1. Open Visual Studio Community. Click Create a New Project. (picture 00)

Create an ASP.NET Web Application (.NET Framework)

* Name it according to your project with “.WebAPI” at the end: ProjectName.WebAPI (picture 01)
* Select Web API (picture 02)
* Click Change under “Authentication” (picture 03)
  + Select Individual User Accounts (picture 04)
  + Click “OK”
* The change should be reflected on the window. (picture 05)

1. Right Click Solution>Add>NewProject (picture 06)

* Choose “Class Library (.NET Framework) (picture 07)
* Name it (ProjectName.Data) and Click “Create” (picture 08)
* Delete the “Class1.cs” file (picture 09)
* Repeat this process for 3 more Class Libraries and name them: Models, Services, Contracts (picture 10)

1. Copy and Paste (or click and drag) “IdentityModels.cs” from the ProjectName.WebAPI>Models folder to the ProjectName.Data folder (picture 11)
   1. Delete the original IdentityModels.cs from the PeojectName.WebAPI>Models folder (picture 12)
2. A lot of errors will ensue: (ended screen shots for a while cause too slow)
   1. Change Namespace in IdentityModels.cs to match current NameSpace “ProjectName.Data”
   2. Install Entity Framework and Identity.EntityFramework into all projects
   3. Install Identity.Owin into ProjectName.Data
   4. Install WebApi.Owin into ProjectName.Data
3. Build Project (several errors)
4. References and Using Statements:
   1. Reference in WebAPI to Data Layer
   2. Using ProjectName.Data in Startup.Auth.cs
   3. Using ProjectName.Data in AccountController.cs
   4. Using ProjectName.Data in ApplicationOAuthProvider.cs
5. Build Project (should be no errors)
6. IdentityModels Configuration:

Under ApplicationDbContext class write:

public class IdentityUserLoginConfiguration : EntityTypeConfiguration<IdentityUserLogin>

{

public IdentityUserLoginConfiguration()

{

HasKey(iul => iul.UserId);

}

}

public class IdentityUserRoleConfiguration : EntityTypeConfiguration<IdentityUserRole>

{

public IdentityUserRoleConfiguration()

{

HasKey(iur => iur.UserId);

}

}

1. Within ApplicationDbContext class, underneath the CreateMethod declare a datatable:
2. public static ApplicationDbContext Create()
3. {
4. return new ApplicationDbContext();
5. }
6. public DbSet<Object> Objects { get; set; } //<--- Add this
7. }

10. Add a Class with the same name to your ProjectName.Data and make it public.

11. This will cause an error in Identity Models; bring in the Using Statement for Data.Entity

12. Change connection string to use a local Database

<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=ProjectName;Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>

Alright. We have our N-Tier Structure set up.

Data Access Layer: Data Classes, Identity Models laying out the foundation for our Identity Users, Database context, and Data Tables within our Database.

Business Layer:

Application Layer: At this level, the Application Layer and Business Layer seem to be combined in our simpler projects. This is all of our Controllers and everything in our Web.API assembly that does not directly write code to format a view. This layer likely communicates with both the Data Layer by delivering information to satisfy queries, as well as receiving information from Database following queries.

-This layer handles packaged JSON models to perform queries and info to assist in manipulating data.

-This layer handles packaged models to convert to JSON, XML, XHTML, and other formats for a Front-End Client to organize and style for an end-user’s view.

Presentation Layer: This is the layer styled for the user. It can be written in many ways. We will be using Postman to view this information in JSON after we make HTTP Requests

13. Right click ProjectName.Services and add a new Class. This class will be the “Service” class for whatever your basic Data Class will be. This will serve as a micro-repository for the methods that will interact with your Data Class For this Tutorial it will be an “AnimalService”.

At this point in time, any user will be able to call on this service to edit our Animals Data Table. We will change this in the near future to require Identity Role Based Authorization for certain tasks. Actually, currently, no user can edit our Animals Data Table because we have not created a Controller in our Application Layer to call on our Services Assembly.

First we will create a method to Post an Animal to our Database, then we will create a Controller and write an HTTP Request to call on our Services Assembly to interact with our Data Layer.

14. Within the AnimalService class in our “AnimalService.cs” file, write the following code:

public bool CreateAnimal(AnimalCreate model)

{

using (var ctx = new ApplicationDbContext())

{

var entity = new Animal()

{

Name = model.Name

};

ctx.Animals.Add(entity);

return ctx.SaveChanges() == 1;

}

}

Our AnimalCreate class has not been written yet. You may choose to write your AnimalCreate model before this method, or you may write it after, it’s up to you. Best practice is to do them successively to one another.

Steps for a Create Class:  
Determine what is required at the first incarnation of your Data Object; those properties will go in to your ObjectCreate class. For our Animal, in this simple program, we currently just need a Name. Our Animal Data Class has two properties: int Id, and string Name. Our Id is set when we add that object to our Data Table. We must set the Name upon instantiation prior to adding it to our Data Table.  
  
15. Right Click ProjectName.Models and add a new class; name it (YourBasicDataClass)Create.cs. Immediately make the class public, unless it shouldn’t be, otherwise this will cause issues later when trying to reference the class from your other Assemblies.

* Give it only the properties that are required for your Data Class to be created. In this case, as mentioned before, we will only be giving this class a Name property. Typically, the properties of this Model will be set in either the body/params of a PostMan request or in the JSON converted Model passed by your Front-End app in a HTTP Request.

public class AnimalCreate

{

public string Name { get; set; }

}

We have no errors, but nothing is going on. We need to create an Animal Controller to handle the various Models of our Animal Class. We only have one Model right now: AnimalCreate, but we will have more in the future.

16. Right Click the Controllers folder in ProjectName.WebAPI and navigate to >Add>Controller. Enter “AnimalController” into the text prompt.

Write:

public IHttpActionResult Post(AnimalCreate model)

{

var service = new AnimalService();

service.CreateAnimal(model);

return Ok(model);

}

Bring in Using Statements for AnimalCreate and AnimalService.

17. Navigate back to AnimalService. Create a method to GetAllAnimals:

public IEnumerable<AnimalListItem> GetAllAnimals()

{

using (var ctx = new ApplicationDbContext())

{

var query = ctx.Animals.Select(

a => new AnimalListItem()

{

Id = a.Id,

Name = a.Name

});

return query.ToArray();

}

}