# Build an ASP.NET Core MVC App with EF Core One-Day Hands-On Lab

## Lab 2

This lab walks you through creating the Models and ViewModels. Prior to starting this lab, you must have completed Lab 1.

All work in this lab is to be completed in the AutoLot.Models project.

# Part 1: The GlobalUsings.cs File

☐ Begin by renaming the autogenerated `Class1.cs` to `GlobalUsings.cs` and clear out the templated so it's an empty file. Add the following global using statements to the file:

```
global using Microsoft.EntityFrameworkCore;
global using Microsoft.EntityFrameworkCore.Metadata.Builders;

global using System.ComponentModel;
global using System.ComponentModel.DataAnnotations;
global using System.ComponentModel.DataAnnotations.Schema;
global using System.Globalization;
global using System.Xml.Linq;
```

# Part 2: Creating the Entities in AutoLot.Models

The entities represent the data that is persisted in SQL Server and can be shaped to be more application specific.

## Step 1: Create the Base Entity

☐ Create a new folder in the `AutoLot.Models` project named `Entities`. Create a subfolder named `Base` under the `Entities` folder. Add a new class to the `Base` folder named `BaseEntity.cs`, and update the code to the following:

```
namespace AutoLot.Models.Entities.Base;
public abstract class BaseEntity
{
  [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)] public int Id { get; set; }
  [Timestamp] public byte[] TimeStamp { get; set; }
}
```

☐ Add the following global using statement to the `GlobalUsings.cs` file:

```
global using AutoLot.Models.Entities.Base;
```

## Step 2: Create the Person Owned Class

Owned classes can be reused between other entities and get folded into the parent entity's table.

☐ Add a folder named `Owned` under the `Entities` folder, and add a new class named `Person.cs`. Update the code for the class to the following:

```
namespace AutoLot.Models.Entities.Owned;

[Owned]
public class Person
{
  [Required, StringLength(50)] public string FirstName { get; set; }
  [Required, StringLength(50)] public string LastName { get; set; }
  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public string FullName { get; set; }
}
```

☐ Add the following global using statement to the `GlobalUsings.cs` file:

```
global using AutoLot.Models.Entities.Owned;
```

## Step 3: Create the Car Entity

☐ Add a new class to the `Entities` folder named `Car.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Inventory", Schema = "dbo")]
[Index(nameof(MakeId), Name = "IX_Inventory_MakeId")]
public class Car : BaseEntity
{
  [Required,StringLength(50)]
  public string Color { get; set; }
  public string Price { get; set; }

  private bool? _isDrivable;
  [DisplayName("Is Drivable")]
  public bool IsDrivable
  {
    get => _isDrivable ?? true;
    set => _isDrivable = value;
  }
  public DateTime? DateBuilt { get; set; }
  [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
  public string Display { get; set; }
  [Required, StringLength(50), DisplayName("Pet Name")]
  public string PetName { get; set; }
  [Required, DisplayName("Make")]
  public int MakeId { get; set; }
}
```

## Step 4: Create the CarDriver Entity

☐  Add a new class to the `Entities` folder named `CarDriver.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("InventoryToDrivers", Schema = "dbo")]
public class CarDriver : BaseEntity
{
  public int DriverId { get; set; }
  [Column("InventoryId")]
  public int CarId { get; set; }
}
```

## Step 5: Create the CreditRisk Entity

The `CreditRisk` entity uses the `Person` owned class.

☐  Add a new class to the `Entities` folder named `CreditRisk.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("CreditRisks", Schema = "dbo")]
[Index(nameof(CustomerId), Name = "IX_CreditRisks_CustomerId")]
public class CreditRisk : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
  public int CustomerId { get; set; }
}
```

## Step 6: Create the Customer Entity

The Customer `entity` uses the `Person` owned class.

☐  Add a new class to the `Entities` folder named `Customer.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Customers", Schema = "dbo")]
public class Customer : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
}
```

## Step 7: Create the Driver Entity

☐  Add a new class to the `Entities` folder named `Driver.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Drivers", Schema = "dbo")]
public class Driver : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
}
```

## Step 8: Create the Make Entity

☐ Add a new class to the `Entities` folder named `Make.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Makes", Schema = "dbo")]
public class Make : BaseEntity
{
  [Required, StringLength(50)]
  public string Name { get; set; }
}
```

## Step 9: Create the Order Entity

☐ Add a new class to the `Entities` folder named `Order.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Orders", Schema = "dbo")]
[Index(nameof(CarId), Name = "IX_Orders_CarId")]
[Index(nameof(CustomerId), nameof(CarId), Name = "IX_Orders_CustomerId_CarId", IsUnique = true)]
public class Order : BaseEntity
{
  public int CarId { get; set; }
  public int CustomerId { get; set; }
}
```

## Step 10: Create the Radio Entity

☐ Add a new class to the `Entities` folder named `Radio.cs` and update the code to the following:

```
namespace AutoLot.Models.Entities;

[Table("Radios", Schema = "dbo")]
public class Radio : BaseEntity
{
  public bool HasTweeters { get; set; }
  public bool HasSubWoofers { get; set; }
  [Required, StringLength(50)]
  public string RadioId { get; set; }
  [Column("InventoryId")]
  public int CarId { get; set; }
}
```

## Step 11: Create the Logging Entity

The SeriLog logging framework has an option to write log entries to a database table. We will use EF Core to create the tables for us, even though it doesn't represent a domain entity.

☐ Add a new class to the `Entities` folder named `SeriLogentry.cs` and update the code to the following:

```csharp
namespace AutoLot.Models.Entities;

[Table("SeriLogs", Schema = "Logging")]
public class SeriLogEntry
{
  [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
  public int Id { get; set; }
  public string Message { get; set; }
  public string MessageTemplate { get; set; }
  [MaxLength(128)]
  public string Level { get; set; }
  [DataType(DataType.DateTime)]
  public DateTime TimeStamp { get; set; }
  public string Exception { get; set; }
  public string Properties { get; set; }
  public string LogEvent { get; set; }
  public string SourceContext { get; set; }
  public string RequestPath { get; set; }
  public string ActionName { get; set; }
  public string ApplicationName { get; set; }
  public string MachineName { get; set; }
  public string FilePath { get; set; }
  public string MemberName { get; set; }
  public int LineNumber { get; set; }
  [NotMapped]
  public XElement PropertiesXml => (Properties != null) ? XElement.Parse(Properties) : null;
}
```

# Part 3: Creating the Navigation Properties

Navigation properties represent foreign key relationships between entities. Collection navigation properties represent the Many end of a One to Many or Many to Many relationship and reference navigation properties represent the One end of a One to Many or One to One relationship.

## Step 1: Update The Car Entity

☐ Update the `Car` entity by adding the following reference and collection navigation properties:

```
public class Car : BaseEntity
{
  //omitted for brevity

  [ForeignKey(nameof(MakeId))]
  [InverseProperty(nameof(Make.Cars))]
  public Make MakeNavigation { get; set; }

  [InverseProperty(nameof(Radio.CarNavigation))]
  public Radio RadioNavigation { get; set; }

  [InverseProperty(nameof(Driver.Cars))]
  public IEnumerable<Driver> Drivers { get; set; } = new List<Driver>();

  [InverseProperty(nameof(CarDriver.CarNavigation))]
  public IEnumerable<CarDriver> CarDrivers { get; set; } = new List<CarDriver>();

  [InverseProperty(nameof(Order.CarNavigation))]
  public IEnumerable<Order> Orders { get; set; } = new List<Order>();
}
```

☐ Add the `[NotMapped]` `MakeName` property and the override for `ToString()`, both of which use the `MakeNavigation` reference navigation property:

```
public class Car : BaseEntity
{
  //omitted for brevity

  [NotMapped]
  public string MakeName => MakeNavigation?.Name ?? "Unknown";

  public override string ToString()
  {
    // Since the PetName column could be empty, supply
    // the default name of **No Name**.
    return $"{PetName ?? "**No Name**"} is a {Color} {MakeNavigation?.Name} with ID {Id}.";
  }
}
```

## Step 2: Update the CarDriver Entity

☐ Update the `CarDriver` entity by adding the following reference navigation properties:

```
public class CarDriver : BaseEntity
{
  public int DriverId { get; set; }
  [ForeignKey(nameof(DriverId))]
  public Driver DriverNavigation { get; set; }

  [Column("InventoryId")]
  public int CarId { get; set; }
  [ForeignKey(nameof(CarId))]
  public Car CarNavigation { get; set; }
}
```

## Step 3: Update the CreditRisk Entity

☐ Update the `CreditRisk` entity by adding the following reference navigation property:

```
public class CreditRisk : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();

  public int CustomerId { get; set; }
  [ForeignKey(nameof(CustomerId))]
  [InverseProperty(nameof(Customer.CreditRisks))]
  public Customer CustomerNavigation { get; set; }
}
```

## Step 4: Update the Customer Entity

☐ Update the `Customer` entity by adding the following collection navigation properties:

```
public class Customer : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
  [InverseProperty(nameof(CreditRisk.CustomerNavigation))]
  public IEnumerable<CreditRisk> CreditRisks { get; set; } = new List<CreditRisk>();
  [InverseProperty(nameof(Order.CustomerNavigation))]
  public IEnumerable<Order> Orders { get; set; } = new List<Order>();
}
```

## Step 5: Update the Driver Entity

☐ Update the `Driver` entity by adding the following collection navigation properties:

```
public class Driver : BaseEntity
{
  public Person PersonInformation { get; set; } = new Person();
  [InverseProperty(nameof(Car.Drivers))]
  public IEnumerable<Car> Cars { get; set; } = new List<Car>();
  [InverseProperty(nameof(CarDriver.DriverNavigation))]
  public IEnumerable<CarDriver> CarDrivers { get; set; } = new List<CarDriver>();
}
```

## Step 6: Update the Make Entity

☐ Update the Make entity by adding the following collection navigation property:

```
public class Make : BaseEntity
{
  [Required, StringLength(50)]
  public string Name { get; set; }
  [InverseProperty(nameof(Car.MakeNavigation))]
  public IEnumerable<Car> Cars { get; set; } = new List<Car>();
}
```

## Step 7: Update the Order Entity

☐ Update the Order entity by adding the following reference navigation properties:

```
public partial class Order : BaseEntity
{
  public int CarId { get; set; }
  [ForeignKey(nameof(CarId))]
  [InverseProperty(nameof(Car.Orders))]
  public Car CarNavigation { get; set; }

  public int CustomerId { get; set; }
  [ForeignKey(nameof(CustomerId))]
  [InverseProperty(nameof(Customer.Orders))]
  public Customer CustomerNavigation { get; set; }
}
```

## Step 8: Update the Radio Entity

☐ Update the Radio entity by adding the following reference and collection navigation properties:

```
public class Radio : BaseEntity
{
  //omitted for brevity
  public int CarId { get; set; }
  [ForeignKey(nameof(CarId))]
  public Car CarNavigation { get; set; }
}
```

# Part 4: Configure the Entities with Fluent API

Fine tuning of the model takes place in the Fluent API. Introduced in EF Core 6, The `IEntityTypeConfiguration<T>` interface allows for placing each entity's Fluent API code in specific configuration classes. This section configures the entities, the final step to utilize these configuration classes will be coded when the `ApplicationDbContext` is built.

Start by creating a folder name `Configuration` under the Entities folder. Add the following to the `GlobalUsings.cs` file:

```
global using AutoLot.Models.Entities.Configuration;
```

## Step 1: Configure The Car Entity

☐ Add a new class in the Configuration folder named `CarConfiguration`. Clear out the code and update it to match the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class CarConfiguration : IEntityTypeConfiguration<Car>
{
  public void Configure(EntityTypeBuilder<Car> builder)
  {
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("InventoryAudit");
    }));
    builder.HasQueryFilter(c => c.IsDrivable);
    builder.Property(p => p.IsDrivable).HasField("_isDrivable").HasDefaultValue(true);
    builder.Property(e => e.DateBuilt).HasDefaultValueSql("getdate()");
    builder.Property(e => e.Display)
           .HasComputedColumnSql("[PetName] + ' (' + [Color] + ')'", stored: true);
    CultureInfo provider = new("en-us");
    NumberStyles style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
    builder.Property(p => p.Price)
           .HasConversion(v => decimal.Parse(v, style, provider), v => v.ToString("C2"));
    builder.HasOne(d => d.MakeNavigation).WithMany(p => p.Cars).HasForeignKey(d => d.MakeId)
           .OnDelete(DeleteBehavior.ClientSetNull).HasConstraintName("FK_Inventory_Makes_MakeId");
    builder.HasMany(p => p.Drivers).WithMany(p => p.Cars).UsingEntity<CarDriver>(
        j => j.HasOne(cd => cd.DriverNavigation).WithMany(d => d.CarDrivers)
          .HasForeignKey(nameof(CarDriver.DriverId))
          .HasConstraintName("FK_InventoryDriver_Drivers_DriverId")
          .OnDelete(DeleteBehavior.Cascade),
        j => j.HasOne(cd => cd.CarNavigation).WithMany(c => c.CarDrivers)
          .HasForeignKey(nameof(CarDriver.CarId))
          .HasConstraintName("FK_InventoryDriver_Inventory_InventoryId")
          .OnDelete(DeleteBehavior.ClientCascade),
        j => { j.HasKey(x => x.Id);
             j.HasIndex(cd => new { cd.CarId, cd.DriverId }).IsUnique(true);
           });
  }
}
```

☐ Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("Inventory", Schema = "dbo")]
[Index(nameof(MakeId), Name = "IX_Inventory_MakeId")]
[EntityTypeConfiguration(typeof(CarConfiguration))]
public class Car : BaseEntity
{
  //omitted for brevity
}
```

## Step 2: Configure the CarDriver Entity

☐ Add a new class named CarDriverConfiguration and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class CarDriverConfiguration : IEntityTypeConfiguration<CarDriver>
{
  public void Configure(EntityTypeBuilder<CarDriver> builder)
  {
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("InventoryToDriversAudit");
    }));
    builder.HasQueryFilter(cd => cd.CarNavigation.IsDrivable);
  }
}
```

☐ Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("InventoryToDrivers", Schema = "dbo")]
[EntityTypeConfiguration(typeof(CarDriverConfiguration))]
public class CarDriver : BaseEntity
{
  //omitted for brevity
}
```

## Step 3: Configure the CreditRisk Entity

The CreditRisk entity can't be set up as a temporal table because it uses an Owned class.

&#9633;  Add a new class named CreditRiskConfiguration and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class CreditRiskConfiguration : IEntityTypeConfiguration<CreditRisk>
{
  public void Configure(EntityTypeBuilder<CreditRisk> builder)
  {
    builder.HasOne(d => d.CustomerNavigation)
           .WithMany(p => p.CreditRisks)
           .HasForeignKey(d => d.CustomerId)
           .HasConstraintName("FK_CreditRisks_Customers");
    builder.OwnsOne(o => o.PersonInformation,
        pd =>
          {
            pd.Property<string>(nameof(Person.FirstName))
              .HasColumnName(nameof(Person.FirstName))
              .HasColumnType("nvarchar(50)");
            pd.Property<string>(nameof(Person.LastName))
              .HasColumnName(nameof(Person.LastName))
              .HasColumnType("nvarchar(50)");
            pd.Property(p => p.FullName)
              .HasColumnName(nameof(Person.FullName))
              .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
          });
    builder.Navigation(d => d.PersonInformation).IsRequired(true);
  }
}
```

&#9633;  Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("CreditRisks", Schema = "dbo")]
[Index(nameof(CustomerId), Name = "IX_CreditRisks_CustomerId")]
[EntityTypeConfiguration(typeof(CreditRiskConfiguration))]
public class CreditRisk : BaseEntity
{
  //omitted for brevity
}
```

## Step 4: Configure the Customer Entity

The `Customer` entity can't be set up as a temporal table because it uses an `Owned` class.

☐ Add a new class named `CustomerConfiguration` and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class CustomerConfiguration : IEntityTypeConfiguration<Customer>
{
  public void Configure(EntityTypeBuilder<Customer> builder)
  {
    builder.OwnsOne(o => o.PersonInformation,
      pd =>
        {
          pd.Property<string>(nameof(Person.FirstName))
            .HasColumnName(nameof(Person.FirstName))
            .HasColumnType("nvarchar(50)");
          pd.Property<string>(nameof(Person.LastName))
            .HasColumnName(nameof(Person.LastName))
            .HasColumnType("nvarchar(50)");
          pd.Property(p => p.FullName)
            .HasColumnName(nameof(Person.FullName))
            .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
    });
    builder.Navigation(d => d.PersonInformation).IsRequired(true);
  }
}
```

☐ Add the `[EntityTypeConfiguration]` attribute to the class:

```
[Table("Customers", Schema = "dbo")]
[EntityTypeConfiguration(typeof(CustomerConfiguration))]
public class Customer : BaseEntity
{
  //omitted for brevity
}
```

## Step 5: Configure the Driver Entity

The Driver entity can't be set up as a temporal table because it uses an Owned class.

☐ Add a new class named DriverConfiguration and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class DriverConfiguration : IEntityTypeConfiguration<Driver>
{
  public void Configure(EntityTypeBuilder<Driver> builder)
  {
    builder.OwnsOne(o => o.PersonInformation,
      pd =>
        {
          pd.Property<string>(nameof(Person.FirstName))
            .HasColumnName(nameof(Person.FirstName))
            .HasColumnType("nvarchar(50)");
          pd.Property<string>(nameof(Person.LastName))
            .HasColumnName(nameof(Person.LastName))
            .HasColumnType("nvarchar(50)");
          pd.Property(p => p.FullName)
            .HasColumnName(nameof(Person.FullName))
            .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
        });
    builder.Navigation(d => d.PersonInformation).IsRequired(true);
  }
}
```

☐ Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("Drivers", Schema = "dbo")]
[EntityTypeConfiguration(typeof(DriverConfiguration))]
public class Driver : BaseEntity
{
  //omitted for brevity
}
```

## Step 6: Configure the Make Entity

☐ Add a new class named MakeConfiguration and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class MakeConfiguration : IEntityTypeConfiguration<Make>
{
  public void Configure(EntityTypeBuilder<Make> builder)
  {
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("MakesAudit");
    }));
  }
}
```

□ Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("Makes", Schema = "dbo")]
[EntityTypeConfiguration(typeof(MakeConfiguration))]
public class Make : BaseEntity
{
  //omitted for brevity
}
```

## Step 7: Configure the Order Entity

□ Add a new class named OrderConfiguration and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class OrderConfiguration : IEntityTypeConfiguration<Order>
{
  public void Configure(EntityTypeBuilder<Order> builder)
  {
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("OrdersAudit");
    }));
    builder.HasIndex(cr => new { cr.CustomerId, cr.CarId }).IsUnique(true);
    builder.HasQueryFilter(e => e.CarNavigation!.IsDrivable);

    builder.HasOne(d => d.CarNavigation)
           .WithMany(p => p.Orders)
           .HasForeignKey(d => d.CarId)
           .OnDelete(DeleteBehavior.ClientSetNull)
           .HasConstraintName("FK_Orders_Inventory");

    builder.HasOne(d => d.CustomerNavigation)
           .WithMany(p => p.Orders)
           .HasForeignKey(d => d.CustomerId)
           .HasConstraintName("FK_Orders_Customers");
  }
}
```

□ Add the [EntityTypeConfiguration] attribute to the class:

```
[Table("Orders", Schema = "dbo")]
[Index(nameof(CarId), Name = "IX_Orders_CarId")]
[Index(nameof(CustomerId), nameof(CarId), Name = "IX_Orders_CustomerId_CarId", IsUnique = true)]
[EntityTypeConfiguration(typeof(OrderConfiguration))]
public class Order : BaseEntity
{
  //omitted for brevity
}
```

## Step 8: Configure the Radio Entity

☐ Add a new class named `RadioConfiguration` and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class RadioConfiguration : IEntityTypeConfiguration<Radio>
{
  public void Configure(EntityTypeBuilder<Radio> builder)
  {
    builder.ToTable( b => b.IsTemporal(t =>
    {
      t.HasPeriodEnd("ValidTo");
      t.HasPeriodStart("ValidFrom");
      t.UseHistoryTable("RadiosAudit");
    }));
    builder.HasIndex(e => e.CarId, "IX_Radios_CarId").IsUnique();
    builder.HasQueryFilter(e => e.CarNavigation.IsDrivable);
    builder.HasOne(d => d.CarNavigation).WithOne(p => p.RadioNavigation)
           .HasForeignKey<Radio>(d => d.CarId);
  }
}
```

☐ Add the `[EntityTypeConfiguration]` attribute to the class:

```
[Table("Radios", Schema = "dbo")]
[EntityTypeConfiguration(typeof(RadioConfiguration))]
public class Radio : BaseEntity
{
  //omitted for brevity
}
```

## Step 9: Configure the SeriLogEntry Entity

☐ Add a new class named `SeriLogEntryConfiguration` and update it to the following:

```
namespace AutoLot.Models.Entities.Configuration;

public class SeriLogEntryConfiguration : IEntityTypeConfiguration<SeriLogEntry>
{
  public void Configure(EntityTypeBuilder<SeriLogEntry> builder)
  {
    builder.Property(e => e.Properties).HasColumnType("Xml");
    builder.Property(e => e.TimeStamp).HasDefaultValueSql("GetDate()");
  }
}
```

☐ Add the `[EntityTypeConfiguration]` attribute to the class:

```
[Table("SeriLogs", Schema = "Logging")]
[EntityTypeConfiguration(typeof(SeriLogEntryConfiguration))]
public class SeriLogEntry
{
  //omitted for brevity
}
```

# Part 5: Create the ViewModels in AutoLot.Models

There are two ViewModels used by the applications.

☐ Create a new folder named `ViewModels` under the project folder. Add the following to the GlobalUsings.cs file:

```
global using AutoLot.Models.ViewModels;
global using AutoLot.Models.ViewModels.Configuration;
```

## Step 1: Create the CustomerOrderViewModel

☐ Add a new class named `CustomerOrderViewModel.cs` and update the code for the class to the following:

```
namespace AutoLot.Models.ViewModels;

[Keyless]
public partial class CustomerOrderViewModel
{
  [Required,StringLength(50)]
  public string FirstName { get; set; }
  [Required,StringLength(50)]
  public string LastName { get; set; }
  [Required,StringLength(50)]
  public string Color { get; set; }
  [Required,StringLength(50)]
  public string PetName { get; set; }
  [Required,StringLength(50)]
  public string Make { get; set; }
  public bool? IsDrivable { get; set; }
  public string Display { get; set; }
  public string Price { get; set; }
  public DateTime? DateBuilt { get; set; }
  [NotMapped]
  public string FullDetail => $"{FirstName} {LastName} ordered a {Color} {Make} named {PetName}";
  public override string ToString() => FullDetail;
}
```

☐ Add a new folder named `Configuration` under the `ViewModels` folder. In that folder add a new class named `CustomerOrderViewModelConfiguration.cs` and update the code for the class to the following:

```
namespace AutoLot.Models.ViewModels.Configuration;

public class CustomerOrderViewModelConfiguration :
  IEntityTypeConfiguration<CustomerOrderViewModel>
{
  public void Configure(EntityTypeBuilder<CustomerOrderViewModel> builder)
  {
    builder.ToView("CustomerOrderView");
  }
}
```

☐ Add the `[EntityTypeConfiguration]` attribute to the class:

```
[Keyless]
[EntityTypeConfiguration(typeof(CustomerOrderViewModelConfiguration))]
public partial class CustomerOrderViewModel
{
  //omitted for brevity
}
```

## Step 2: Create the TemporalViewModel Class

This class is used when querying temporal tables and does not exist in the database.

☐ Add a new class named `TemporalViewModel.cs` and update the code to the following:

```
namespace AutoLot.Models.ViewModels;

public class TemporalViewModel<T> where T: BaseEntity, new()
{
    public T Entity { get; set; }
    public DateTime ValidFrom { get; set; }
    public DateTime ValidTo { get; set; }
}
```

# Summary

In this lab, you created the Models (Entities) and the ViewModels for the applications.

# Next steps

In the next part of this tutorial series, you will create the DbContext, DbContext Factory, and run your first migration.