

CIS 341

# Interactive Web Programming

Week 9 – Class meeting 2

# Agenda

- Exercise: Discuss project domain object model structure
- Revisit routing vs. model binding
- Entity Framework Core tutorial – basics of setting up model persistence

# Exercise

- Share the domain object model structure.
- Consider the relationships between the objects:
  - One-to-one (e.g., User <-> Shopping Cart)
  - One-to-many (e.g., User <-> Order)
  - Many-to-many (e.g., Product <-> Shopping Cart)
- Are all objects related? If not, why not?
- Which properties in the objects are required? Have other validation requirements?

# Routing vs. model binding

- The distinction was incorrectly explained in the lecture materials.
- Routing is the process of selecting an endpoint – model binding is the process of extracting data from the request.
- Model binding does not affect routing since it takes place after the endpoint has been selected.
  - In other words, action method parameters or their types do not affect which endpoint is selected.
- Route parameter constraints should be used to distinguish between two similar looking routes by directing them to different endpoints.
- Input validation should happen in the controller based on the model validation constraints / action method parameter types.

# Distinguishing between routes with parameter constraints

```
endpoints.MapControllerRoute(  
    name: "postsByID",  
    pattern: "/post/{id:int}",  
    new { controller = "Posts", action = "PostsByID" });
```

```
endpoints.MapControllerRoute(  
    name: "postsByName",  
    pattern: "/post/{id:alpha}",  
    new { controller = "Posts", action = "PostsByName" });
```

# Action method parameter values do not affect which endpoint is selected

```
public IActionResult About(int id)
{
    return View();
}
```

```
public IActionResult About(int id, string name)
{
    return View();
}
```

**An unhandled exception occurred while processing the request.**

AmbiguousMatchException: The request matched multiple endpoints. Matches:


RoutingTest.Controllers.HomeController.About (RoutingTest)

RoutingTest.Controllers.HomeController.About (RoutingTest)


Microsoft.AspNetCore.Routing.Matching.DefaultEndpointSelector.ReportAmbiguity(CandidateState[] candidateState)

# Input validation in the controller

```
public IActionResult About(int id)
{
    if(!ModelState.IsValid)
    {
        ViewData["Message"] = "Not Found!";
        return View();
    } else
    {
        return View();
    }
}
```



The diagram shows a blue arrow pointing from the URL `https://www.mysite.com/Home/About/abc` to the `if(!ModelState.IsValid)` branch of the `About` method.



The diagram shows a blue arrow pointing from the URL `https://www.mysite.com/Home/About/111` to the `else` branch of the `About` method.

# Entity Framework Core

Tutorial & hands-on



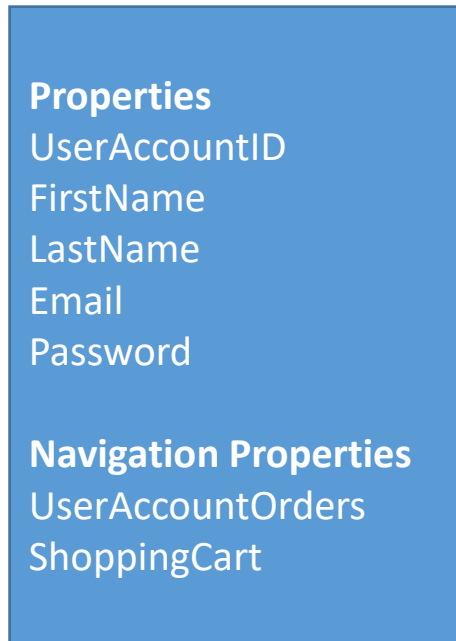
# EF Core tutorial

- New > Project > ASP.NET Core Web Application
- Create entity classes (data model)
- Create database context
- Register database context
- Initialize (seed) database with test data
- View database structure

# Create entity classes

- Entity classes contain two types of variables:
  - **Properties**: define table columns
  - **Navigation Properties**: map entities related to the current entity – define relations to other tables that can be used to query related data
- The EF Core relationship model is described in more detail in <https://docs.microsoft.com/en-us/ef/core/modeling/relationships>
- Sample ER model we'll work through in this tutorial:
  - User Account
  - Order
  - Shopping Cart
  - Book

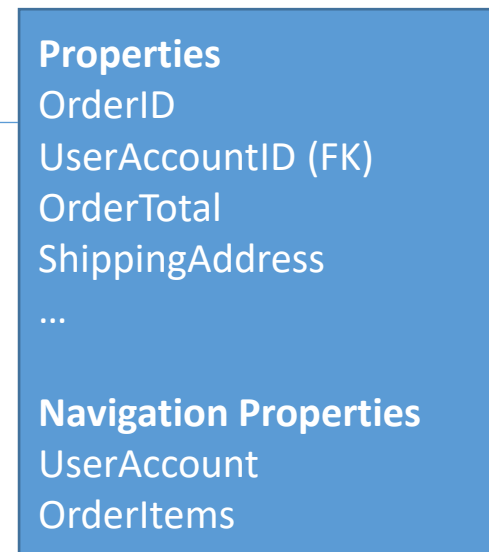
## UserAccount



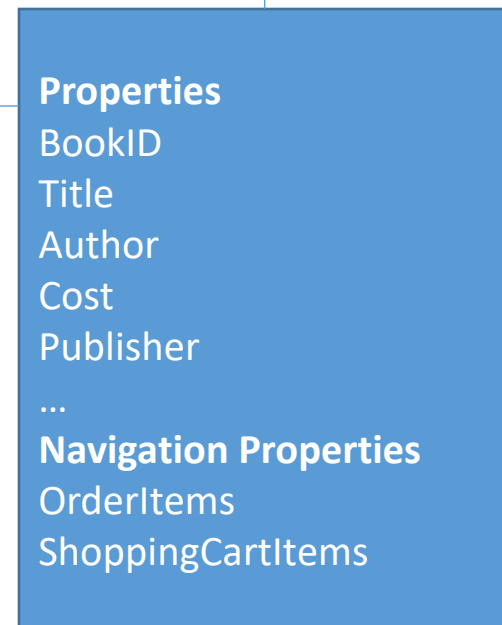
## ShoppingCart



## Order



## Book



Many-to-many relations require join tables (join entity types) to be created.

# Create database context

- Add > New Folder > Data
- In *Data* folder > Add > Class > `BookstoreContext.cs`
  - Should inherit from `DbContext` class
- Define class constructor and its options.
- Define `DbSet` properties:
  - Each `DbSet` will correspond to a table in the generated database.
  - Convention is to use a plural of the entity class name, e.g., `UserAccount` → `UserAccounts`.

# Register database context

- Register via dependency injection at app startup:
  - `Startup.ConfigureServices > AddDbContext()`
  - Requires configuration options – we will use `UseSqlServer` to persist in a SQL Server Express LocalDB (.mdf file).
- Edit `appsettings.json` to add connection string.
- LocalDB will create the database file in `C:\Users\<username>\.`

# Create database via migrations

- It is possible to now create the database based on the model objects by using `Add-Migration` and `Update-Database` in the Package Manager Console.
  - Note: Need to install EF Core Tools package first:  
`Install-Package Microsoft.EntityFrameworkCore.Tools`
- Migrations allow changes to model entities to be propagated to the database schema.
- You cannot use migrations and the technique described in the following slides at the same time.
  - If you want to use migrations together with database content initialization, use `context.Database.Migrate()` instead of `EnsureCreated()`.

# Initialize database with test data

- In Data folder > Add > Class > `DbInitializer.cs`
  - Defined as static class
- Add static method `Initialize(DbContext context)`
  - Method should call `context.Database.EnsureCreated()` to create database if it does not exist based on the **current entity objects** – will ignore migrations.
  - Method should check if database contains data and terminate if this is the case – no need to initialize.
  - Use `context.<DbSetName>.Add(entity)` to add new rows.
  - Use `context.<DbSetName>.SaveChanges()` to commit to database.
- Modify `Main` method in `Program` class
  - Call `Initialize()` with the `DbContext` provided via dependency injection.

# Errors?

Introducing FOREIGN KEY constraint

'FK\_UserAccountOrders\_UserAccounts\_UserAccountID' on table  
'UserAccountOrders' may cause cycles or multiple cascade paths.  
Specify ON DELETE NO ACTION or ON UPDATE NO ACTION

- The problem arises because of the Entity Framework behavior for handling entity removal when setting up foreign key constraints.
  - The error message essentially means that entities in the UserAccountOrders table could be deleted either when a referencing UserAccount entity is removed, or when a referencing Order entity is removed.
  - SQL Server does not allow multiple cascades.
- There are a couple of solutions:
  - Add a trigger: <https://www.mssqltips.com/sqlservertip/2733/solving-the-sql-server-multiple-cascade-path-issue-with-a-trigger/>
  - Disable cascading deletions in the foreign key constraints – but now you have to make sure to manually clean up any relations that are potentially left hanging in the database!



# View database

- View > SQL Server Object Explorer > select database > Tables
- SSOX provides functionality for
  - designing and editing table structure
  - viewing table content
  - inserting and deleting rows
  - dropping tables and the database, If needed
- If you need to recreate database initialization after making changes to the entities
  - delete database in SSOX
  - after to updating the `DbInitializer` class, run app to create database and reseed

# EF Core conventions

- Names of `DbSet` **objects** are used as table names.
  - Entities not referenced by a `DbSet` object will have their class name used as table name.
- Model entity property names are used for table column names.
- Entity properties that are named `ID` or `<classname>ID` are recognized as primary key properties.

# EF Core conventions

- A property is interpreted as a foreign key property if it is named
  - `<navigation property name><primary key property name>`
  - `<primary key property name>` of related entity
- This behavior can be overridden by explicitly specifying table, column and keys using attributes.
- Join tables for many-to-many relationships are commonly named `EntityName1EntityName2`.
  - It is often a better practice use a descriptive name for the relation that represents its natural name in the business domain, e.g., `OrderProducts` → `Shipment`.

# What we did not cover this time

- Creating the full entity relationship model for the course project.
  - The customer-facing side should give a reasonable idea of how to tackle the employee entities.
- Creating CRUD functionality for managing and using the entities in the web application.
- Reading and updating related data.
- These will be addressed in Assignment 5 and the final project submission.

# Topics for next week

- Front-end vs. back-end validation
- Showing model validation errors in the View
- Continue working with EF Core concepts
  - Reading related data
  - Updating related data