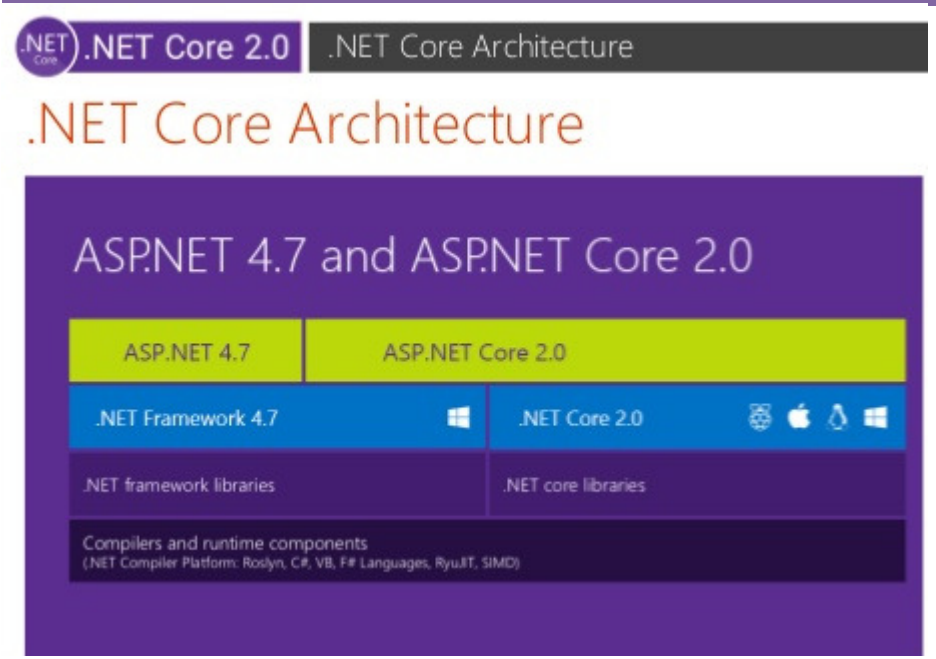
07

ASP.NET Core MVC – part 3 (Complex models again, EF core querying, Migrations, HTML helpers, DI, routing, Login)

Web Development Technologies

COSC2276/2277 Semester 1, 2018



Slides Prepared by: Shekhar Kalra

RMIT UNIVERSITY

Today we will cover

- Migrations and adding new fields
- Complex models again
- Read (related) data
- Eager and Explicit loading
- EF Core querying *in depth*
- Routing
- HTML and Tag helpers
- DI (dependency injection) and,
- Social media authentication

SEGMENT 1

Adding new fields

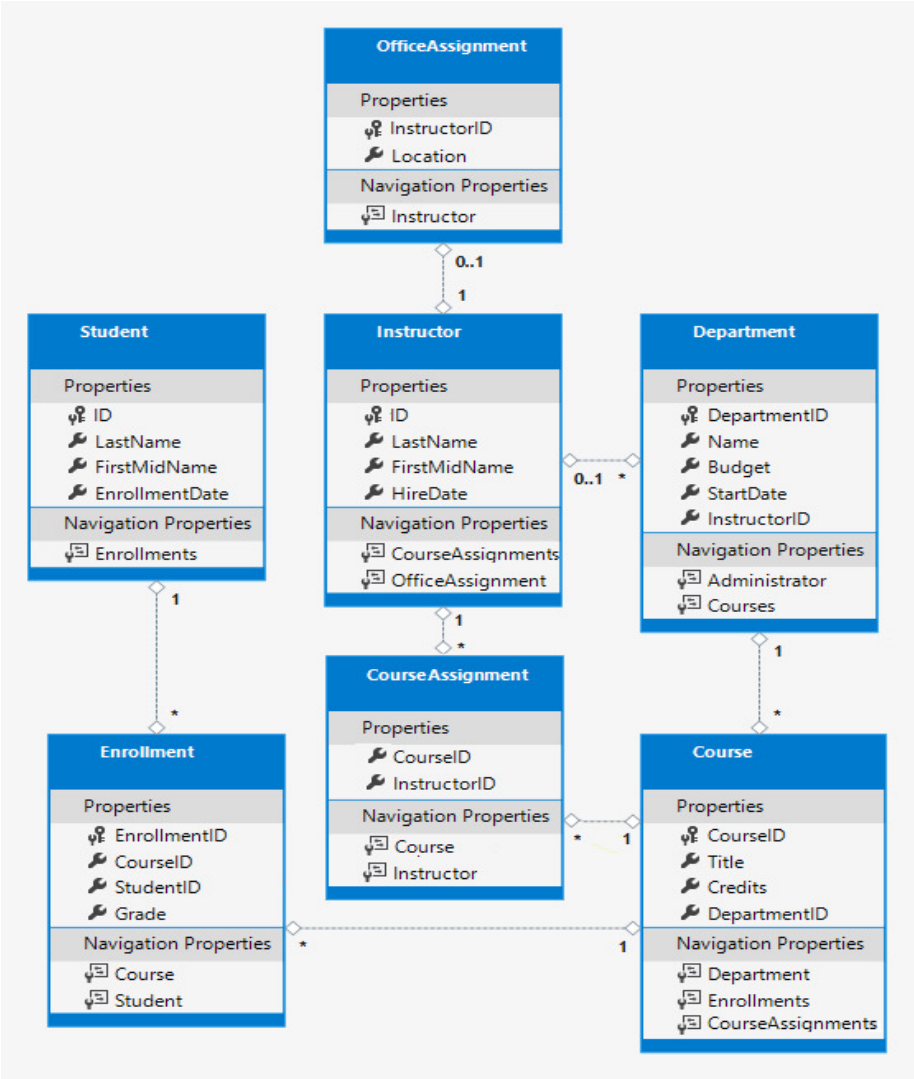
- Lets us change the movie table from the previous week's example by adding a new field into it
- Obviously we will now need to make changes to
  - *Model file*
  - *View file*

  And then update database via migration file
- We will add a "Rating" field to the database
- Instructions based upon https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/new-field?view=aspnetcore-2.1
- Example1
- It uses a feature called *Code First Migrations*

Complex models- 1

- We looked at an example of a complex model during previous week's lecture and
  - You are covering a similar exercise in this week's tutorial # 7 (*exercise 2*)
- Let us now look at an example of another complex model which requires us to write several model files and establish complex relationships as:
  - PK-FK
  - Composite key etc.

Complex models- 2



RMIT UNIVERSITY

Complex models- 3

- We will build a web application around the ER diagram shown on the previous slide and then query the data
- This is based upon
  - https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/complex-data-model?view=aspnetcore-2.1
- In this example we will
  - Create database tables and
  - Seed them
- Example2

Read related data

- Now that we have the database ready, we can do more
- Read and update related data
- This involves joining of tables in the database
- But remember we are not dealing with tables, our concern is with entities only
  - Such as dealing with department entity and all related course entities
- There are two ways of doing this: *eager* and *explicit* loading
- Example3
- *We will look at dealing with updating related data next week*

Eager loading

- When the entity is read, related data is retrieved along with it.
- This typically results in a single join query that retrieves all of the data that's needed.
- You specify eager loading in Entity Framework Core by using the *Include* and *ThenInclude* methods.

```
var departments = _context.Departments.Include(d => d.Courses);
foreach (Department d in departments)
{
    foreach(Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department entities and related Course entities

RMIT UNIVERSITY

Explicit loading

- When the entity is first read, related data isn't retrieved.
- You write code that retrieves the related data if it's needed.
- As in the case of eager loading with separate queries, explicit loading results in multiple queries sent to the database.
- The difference is that with explicit loading, the code specifies the navigation properties to be loaded

```
var departments = _context.Departments;
foreach (Department d in departments)
{
    _context.Entry(d).Collection(p => p.Courses).Load();
    foreach (Course c in d.Courses)
    {
        courseList.Add(d.Name + c.Title);
    }
}
```

Query: all Department rows

Query: Course rows related to Department d

RMIT UNIVERSITY

SEGMENT 2

EF CORE QUERYING

EF Core queries

- Queries are specified using Language Integrated Query (LINQ) using *query syntax* or *method syntax*.
  - Query syntax shares a resemblance with SQL.

    var data = from a in Authors select a orderby a.LastName

  - Method syntax uses chained method calls. Many of the method names also resemble SQL.

    var data = context.Authors.OrderBy(a => a.LastName);

EF Core query types

1. Queries that retrieve single object
2. Queries that retrieve multiple objects
3. Filtering and ordering queries
4. Grouping
5. Include related data
6. *No tracking queries: self-read*
7. *Queries that return non-entity type: self-read*

EF Core query types

- Queries that retrieve single object: performed using variations of the First, FirstOrDefault, Single, SingleOrDefault and Find methods.
  - If you expect at least one record to match the criteria, you can use the First method.
  - If there is a possibility of no records matching the criteria, use the FirstOrDefault method, which will return null, the default, in the event of no records being found.

```
var author = context.Authors.First();
```

Resulting SQL:

```
SELECT TOP(1) [a].[AuthorId], [a].[FirstName], [a].[LastName]
FROM [Authors] AS [a]
```

EF Core query types

- Queries that retrieve single object: performed using variations of the First, FirstOrDefault, Single, SingleOrDefault and Find methods.
  - The Single and SingleOrDefault methods are used to return a single record where only one should match the criteria specified

```
var author = context.Authors.Where(a => a.AuthorId == 1).Single();

var author = context.Authors.Single(a => a.AuthorId == 1);
```

Both approaches result in identical SQL being generated:

```
SELECT TOP(2) [a].[AuthorId], [a].[FirstName], [a].[LastName]
FROM [Authors] AS [a]
WHERE [a].[AuthorId] = 1
```

EF Core query types

- Queries that retrieve multiple objects: are only executed against a database when the data is iterated over.
  - This is known as *deferred execution*. Data is iterated over when you use a foreach loop, or a finalising method on the query such as ToList, Sum or Count.
  - You can also use what is known as *immediate execution*

```
var products = context.Products; // define query
foreach(var product in products) // query executed and data obtained from database
{
    ...
}
```

```
var products = context.Products.ToList(); // define query and force execution
```

EF Core query types

- Filtering and ordering queries
  - The Where method is the principal method for filtering results
  - The OrderBy, OrderByDescending, ThenOrderBy and ThenOrderByDescending methods are used for specifying the order of results

```
var products = context.Prducts.Where(p => p.CategoryId == 1 && p.UnitsInStock < 10);
```

```
var products = context.Products.OrderBy(p => p.ProuctName);
vat categories = context.Categories.OrderBy(c => cCategoryName).ThenOrderBy(c => c.CategoryId);
```

EF Core query types

- Grouping
  - The GroupBy method is used to group results
  - This results in a collection of types that implement the IGrouping interface. The types have a Key property, which holds the value of the value that was used for grouping
  - If you want to use multiple properties to group by, you will use an anonymous type to represent the Key
  - Currently done *in memory- t*he data is obtained from the database and then the grouping is performed in the client application by C# code

EF Core query types

- Grouping

```
var groups = context.Products.GroupBy(p => p.CategoryId);
foreach(var group in groups)
{
    //group.Key is the CategoryId value
    foreach(var product in group)
    {
        // you can access individual product properties
    }
}
```

```
var groups = context.Products.GroupBy(p => new {Supplier = p.SupplierId, Country = p.CountryId});
```

```
foreach(var group in groups)
{
    //group.Key.SupplierId is the SupplierId value
    //group.Key.Country is the CountryId value
}
```

- Complex Grouping requires the use of embedding Raw SQL queries

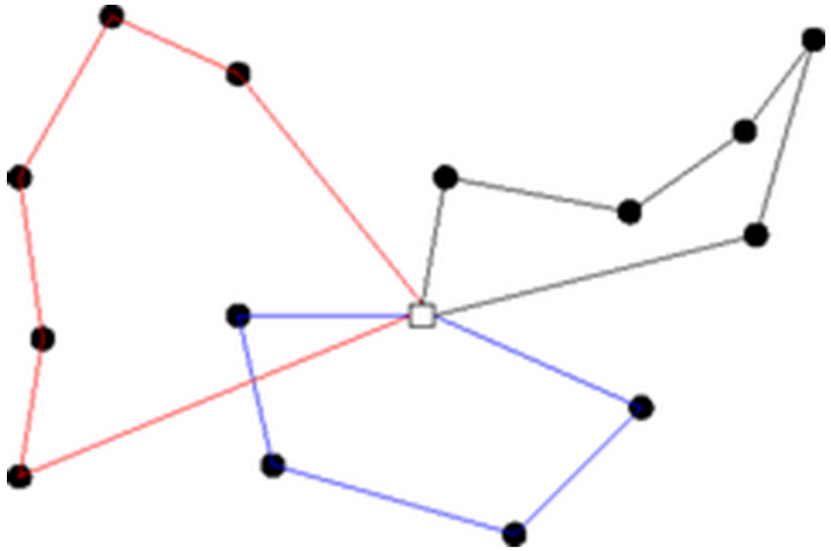
EF Core query types

- Include related data
  - The Include method is used to eagerly load related data. You pass in the navigation property that you want to include in the result set.
  - You can use the ThenInclude method to retrieve data from second and subsequent level relationships

```
var authors = context.Authors
                     .Include(a => a.Books)
                        .ThenInclude(b => b.Publisher)
                     .ToList();
```

EF Core query

- Here are some great website that will give you more information:
  - Dixin's Blog
    - *[https://weblogs.asp.net/dixin/entity-framework-core-and-linq-to-entities-4-query-methods]*



ROUTING

Routing

- During our discussion in the previous week(s), we learned about the URL routing –

  http://mysite.com/Admin/Index

  - 1st segment of Url = Controller
  - 2nd segment = Action and the
  - 3rd argument Id is optional

    http://mysite.com/{controller}/{action}/{Id}
- Route information can be found in Startup.cs file

Routing (Startup.cs)

- The default route pattern:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

- You can define your own route pattern and customise routes.
- You can even generate URLs

Customising Routes

- Lots of clever things can be done with route customisation, for example, you can create a route like

  files/{filename}.{ext?}

  This route matches a filename or an optional extension.

- Perhaps you want a dateTime in the URL, you can make a route like:

  person/{dob:datetime}

- Or perhaps a Regular Expression for a particular pattern like this:

  user/{pattern:regex(d{3}-d{2}-d{4})}

Customising Route

- There are two ways of customising routes
  - Globally (*ie. in Startup.cs*) and,
  - Use of attributes in a controller file
- Let us now look at two examples
  - Example4: CustomisingRouteGlobal
  - Example5: RoutesInControllers
- These examples will give you an idea of how routes can be customised as per your need(s).

Constraints in Routes-1

- There are lot of interesting things you can do with a route such as- generate custom routes, even constraint them using regular expressions

This example adds route constraints and data tokens:

C#

```
routes.MapRoute(
    name: "us_english_products",
    template: "en-US/Products/{id}",
    defaults: new { controller = "Products", action = "Details" },
    constraints: new { id = new IntRouteConstraint() },
    dataTokens: new { locale = "en-US" });
```

This template will match a URL path like `/en-US/Products/5` and will extract the values `{ controller = Products, action = Details, id = 5 }` and the data tokens `{ locale = en-US }`.

Constraints in Routes-2

- ASP.NET Core MVC Routing supports various constraints:

```
private static IDictionary<string, Type> GetDefaultConstraintMap()
{
    return new Dictionary<string, Type>(StringComparer.OrdinalIgnoreCase)
    {
        // Type-specific constraints
        { "int", typeof(IntRouteConstraint) },
        { "bool", typeof(BoolRouteConstraint) },
        { "datetime", typeof(DateTimeRouteConstraint) },
        { "decimal", typeof(DecimalRouteConstraint) },
        { "double", typeof(DoubleRouteConstraint) },
        { "float", typeof(FloatRouteConstraint) },
        { "guid", typeof(GuidRouteConstraint) },
        { "long", typeof(LongRouteConstraint) },

        // Length constraints
        { "minlength", typeof(MinLengthRouteConstraint) },
        { "maxlength", typeof(MaxLengthRouteConstraint) },
        { "length", typeof(LengthRouteConstraint) },

        // Min/Max value constraints
        { "min", typeof(MinRouteConstraint) },
        { "max", typeof(MaxRouteConstraint) },
        { "range", typeof(RangeRouteConstraint) },

        // Regex-based constraints
        { "alpha", typeof(AlphaRouteConstraint) },
        { "regex", typeof(RegexInlineRouteConstraint) },

        { "required", typeof(RequiredRouteConstraint) },
    };
}
```

Areas

- The MVC pattern separates the model (data) logic of an application from its presentation logic and business logic.
- However some applications can have a large number of controllers, and each controller can be associated with several views.
  - For these types of applications, the default ASP.NET MVC project structure can become difficult to maintain.
- To accommodate large projects, ASP.NET MVC Core lets you partition Web applications into smaller units that are referred to as areas.
- Example6: AreasDemo
- Some interesting discussions:
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/areas?view=aspnetcore-2.1
  - [https://tahirnaushad.com/2017/08/25/asp-net-core-2-0-mvc-areas/]

SEGMENT 3

HTML helpers- 1

- HTML Helpers are helpers that create HTML code. You can use them directly within the view using Razor syntax.
- Html is a property of the view base class RazorPage and is of type IHtmlHelper.
- The class InputExtensions defines HTML Helper methods to create check boxes, password controls, radio buttons, and text box controls.
- HTML Helper methods BeginForm, Label, and CheckBox. BeginForm starts a form element. There's also an EndForm for ending the form element.

HTML helpers- 2

- Most HTML Helper methods have overloads in which you can pass any HTML attributes. For example, the following TextBox method creates an input element of type text.

```
@Html.TextBox("text1", "input text here",
  new { required="required", maxlength=15, @class="CSSDemo" });
```

- Example7: Razor syntax demo
- Example8: HTMLhelpers

Tag helpers - 1

- MVC Core offers another type of helpers that can be used instead of HTML helpers i.e. Tag helpers.
- With Tag Helpers you don't write C# code mixed with HTML; instead you use HTML attributes and elements that are resolved on the server.
- Many of the ASP.NET MVC Tag Helpers have the prefix *asp-*, so you can easily see what's resolved on the server. These attributes are not sent to the client but instead are resolved on the server to generate HTML code.

Tag helpers - 2

- To have the Tag Helpers available with all views, add the addTagHelper statement to the shared file _ ViewImports.cshtml (*where is this file located?*)

```
@addTagHelper *, Microsoft.AspNet.Mvc.TagHelpers
```

- Example9: Taghelpers
- Why use a tag helper instead of an HTML helper?
- *You can even create a custom tag helper, this is left as a self exercise*.

LOGIN-LOGOUT-REGISTRATION

User authorization

- User login, logout and registration are common features in any website.
- There are various ways to implement above in an ASP.NET MVC Core application
  - Writing your own (*this is an advanced discussion and outside the context of this course; requires you to understand Identity API*)
  - Use 3rd party authentication such as Google or Facebook (*we will follow this path*)

Implementing Google login - 1

- Core MVC relies on an external framework- OAuth 2
- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean.
- It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

Implementing Google login - 2

- Complete details of how internals of OAuth 2 works is outside the context of this course.
  - If you are keen read this – [https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2]
  - You really don't need that kind of depth at this stage
- Let us now implement Google login in an ASP.NET Core MVC project.
- Example10: GoogleLogin
- You will not find this project in code archive, create your project following the steps at https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/
- Left as a self exercise BUT please complete this as you will need it for assignment 2

Dependency Injection (DI)

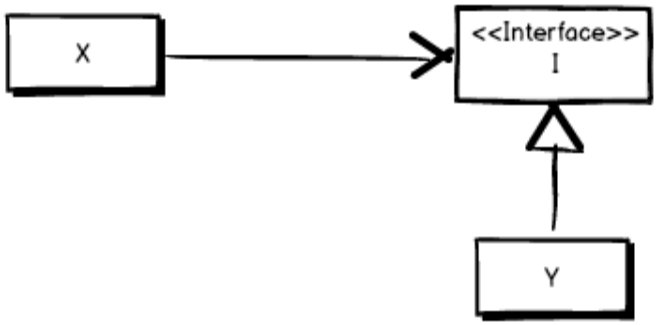
- Dependency injection (DI) is a technique for achieving loose coupling between objects and their collaborators, or dependencies.
- Rather than directly instantiating collaborators, or using static references, the objects a class needs in order to perform its actions are provided to the class in some fashion.
- Most often, classes will declare their dependencies via their constructor.

Dependency Injection (DI)

The class, X, needs the class, Y, to accomplish something. That's all good, but does X really need to know that it uses Y?

Ref: Images used for learning purposes from http://joelabrahamsson.com/

Y implements I and X uses an instance of I. While it's quite possible that X still uses Y what's interesting is that X doesn't know that. It just knows that it uses something that implements I.



- Let us look at an example:
  - Example 11: DI_Demo

RMIT UNIVERSITY

Core MVC DI

- Let us go back to Startup.cs
- ASP.NET Core includes a simple built-in container (represented by the IServiceProvider interface) that supports constructor injection by default, and ASP.NET makes certain services available through DI.
- ASP.NET's container refers to the types it manages as services.
- *What does that mean?*

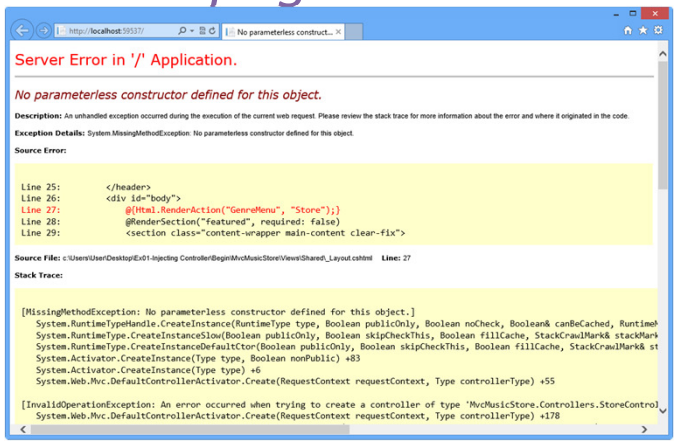
MVC Core Error handling

- Let us know learn how error handling is implemented at a global level in an MVC Core project.
- In the Visual Studio project, you will already see an in-built package: Microsoft.AspNetCore.Diagnostics
- Configure method in Startup.cs has an important line that you will need in order to use the above package-

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
    app.UseBrowserLink();
}
```

MVC Core Error handling

- Let us know look at an example to see how cutsomised error pages are created in an MVC Core application.
  - *What are customised error pages?*
  - *Why not just rely on the system error pages?*



- Example12: CustomError_Demo

Adding searching and paging features

- Most web applications will require you to add extra features such as – searching, paging data for better display, sorting, grouping and filtering.
- While sorting is simple to implement codewise, Paging can be accomplished in two ways-
  - Writing long code
  - Using 3rd party package available via NuGet
- You will complete an exercise on paging, searching, grouping and filtering in tute/lab sheet 8.

References

- https://docs.microsoft.com/en-us/aspnet/core/
- https://docs.microsoft.com/en-us/aspnet/core/getting-started

Assignment 2 is online

- Deadline: Sunday 20.05.2018 (11:59 pm)
- Face2Face demo: 21-25 May 2018
- Let us have a look.

April 25 is a public holiday & week 8 lecture

- Next Wednesday is a public holiday
- We will have our lecture held on
  - THURSDAY 26/04/2018
  - 2:30 – 4:30 pm
  - VENUE: 080.06.005
- From week 9 we will revert to the normal schedule.