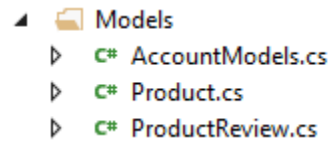


## Playbook for Code First Approach MVC

### 1. Define the models



### 2. Define the Attributes in the Model (Keys)

```
public class Product
{
    [Key]
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public string ProductType { get; set; }

    public int ProductPrice { get; set; }

    public string ProductDesc { get; set; }
}
```

### 3. Define Navigation Property

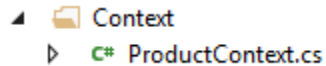
```
public class ProductReview
{
    [Key]
    public int ReviewId { get; set; }
    public int Rating { get; set; }

    public string comment { get; set; }

    // foreign key
    public int ProductId { get; set; }

    //Navigation property
    public Product Product { get; set; }
}
```

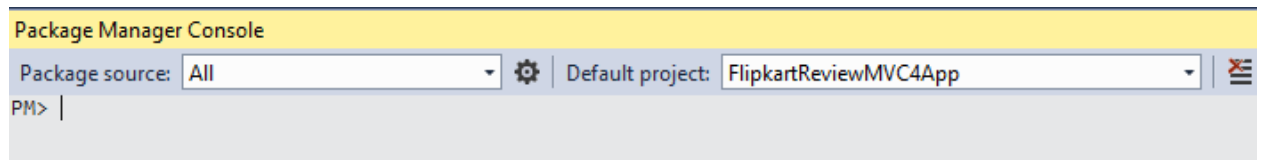
#### 4. Create Corresponding Context for the application



#### 5. Register the Context in Web.Config by copying and replacing the connection string and replacing the default connection string to the context name define in step 4.

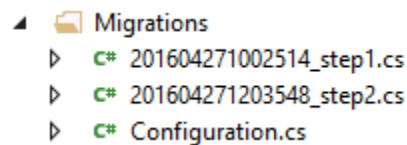
```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;Initial C;
  <add name="ProductContext" connectionString="Data Source=DESKTOP-I57J0S9\SQLEXPRESS20;
</connectionStrings>
```

#### 6. Run the Package Manager Console



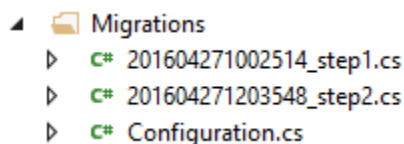
#### 7. Enable-Migrations -ContextType ContextName(Specified in Step 4)

- A Migrations folder with Configuration.cs is created as shown below



#### 8. Update-Database

#### 9. Add-Migration step1



- Define the corresponding controllers for the models to tie up the models to the controllers. Use appropriate scaffolding to suite to the requirements ( preferably read/write actions)

Add Controller

Controller name:  
ProductController

Scaffolding options

Template:  
MVC controller with empty read/write actions

Model class:

Data context class:

Views:  
None

Advanced Options...

Add Cancel

0 references

```
public class ProductController : Controller
{
    ProductContext db = new ProductContext();
    //
    // GET: /Product/
```

0 references

```
public ActionResult Index()
{
    return View(db.Products.ToList());
}

//
// GET: /Product/Details/5
```

0 references

```
public ActionResult Details(int id)
{
    return View();
}
```

11. Render corresponding views by clicking on the controller action methods and choose appropriate scaffolding to suit your requirements.

The screenshot shows the 'Add View' dialog box with the following fields and options:

- View name:** Index
- View engine:** Razor (CSHTML)
- ☒ **Create a strongly-typed view**
  - Model class:** Product (FlipkartReviewMVC4App.Models)
  - Scaffold template:** List
  - ☒ **Reference script libraries**
- ☐ **Create as a partial view**
- ☒ **Use a layout or master page:**
  - ~/Views/Shared/\_Layout.cshtml
  - (Leave empty if it is set in a Razor \_viewstart file)
- ContentPlaceHolder ID:** MainContent

Buttons: Add, Cancel

12. Build your application and run the application
13. Navigate to the corresponding routes
14. <http://localhost:port/product/index>