1Entities1())

{

ctx.Students.Add(**new** Student()

{

Name = student.Name,

Email= student.Email,

Address = student.Address,

Class = student.Class

});

ctx.SaveChanges();

}

**return** Ok();

}

**public** IHttpActionResult Put(StudentViewModel student)

{

**if** (!ModelState.IsValid)

**return** BadRequest("Not a valid model");

**using** (**var** ctx = **new** School1Entities1())

{

**var** existingStudent = ctx.Students.Where(s => s.ID == student.Id)

.FirstOrDefault<Student>();

**if** (existingStudent != **null**)

{

existingStudent.Name = student.Name;

existingStudent.Address = student.Address;

existingStudent.Email = student.Email;

existingStudent.Class = student.Class;

ctx.SaveChanges();

}

**else**

{

**return** NotFound();

}

}

**return** Ok();

}

[HttpDelete]

**public** IHttpActionResult Delete(**int** id)

{

**if** (id <= 0)

**return** BadRequest("Not a valid student id");

**using** (**var** ctx = **new** School1Entities1())

{

**var** student = ctx.Students

.Where(s => s.ID == id)

.FirstOrDefault();

ctx.Entry(student).State = System.Data.Entity.EntityState.Deleted;

ctx.SaveChanges();

}

**return** Ok();

}

}

}

**Step 5.5.7:** Adding a custom route to WebApiConfig

* In **Solution Explorer**,expand **App\_Start** and double click WebApiConfig.cs.
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Web.Http;

**namespace** Phase3Section5.\_7

{

**public** **static** **class** WebApiConfig

{

**public** **static** **void** Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

// school route

config.Routes.MapHttpRoute(

name: "Custom",

routeTemplate: "testapi/{controller}/{id}",

defaults: **new** { id = RouteParameter.Optional }

);

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: **new** { id = RouteParameter.Optional }

);

}

}

}

**Step 5.5.8:** Building the project

* From the top menu, choose **Build->Build Solution**.
* If any compile errors are shown, fix them as required.

**Step 5.5.9:** Publishing and running the project

* From the top menu, select **Debug->Start Without Debugging**.
* This will execute the program in the default browser.
* Change the url to [http://localhost:xxxx/api/student](about:blank) to display the student list.
* Use the url to [http://localhost:xxxx/api/student](about:blank)/names to display only student names in the list.
* Use the url to [http://localhost:xxxx/testapi/student](about:blank) to display the student list using an alternate route.
* Do an HTTP POST to [http://localhost:xxxx/api/student](about:blank) with the parameters: name=xxx&address=xxxx&email=xxx&class=xxx for adding a new student.
* Do an HTTP PUT to [http://localhost:xxxx/api/student](about:blank) with the parameters: id=xxx&name=xxx&address=xxxx&email=xxx&class=xxx for updating an existing student based on id.
* Do an HTTP DELETE to [http://localhost:xxxx/api/student](about:blank) with the parameters: id=xxx to delete a student based on id.

**Step 5.5.10:** Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit -m “Changes have been committed.”

Push the files to the folder you created initially using the following command:

git push -u origin master