

Introduction

This is a walk through exercise to help you set up the Rental Properties solution using the Entity Framework Code-First walk-through document from the first day of .NET Core. In that first walk-through document we left off having created a Razor Pages front end project (the presentation layer). In this walk-through exercise, we will add a new presentation project and set it as the startup project. Then move on to adding manager classes, a view model representation of the rental domain entity, controllers and views.

New Presentation Layer

1. We will start by adding a new Web Application (Model-View-Controller) project to our solution and change the name to CPRG214.Properties.Presentation and set it as the startup project in the solution explorer window.
2. Follow same steps as you did when adding the razor pages project. You need to add project references to the BLL and Domain projects. The namespaces from these projects will be used in files requiring them so that we can use the classes defined in them.

BLL Layer Changes

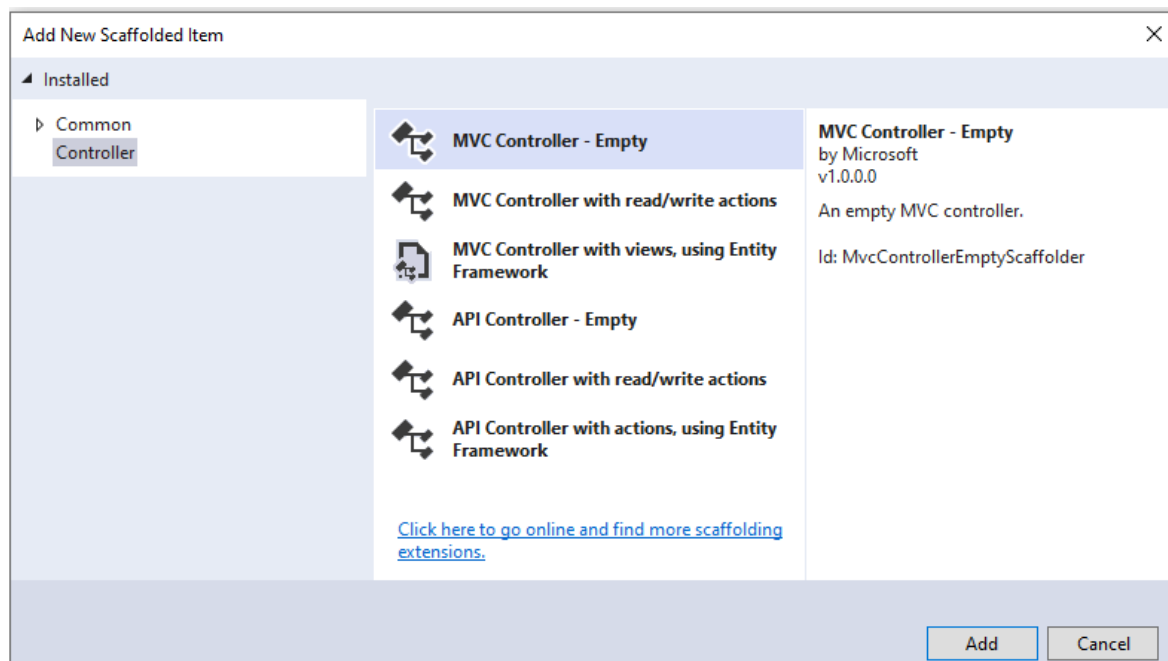
3. Add manager classes to the BLL project. Existing manager is *RentalsManager*, so add *OwnerManager*, *RenterManager* and *PropertyTypeManager*.
4. Add the same method from *RentalsManager* to the three new managers. The *GetAsKeyValuePairs* method is used for populating select lists thus we only need 2 values as strings: the text to display in the select and the value. Copy the method from the rentals manager to the three new ones and change the *DbSet* property of the context to *Owners*, *Renters*, *PropertyTypes* respectively in the method. The value field property does not change as all are the *Id* property but change the *Text* property assignment to *First and Last name (concatenated)* for renters, *Name* for owner and *Style* for property types.
5. Create a rental view model by adding a new class to the Models folder called *RentalViewModel*. The view model is another representation of the rental property entity within the application. Each layer may have a different representation of the entity. In the presentation layer the entity may be known as the view model—what does the View require to render its display. The properties of the *RentalViewModel* class are similar to the Rental domain entity but replacing foreign key values with the main name of that entity. A simple way to create view models is to start with an existing domain entity. Copy the properties from the domain entity class to the new view model class and change the data types to strings and foreign key values to a string display. So for property type id, I replaced it with a string called *PropertyType* and it will store the style property value from

the property type object:

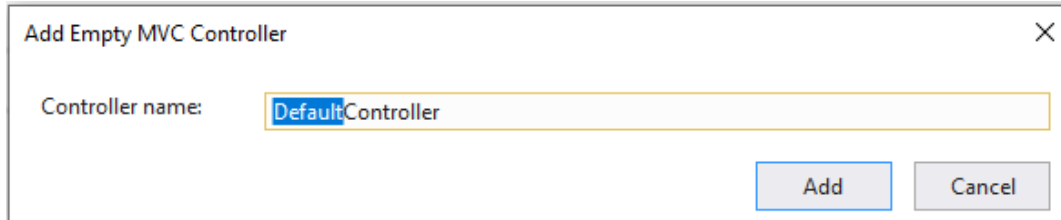
```
public class RentalViewModel
{
    4 references
    public int Id { get; set; }
    6 references
    public string Address { get; set; }
    6 references
    public string City { get; set; }
    6 references
    public string Province { get; set; }
    6 references
    public string Rent { get; set; }
    [Display(Name = "Property Type")]
    6 references
    public string PropertyType { get; set; }
    6 references
    public string Owner { get; set; }
}
```

More Presentation Layer Changes

6. Add a new controller to the Controllers folder by right clicking the folder and selecting Add > Controller. Select MVC Controller-Empty from the Add New Scaffolded Item dialog:



7. Change the name to RentalsController. You see DefaultController with the word Default highlighted so just type over it:



The dialog box is titled "Add Empty MVC Controller" and has a close button (X) in the top right corner. It contains a label "Controller name:" followed by a text input field. The input field contains the text "DefaultController", with the word "Default" highlighted in blue. At the bottom right of the dialog, there are two buttons: "Add" and "Cancel".

8. The controller is created from a template and the code generated is ready for us to modify:

```
public class RentalsController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

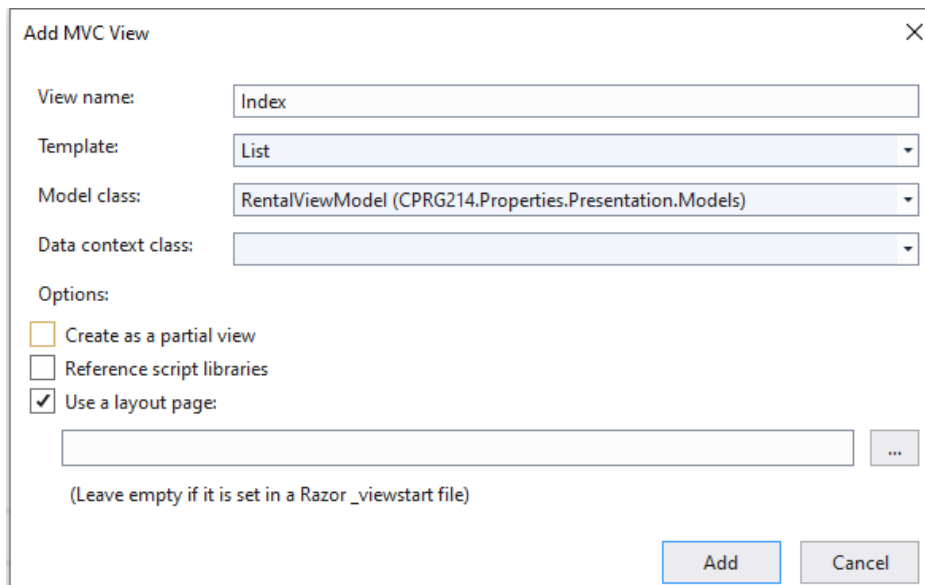
9. Before we modify the controller code, we need to modify the RentalsManager.GetAll method to return navigation property references not just the foreign key values (called the object graph). Unless we do that, we won't be able to read the Owner name or PropertyType style properties because the Owner and Property type references will be null. We need to "include" them when we query the context in the manager class:

```
public static List<RentalProperty> GetAll()
{
    var context = new RentalsContext();
    var rentals = context.RentalProperties.
        Include(r => r.Owner).Include(rp => rp.PropertyType).ToList();
    return rentals;
}
```

10. So now, we can modify the Index action method of the rentals controller:

```
public IActionResult Index()
{
    //this code could go in a seperate class or method
    var rentals = RentalsManager.GetAll();
    var viewModels = rentals.Select( r => new RentalViewModel
    {
        Id = r.Id,
        Address = r.Address,
        City = r.City,
        Province = r.Province,
        Owner = r.Owner.Name,
        Rent = r.Rent.ToString(),
        PropertyType = r.PropertyType.Style
    }).ToList();
    return View(viewModels);
}
```

11. we can create the view for this action method. Click on View(viewModels) and right click and select Add View. In the Add MVC View dialog, change as follows:



The 'Add MVC View' dialog box is shown with the following settings:

- View name: Index
- Template: List
- Model class: RentalViewModel (CPRG214.Properties.Presentation.Models)
- Data context class: (empty)
- Options:
 - ☐ Create as a partial view
 - ☐ Reference script libraries
 - ☒ Use a layout page
- Layout page: (empty text box with a dropdown arrow)
- Buttons: Add, Cancel

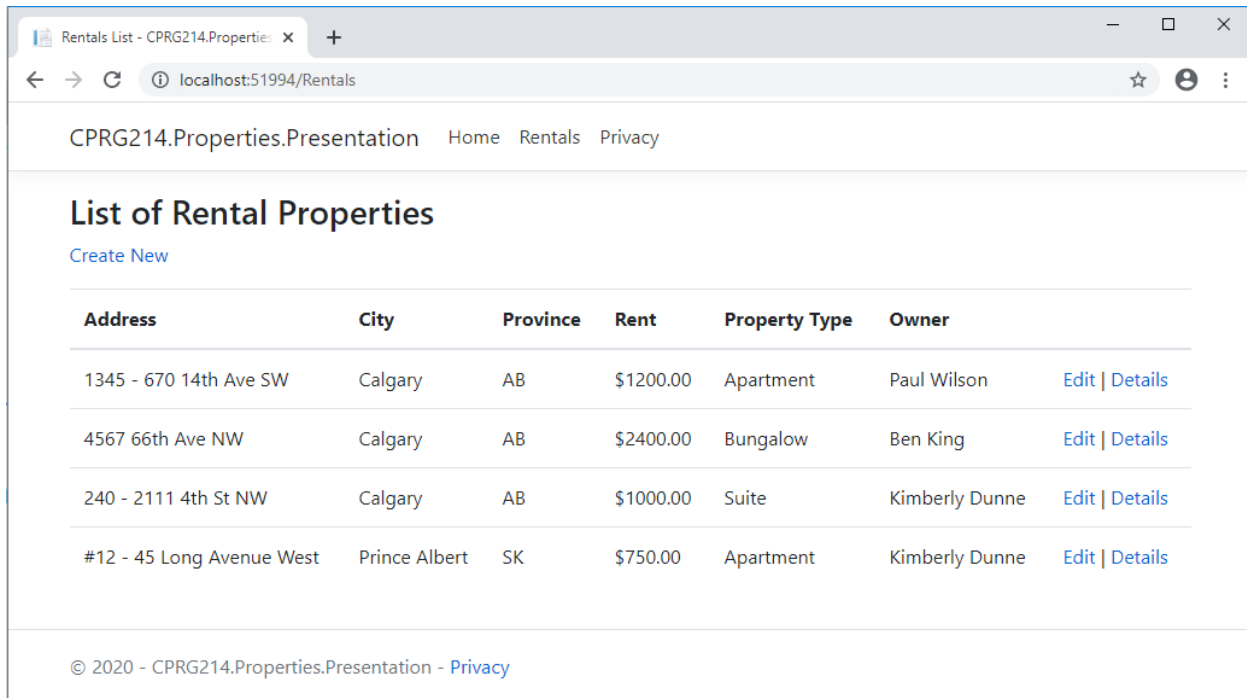
(Leave empty if it is set in a Razor _viewstart file)

This creates a view with same name as action method in a Rentals folder under the Views folder. The List template creates a table markup view based on the model selected.

12. We are almost ready to test our changes, but first add a new list item in the bootstrap menu:

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area=""
    asp-controller="Rentals" asp-action="Search">Rentals</a>
</li>
```

13. Run the application. When the app starts you should see the Rentals menu item. When it is selected, the Index view is called and we see the rental view models displayed.



Address	City	Province	Rent	Property Type	Owner	
1345 - 670 14th Ave SW	Calgary	AB	\$1200.00	Apartment	Paul Wilson	Edit Details
4567 66th Ave NW	Calgary	AB	\$2400.00	Bungalow	Ben King	Edit Details
240 - 2111 4th St NW	Calgary	AB	\$1000.00	Suite	Kimberly Dunne	Edit Details
#12 - 45 Long Avenue West	Prince Albert	SK	\$750.00	Apartment	Kimberly Dunne	Edit Details

As we develop the MVC application, we add controllers for each domain entity and action methods. The Index action method is usually displaying the List of entities and from that page you can link to adding a new entity using the *Create* link or update an existing entity using the *Edit* link. The details link is used to display a read-only view showing the details of the entity—because you may choose to only display a few properties in the Index view and reserve the details view to display all the property values.

14. Development can continue by adding the ability to add new rental properties. For that we need a new action method called Create added to the rentals controller. You can just as easily call it Add or something similar. As long as the view created has the same name as the controller method.

```
public IActionResult Create()
{
    return View();
}
```

15. Code the logic. The view will need to present drop downs so that the user can select the property type and owner when completing the form to add a new rental property.

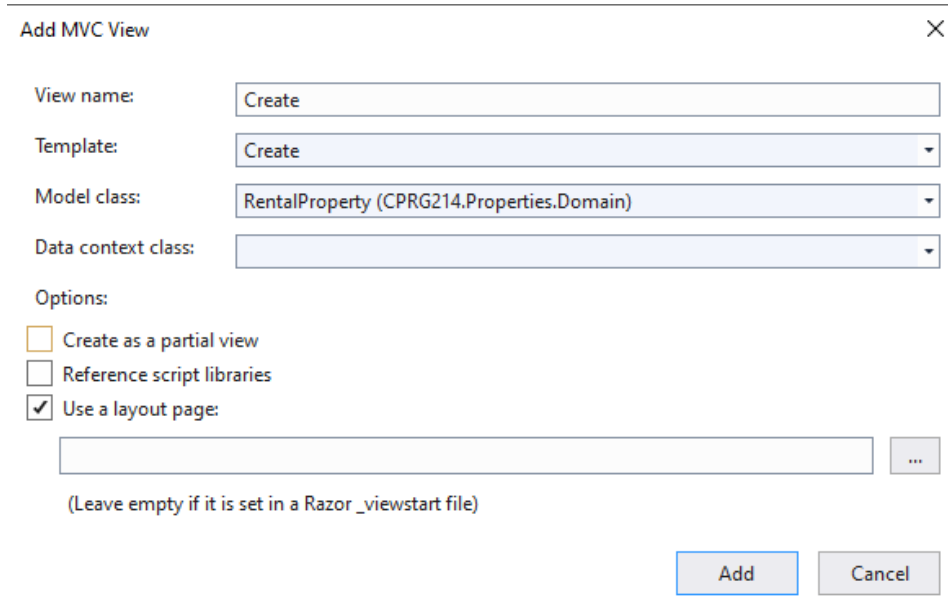
```
public IActionResult Create()
{
    //call the owner manager to get the collection of key value objects
    var owners = OwnerManager.GetAsKeyValuePairs();
    //create a collection of SelectListItem
    var list = new SelectList(owners, "Value", "Text");
    ViewBag.OwnerId = list;

    //create the collection of property types select list items
    var types = PropertyTypeManager.GetAsKeyValuePairs();
    //create a collection of SelectListItem
    var styles = new SelectList(types, "Value", "Text");
    ViewBag.PropertyType = styles;
    return View();
}
```

16. Once the view is completed, and it is submitted, the POST comes to the overloaded Create method with the HttpPost attribute annotating the method:

```
[HttpPost]
0 references
public IActionResult Create(RentalProperty rental)
{
    try
    {
        RentalsManager.Add(rental);
        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
```

17. To create the view for the create action method, click the View() method, right click and add view. This time use the Create template and select the model class as RentalProperty. Double check the view name to ensure it is the same as the action method in the controller.



The 'Add MVC View' dialog box is shown. It has a title bar with a close button (X). The fields are: View name: 'Create', Template: 'Create', Model class: 'RentalProperty (CPRG214.Properties.Domain)', Data context class: (empty). Under Options, there are three checkboxes: 'Create as a partial view' (unchecked), 'Reference script libraries' (unchecked), and 'Use a layout page' (checked). Below the checkboxes is a text box for the layout page name, which is empty, followed by a button with three dots (...). At the bottom are 'Add' and 'Cancel' buttons.

18. Make corrections to the view markup code. The id section required. The html input tags for property types and owners should be replaced with selects and coded as follows:

```
<div class="form-group">
  <label asp-for="PropertyTypeId" class="control-label"></label>
  <select asp-items="ViewBag.PropertyTypeId" asp-for="PropertyTypeId" class="form-control"></select>
  <span asp-validation-for="PropertyTypeId" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="OwnerId" class="control-label"></label>
  <select asp-items="ViewBag.OwnerId" asp-for="OwnerId" class="form-control"></select>
  <span asp-validation-for="OwnerId" class="text-danger"></span>
</div>
```

19. Modify the rentals manager class by adding an *Add* method that can be called from the controller. The rental property is passed from the controller to the manager.

```
public static void Add(RentalProperty rental)
{
    var context = new RentalsContext();
    context.RentalProperties.Add(rental);
    context.SaveChanges();
}
```

20. Run the application and test the changes. Go to rentals and click the create link. In the create view, fill it out and submit. You should be returned to the main rentals view (Index) and see the newly added rental property.

Add a New Rental Property

Address

City

Province

PostalCode

Rent

Property Type

Owner

Create

[Back to List](#)

Address	City	Province	Rent	Property Type	Owner	
1345 - 670 14th Ave SW	Calgary	AB	\$1200.00	Apartment	Paul Wilson	Edit Details
4567 66th Ave NW	Calgary	AB	\$2400.00	Bungalow	Ben King	Edit Details
240 - 2111 4th St NW	Calgary	AB	\$1000.00	Suite	Kimberly Dunne	Edit Details
#12 - 45 Long Avenue West	Prince Albert	SK	\$750.00	Apartment	Kimberly Dunne	Edit Details
333 33rd St West	Saskatoon	SK	\$1200.00	Suite	Zoe Williams	Edit Details