.NET Framework: Developing Modern Web Apps with ASP.NET MVC – Workshop*PLUS*

Module 02: Models

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.0

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2017 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2017 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Microsoft, SQL Server, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 2: Creating Model 1](#_Toc478756016)

[Introduction 1](#_Toc478756017)

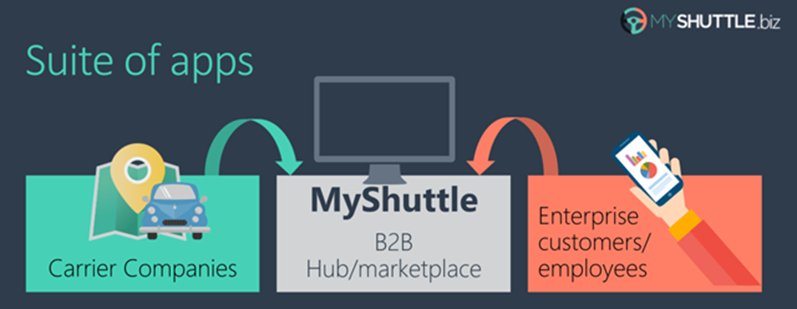
[Exercise 1: Create MyShuttle Model 5](#_Toc478756018)

[Exercise 2: Implement MyShuttle Data CRUD Logic 10](#_Toc478756019)

# Lab 2: Creating Model

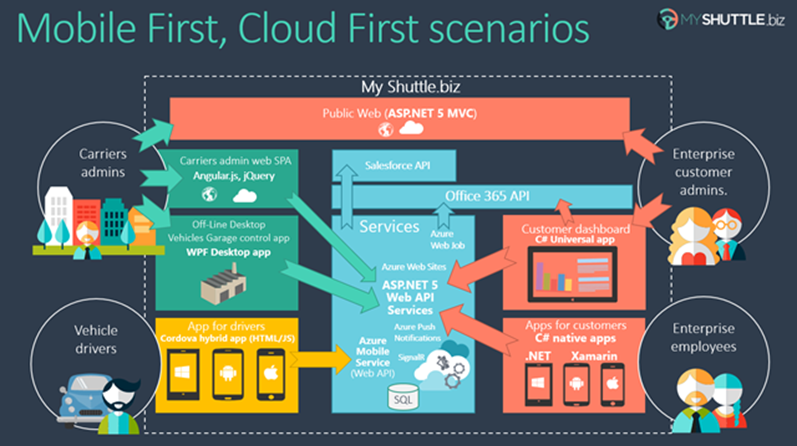
Introduction

MyShuttle is a B2B highly scalable multi-tenant software as a service (SaaS) solution that targets corporate scenarios in which carrier companies offer transport services to enterprise customers.



This multi-tenant SaaS system would allow any number of carrier companies who must be syndicated with the system, to provide their services (cabs/shuttles) directly to any number of customer-enterprises/companies who would also be registered in the MyShuttle.biz system. The final outcome is that any employee in those customer companies would be able to request a cab/shuttle at any time in any place/city without worrying about how to pay. Everything would take place underneath between their company and the carrier company for that ride.

The global scenario is described in the following image:

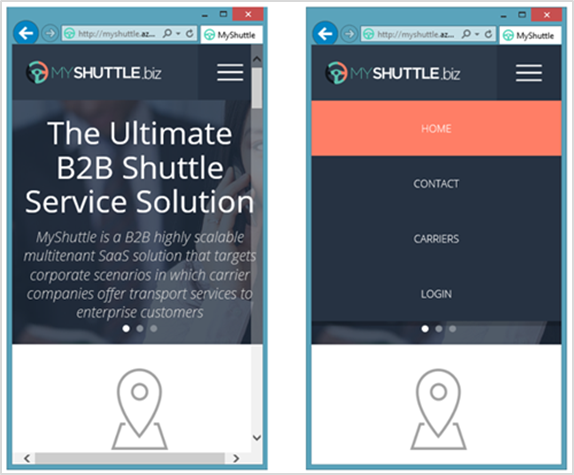


There are two web applications in the above scenario:

**Public Website:**

It is a typical public website. Its main purpose is to show information about the business but in a modern and clean way. It provides a responsive design and even if you resize the browser, you can see how it would also be perfectly valid for mobile devices, like a smartphone.

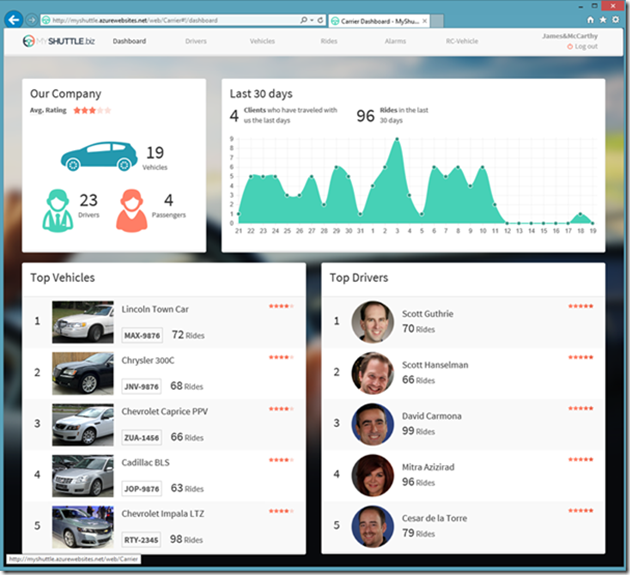




**Private Web Application:**

The second application is a Web Single Page Application (SPA), which you can access by logging in from the public website. However, in reality it is like a different web application, simulating a private web application especially made for the Carriers’ administrators.

This application is a data-driven and CRUD app so you can create and update information about your drivers, vehicles, etc. This application consumes the ASP.NET Web API Services using client-side frameworks.



In this series of labs, you will build the public website and some parts of the private web application above.

#### Objectives

This lab will show you how to:

* Create a new ASP.NET MVC 5 application in Microsoft Visual Studio 2017.
* Create Visual Studio projects for application model.
* Create application model using code-first technique.

#### System Requirements

To complete this lab, you need:

* Visual Studio 2017
* Microsoft SQL Server (any edition)

#### Hosted Lab Credentials

If the lab is exercised in Microsoft cloud environment, use the following user credentials to sign in:

* Username: aspnetuser
* Password: @Cir9hvc6!w

#### Estimated Time to Complete This Lab

60 minutes

Exercise 1: Create MyShuttle Model

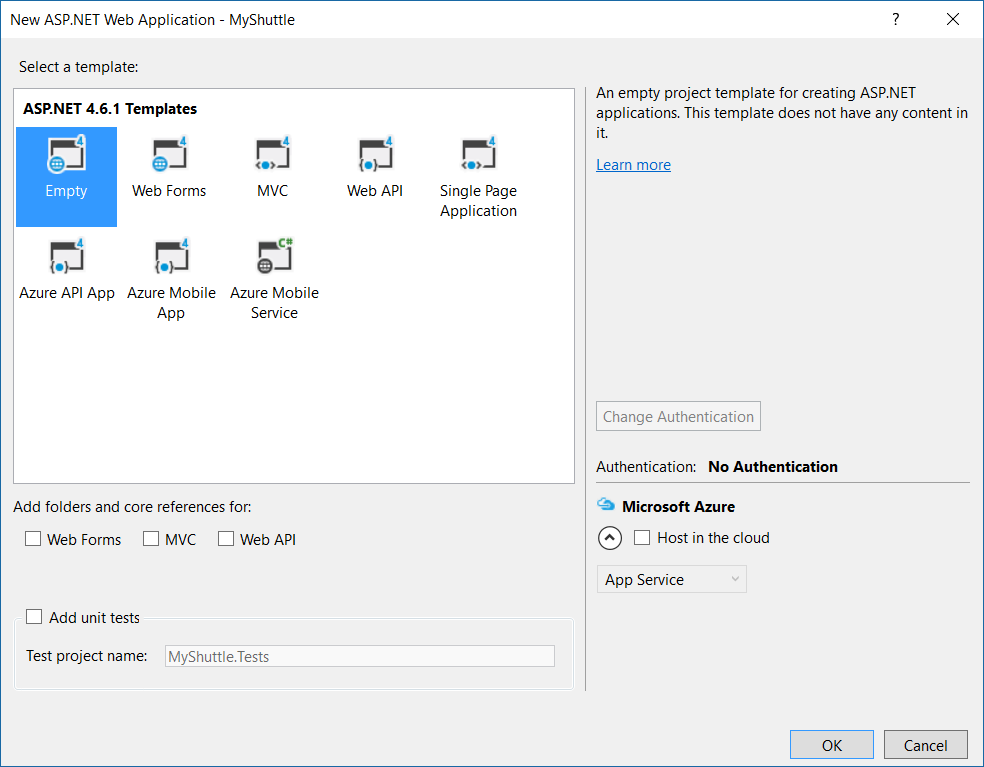
#### Objectives

In this exercise, you will:

* Create a new Visual Studio solution for MyShuttle application.
* Create a project to implement MyShuttle model through code-first technique.

Task 1: Create the Visual Studio solution

1. Open Visual Studio 2017.
2. Create a new ASP.NET Web application project by going to **File** 🡺 **New Project**.
3. Name the project as **MyShuttle**, select the **Visual C# > Web > ASP.NET Web Application (.NET Framework)** template. Create a new solution, and leave the **Create directory for solution** option selected, and then click **OK**.
4. Choose **Empty** project template.

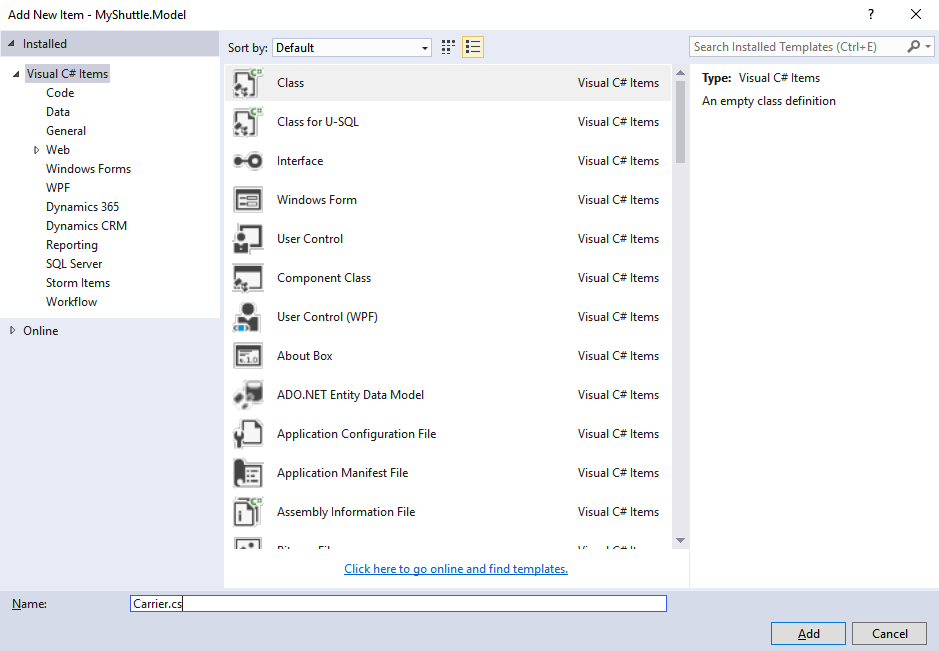


1. Close or cancel the Configure Microsoft Azure Web App pop-up window, if one appears.
2. Rename the default project as **MyShuttle.Web.**

Task 2: Create the Model Project

1. Right-click **MyShuttle** solution node in the **Solution Explorer** window and then click **Add >** **New** **project**.
2. Choose **Visual C# > Class Library** project template and name it **MyShuttle.Model**, and then click **OK**.
3. Delete **Class1.cs**.

Task 3: Implement the Model Project

1. Right-click the **MyShuttle.Model** project and click **Add** **>** **New Item**.
2. Choose **Class** from available templates and name it **Carrier.cs**
3. Remove all the “using” statements above the namespace block, apart from **System.Collections.Generic** which will be used shortly.
4. Make the class public by adding **public** keyword in front of **class Carrier**
5. Make note of the carrier information in the database, such as its name, description, address, email, logo, etc. In order to store carrier information, add the following fields to **Carrier** model class. You will add foreign key collections later.

public int CarrierId { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public string CompanyID { get; set; }

public string Address { get; set; }

public string ZipCode { get; set; }

public string City { get; set; }

public string State { get; set; }

public string Country { get; set; }

public string Phone { get; set; }

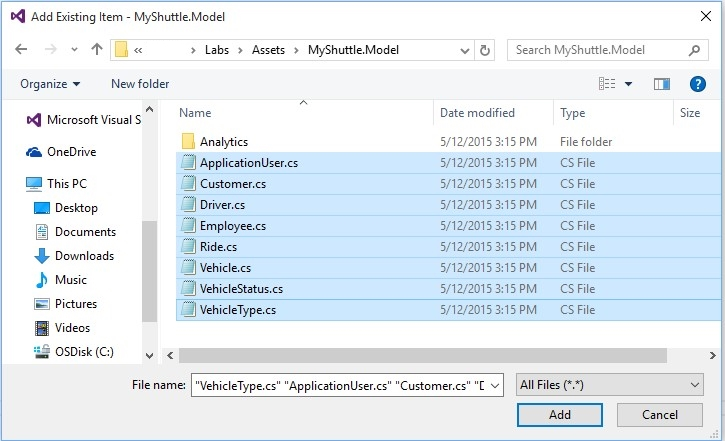
public string Email { get; set; }

public byte[] Picture { get; set; }

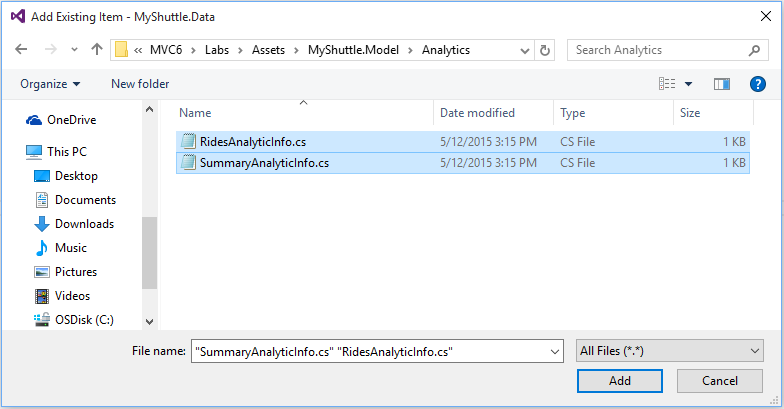
public double RatingAvg { get; set; }

1. In the same way, you also need to create models for enterprise company, company’s employees, carrier’s vehicles and drivers, and rides that enterprise company’s employees take. Additionally, you will also add model for performing analytics. Instead of writing code for each model entity right-click to **MyShuttle.Model** project, click **Add** **>** **Existing Item**, and then add the following files from the respective ***…/Assets/MyShuttle.Model*** folder.

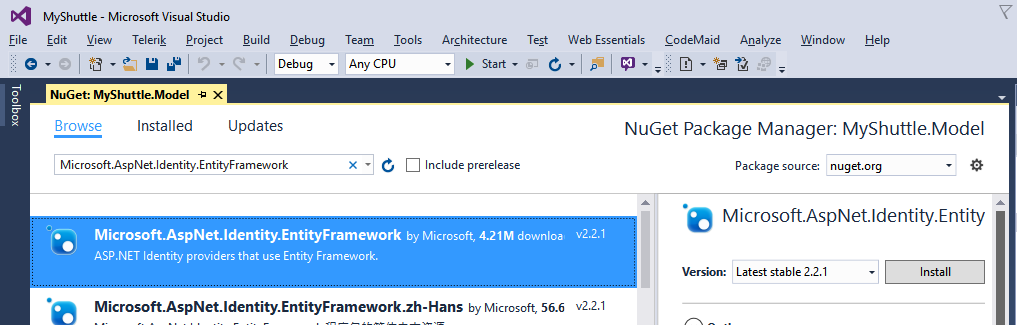
* **ApplicationUser.cs**: User of the application, connected to carrier;
* **Customer.cs**: Customer enterprise company;
* **Driver.cs**: Registered driver of the shuttle company who drives the vehicle;
* **Employee.cs**: Employee of the enterprise company who rides/rents a vehicle;
* **Ride.cs**: Record of trip taken by an employee of a company in a vehicle driven by a driver;
* **Vehicle.cs**: Vehicle driven by a Driver;
* **VehicleStatus.cs**: Current status of a vehicle, whether it’s available, busy, or unknown;
* **VehicleType.cs**: Type of vehicle, such as compact, van, luxury, etc.



1. Now that you have all the model entities defined, you will also like to perform some analytics on the database records. In order to hold analytical data, you need to define model for it. Right-click **MyShuttle.Model** project, click **Add > New Folder**, and then rename the folder as **Analytics**.
2. Right-click the **Analytics** folder and click **Add** **>** **Existing Item**. Add the two files from the **…/Assets/MyShuttle.Model/Analytics** folder.



1. You would like to add the Identity dependency for the project. Before ASP.NET MVC, you would have added either a direct assembly reference or a NuGet package. Beginning with ASP.NET MVC, direct assembly reference is not possible. Hence, you would like to add a NuGet package:
   1. Right click the **References** node under **MyShuttle.Model** Project and select **Manage NuGet Packages**
   2. Click **Browse** and search for **Microsoft.AspNet.Identity.EntityFramework**



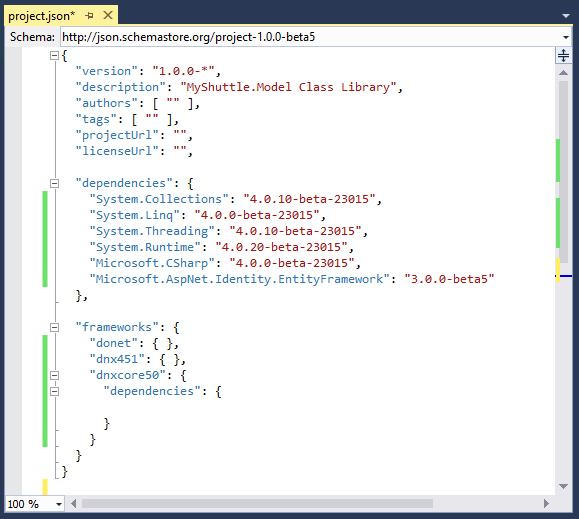
* 1. In the search results select item as in the previous picture and click **Install** button on the right
  2. Review the changes window then click **OK**.
  3. Accept Click **Install** and accept the license agreement dialog box.
  4. The Package Manager will install the other two dependencies which are:
     1. Microsoft.AspNet.Identity.Core
     2. EntityFramework

1. Each carrier has vehicles and drivers associated with it. You did not add any collection references before because model classes for vehicles and drivers did not exist then. Now that they have been added to the project, add the following collections at the bottom of **Carrier** class.

public ICollection<Vehicle> Vehicles { get; set; }

public ICollection<Driver> Drivers { get; set; }

1. Take the time to go through all the model classes and understand the attributes they have and how they are connected to each other.



1. Build the application to ensure that it compiles successfully.

You have now completed the model definition and are ready to implement data repository pattern to perform database CRUD operations.

Exercise 2: Implement MyShuttle Data CRUD Logic

#### Objectives

In this exercise, you will:

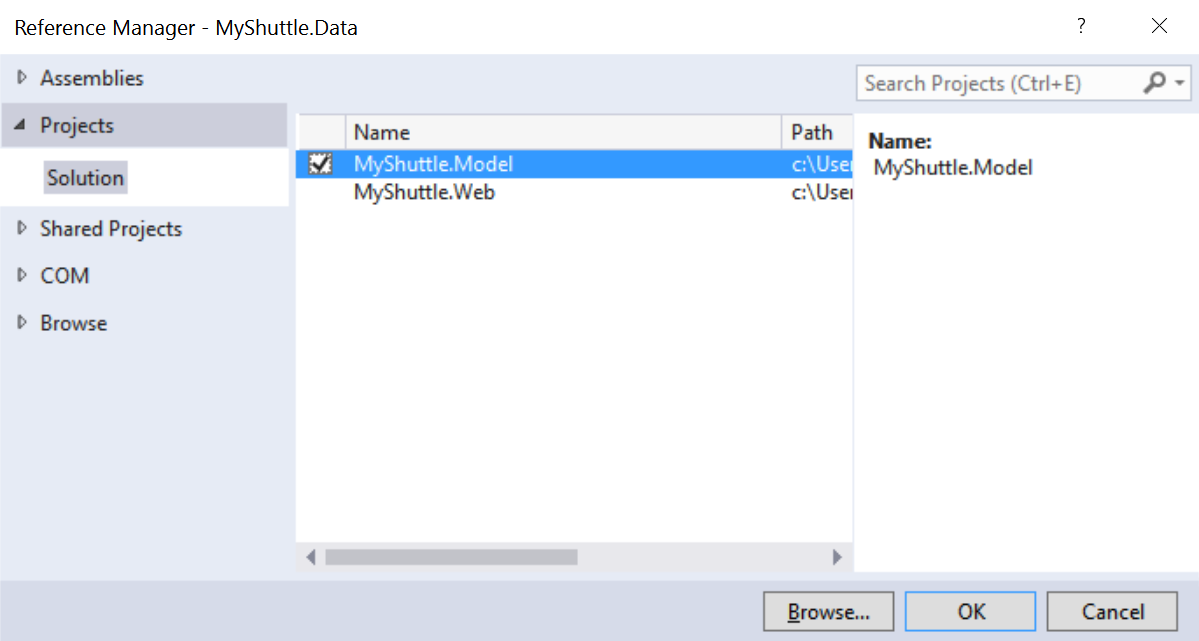
* Create a project to implement repository pattern for database CRUD operations.
* Create dummy data for this app.

Task 1: Create the data project

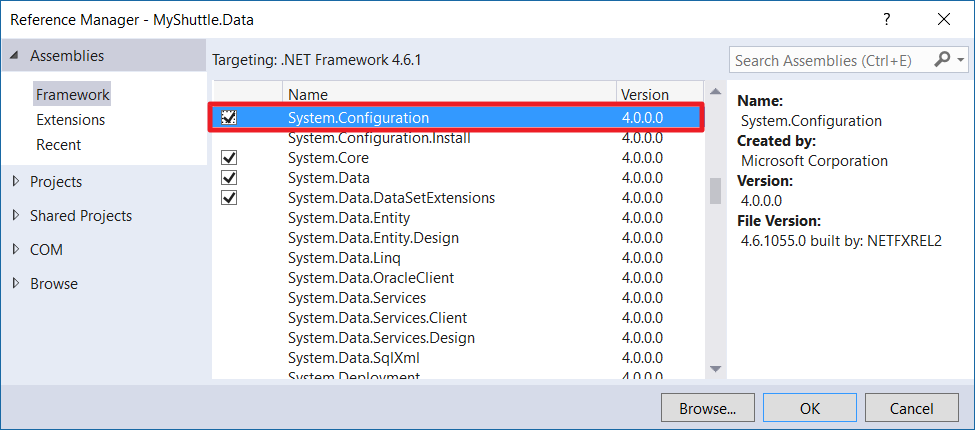
1. If it is not already open, open **MyShuttle** solution in **Visual Studio 2017**.
2. Right-click **MyShuttle** solution in the **Solution Explorer** window and then click **Add new project**.
3. Choose **Visual C# > Class Library** project template and name it **MyShuttle.Data**, and then click **OK**.
4. Delete **Class1.cs**.

Task 2: Implement the data initialization logic

1. You would first like to define dependencies for this project.
   1. Right click the **References** node under **MyShuttle.Data** project and click Add References. Select **MyShuttle.Model** from the solutions under Projects tab:



* 1. Add **System.Configuration.dll** from **Assemblies > Framework**



* 1. Add **Microsoft.AspNet.Identity.EntityFramework** NuGet package the same you did for **MyShuttle.Model** project

1. Right-click the **MyShuttle.Data** project and click **Add** **>** **Existing Item**. Add the **MyShuttleContext.cs** file to the project root from **…/Assets/MyShuttle.Data** folder. This class implements the database context of Entity Framework (EF) which declares all entities and their relationships using Code First approach.
2. Right-click the **MyShuttle.Data** project and click **Add** **>** **New Folder**. Rename the folder as **Infrastructure**.
3. Add the following two existing items under the **Infrastructure** folder from the **…/Assets/MyShuttle.Data/Infrastructure** folder.

* **FakeImages.cs:** It contains images for drivers, vehicles, etc.
* **MyShuttleDataInitializer.cs:** It contains dummy data and initialization logic.

1. Build the application to ensure that it compiles successfully.

Task 3: Implement Data Repositories

1. Right-click the **MyShuttle.Data** project and click **Add** > **New Folder**. Name the folder as **Interfaces**. This folder will contain all repository interfaces.
2. Right-click the **Interfaces** folder and click **Add >** **New** **Item** to add a new interface. Name it **ICarrierRepository.cs**
3. Define the interface as following:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using MyShuttle.Model;

namespace MyShuttle.Data

{

public interface ICarrierRepository

{

Task<int> AddAsync(Carrier carrier);

Task<SummaryAnalyticInfo> GetAnalyticSummaryInfoAsync(int carrierId);

Task<Carrier> GetAsync(int carrierId);

Task<List<Carrier>> GetCarriersAsync(string filter);

Task UpdateAsync(Carrier carrier);

}

}

* **AddAsync:** Adds a new carrier to the database.
* **GetAnalyticSummaryInfoAsync:** Performs some analytics and returns summary analytics like rating, total drivers, passengers, and vehicles of a carrier.
* **GetAsync:** Returns a carrier record matching the input Carrier ID.
* **GetCarriersAsync:** Used by the search function to return all carriers that match the input criteria.
* **UpdateAsync:** Updates carrier record in the database.

1. In order to add interfaces for other repositories, add the following existing items from the **…/Assets/MyShuttle.Data/Interfaces/**folder.

* ICustomerRepository.cs
* IDriverRepository.cs
* IEmployeeRepository.cs
* IRidesRepository.cs
* IVehicleRepository.cs

1. After defining all the interfaces, it is time to implement them. You will implement **CarrierRepository** on your own. The rest of them will be added from the **Assets** folder.
2. Right-click the **MyShuttle.Data** project and click **Add** **>** **New Folder**. Name the folder as **Repositories**. This folder will contain all repository interfaces.
3. Right-click the **Repositories** folder and click **Add** **>** **New Item** to add a new class. Name it **CarrierRepository.cs**.
4. **CarrierRepository** should implement the **ICarrierRepository** interface. Implement **CarrierRepository.cs** with the following code. Note that the default namespace would be *MyShuttle.Data.Repositories*, but it is renamed this to be *MyShuttle.Data* for simplicity.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Data.Entity;

using MyShuttle.Model;

namespace MyShuttle.Data

{

public class CarrierRepository : ICarrierRepository

{

MyShuttleContext \_context;

static readonly int DEFAULT\_PICTURE = 0;

public CarrierRepository(MyShuttleContext dbcontext)

{

\_context = dbcontext;

}

public async Task<int> AddAsync(Carrier carrier)

{

carrier.Picture = Convert.FromBase64String(FakeImages.Carriers[DEFAULT\_PICTURE]);

\_context.Carriers.Add(carrier);

await \_context.SaveChangesAsync();

return carrier.CarrierId;

}

public async Task<SummaryAnalyticInfo> GetAnalyticSummaryInfoAsync(int carrierId)

{

var passengers = await \_context.Rides.Where(r => r.CarrierId == carrierId).Select(r => r.EmployeeId).ToListAsync();

var rating = \_context.Rides.Where(r => r.CarrierId == carrierId).Select(r => r.Rating);

return new SummaryAnalyticInfo()

{

Rating = (rating.Count() > 0) ? rating.Average() : 0,

TotalDrivers = await \_context.Drivers.Where(r => r.CarrierId == carrierId).CountAsync(),

TotalPassengers = passengers.Distinct().Count(),

TotalVehicles = await \_context.Vehicles.Where(r => r.CarrierId == carrierId).CountAsync()

};

}

public async Task<Carrier> GetAsync(int carrierId)

{

return await \_context.Carriers

.Where(c => c.CarrierId == carrierId)

.SingleOrDefaultAsync();

}

public async Task<List<Carrier>> GetCarriersAsync(string filter)

{

var carriers = \_context.Carriers.AsQueryable();

if (!String.IsNullOrEmpty(filter))

{

carriers = carriers.Where(c => c.Name.ToLowerInvariant().Contains(filter.ToLowerInvariant()));

}

return await carriers.ToListAsync();

}

public async Task UpdateAsync(Carrier carrier)

{

\_context.Carriers.Attach(carrier);

\_context.Entry(carrier).State = EntityState.Modified;

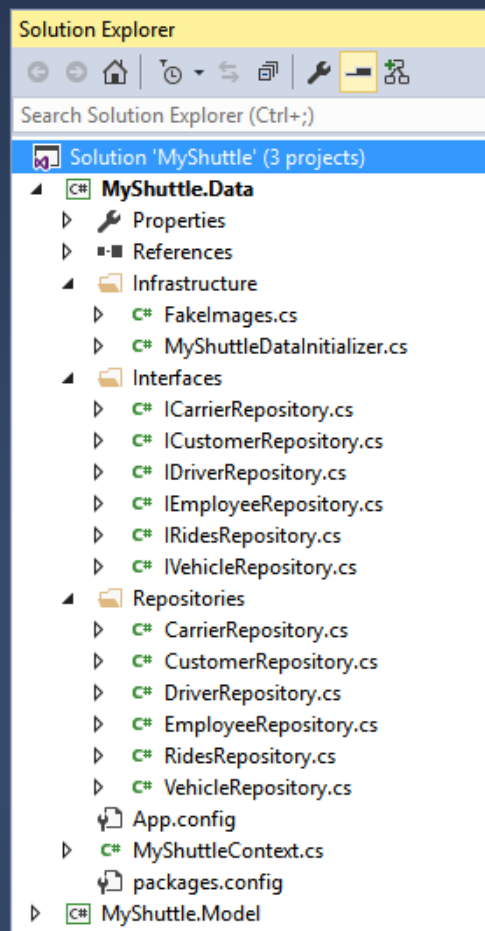
await \_context.SaveChangesAsync();

}

}

}

1. Review and understand the data CRUD logic implemented by the **CarrierRepository** class.
2. In order to add the remaining repositories, right-click the **MyShuttle.Data** project and click **Add** **>** **Existing Item** to add the following repositories:
3. CustomerRepository.cs
4. DriverRepository.cs
5. EmployeeRepository.cs
6. RidesRepository.cs
7. VehicleRepository.cs
8. After adding all the repositories, the **MyShuttle.Data** project hierarchy should look like the following:



1. Compile the solution to ensure it builds successfully.

You have now finished defining the model and have implemented the repository pattern to perform database CRUD operations. Additionally, you have also created dummy data to fill out the database initially.

In the next lab, you will implement Controllers, which will handle user requests and respond with the model data.