ASP.NET CORE

MVC

issntt@nus.edu.sg

Objectives

At the end of this lesson, students will be able to

- Describe the MVC design pattern
- Compare and contrast the roles of Model, View and Controller
- Analyze how a web app implemented with MVC handles some typical business scenarios
- Explain why MVC design pattern is so popular

Topics

- **Design Patterns**
- The MVC Design Pattern
  - Model
  - View
  - Controller

Questions

Can we reuse **codes**? How?

Can we reuse a **class**? How?

Can we reuse a **design solution**? How?

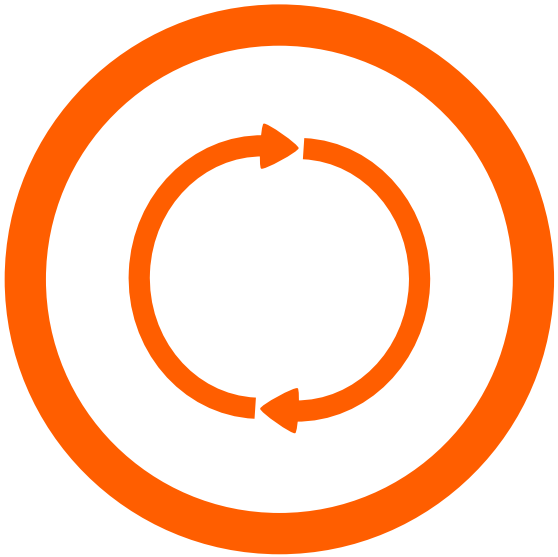


Image by tFity from Pixabay

Design Patterns

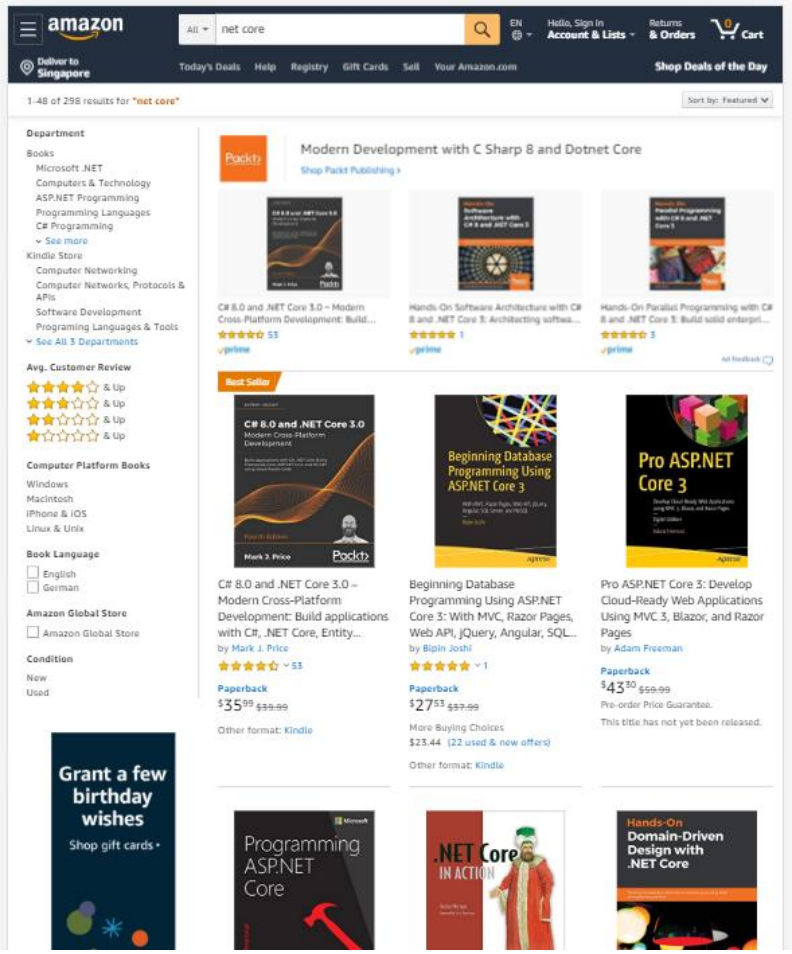
- **Repeatable solutions** to **recurring design problems**
- Can be used in **many types** of systems
  - Windows, MacOS, Linux
  - Desktop, Web, Mobile apps
  - ASP.NET Core, J2EE frameworks

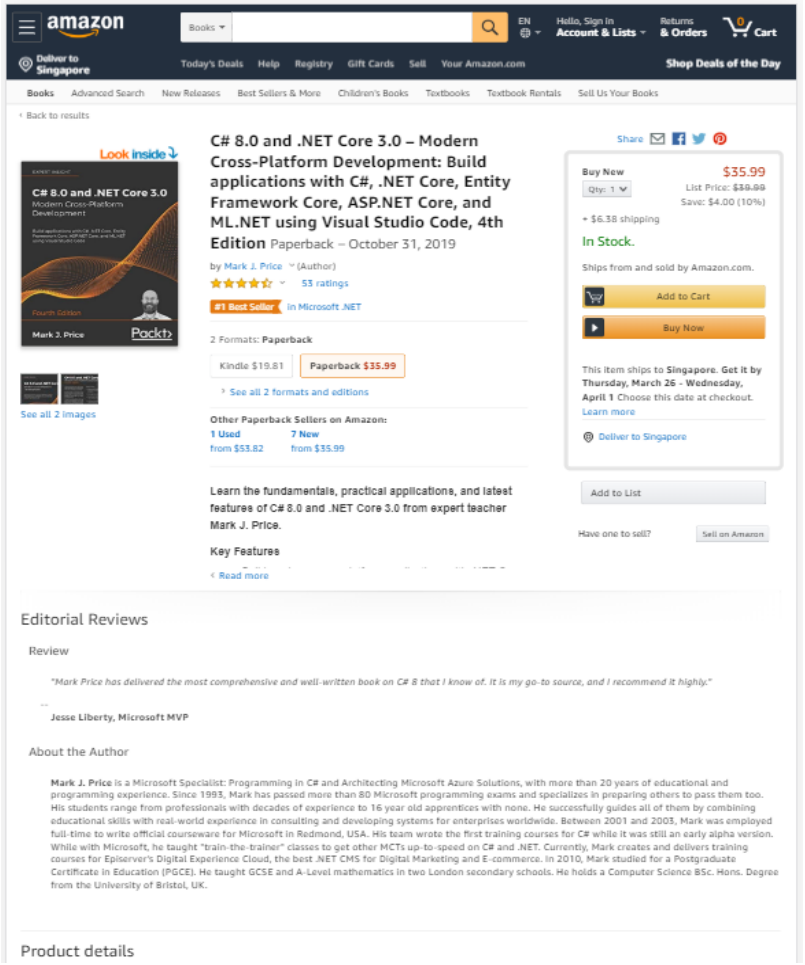
Topics

- Design Patterns
- **The MVC Design Pattern**
  - Model
  - View
  - Controller

Problem

Given requests in these 2 scenarios:





https://www.amazon.com/s?k=.net+core

https://www.amazon.com/8-0-NET-Core-3-0-Cross-Platform/dp/1788478126/

Problem

We have learnt OOPCS and Database, let's say that

1. There are **two classes** in the server to **handle requests**
2. The book data is **stored in the database**

What **tasks** does the server need to do to **serve** the **requests** in those scenarios?

Try to imagine as many tasks as possible

```
public class Handler1 {
    public void MethodHandler1_1() {
        // Handle a type request
    }
    public void MethodHandler1_2() {
        // Handle another type of request
    }
}
```

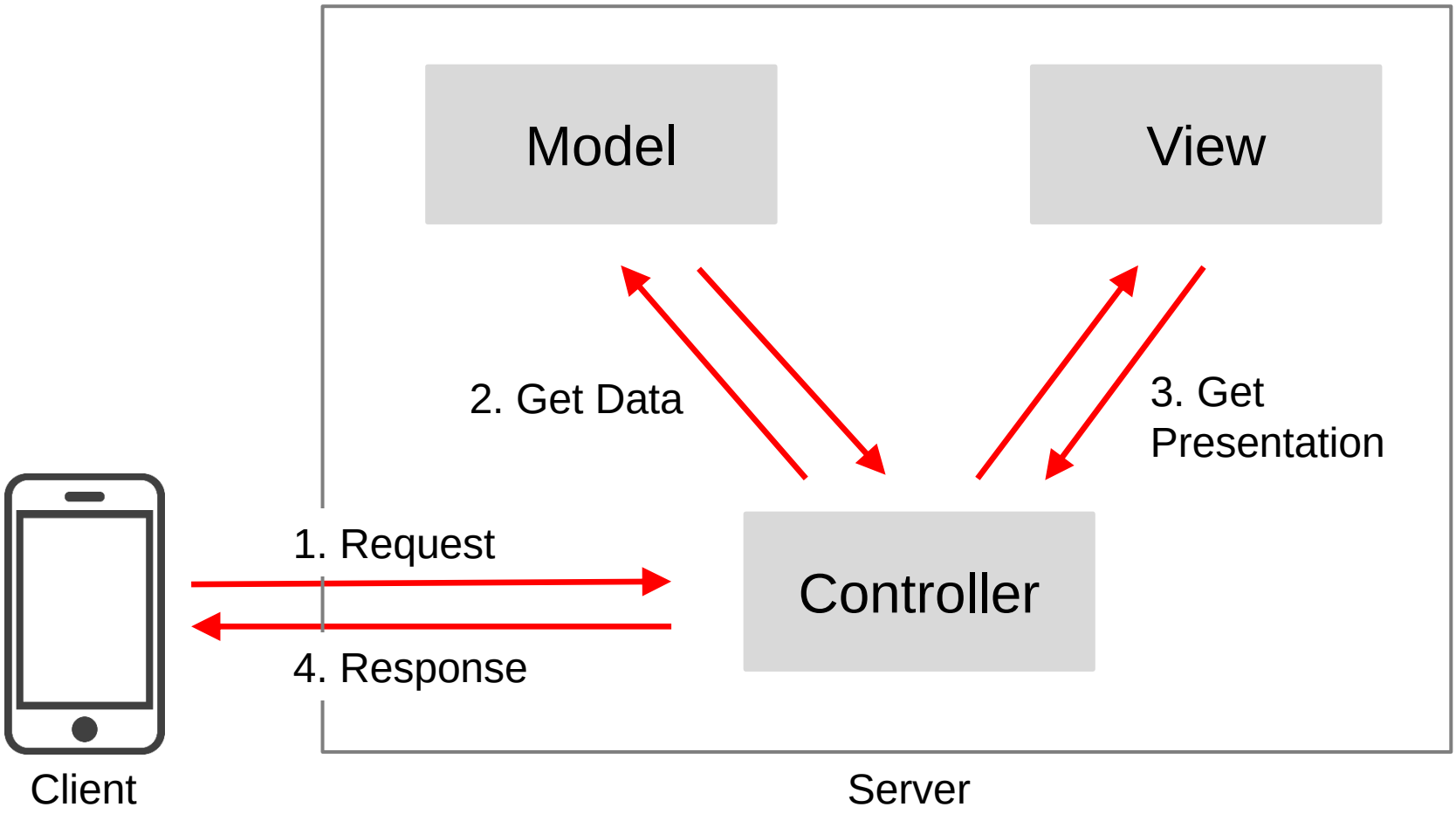
```
public class Handler2{
    public void MethodHandler2_1() {
        // Handle another type of request
    }

    public void MethodHandler2_2() {
        // Handle another type of request
    }
}
```

The MVC Pattern

https://www.youtube.com/watch?v=DUg2SWWK18I

The MVC Pattern

Separates an application into **3 groups** of **components**



Within **each** Model, View, Controller, there may be **many classes** sharing the required tasks

Topics

- Design Patterns
- **The MVC Design Pattern**
  - **Model**
  - View
  - Controller

Model

Among M, V and C, Models have most responsibilities. One of them is to **represent** the **state/data** of the app

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Required, StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100), DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }
}
```

Model

Models also take care of **business logics**

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100), DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [StringLength(5)]
    public string Rating { get; set; }
}
```

Model

Models also deal withs **persisting** and **retrieving** data

```
public class MvcMovieContext : DbContext
{
    public MvcMovieContext(DbContextOptions<MvcMovieContext> options)
        : base(options)
    {
    }

    public DbSet<Movie> Movie { get; set; }
}
```

Class *DbContext* helps **create**, **retrieve**, **update** and **delete** the **data** to/from database

Topics

- Design Patterns
- **The MVC Design Pattern**
  - Model
  - **View**
  - Controller

View

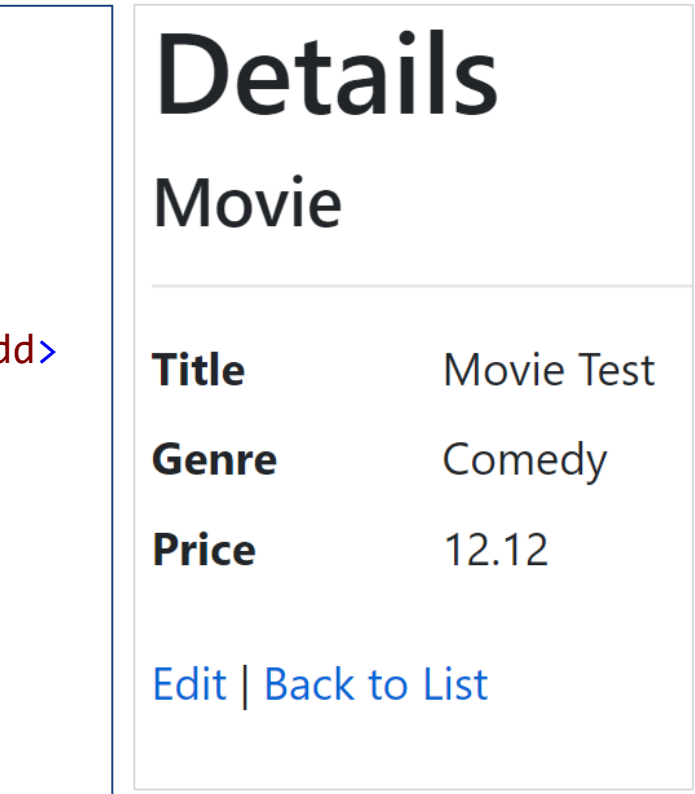
Given some **data**, the main responsibility of a View is to **generate** some **presentation** of the data

```
@model MvcMovie.Models.Movie
@{
    ViewData["Title"] = "Details";
}
<h1>Details</h1>
<div>
    <h4>Movie</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Title)</dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Title)</dd>
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Genre) </dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Genre) </dd>
        @* ...more code *@
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

View

The output HTML and how it is displayed in Browsers

```
<h1>Details</h1>
<div>
    <h4>Movie</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">Title</dt>
        <dd class="col-sm-10">Movie Test</dd>
        <dt class="col-sm-2">Genre</dt>
        <dd class="col-sm-10">Comedy</dd>
        <dt class="col-sm-2">Price</dt>
        <dd class="col-sm-10">12.12</dd>
    </dl>
</div>
<div>
    <a href="/Movies/Edit/1">Edit</a> |
    <a href="/Movies">Back to List</a>
</div>
```



For some given data, e.g., a Movie, **how many** possible **representations** are there?

View

Views may also contain the **user-interactions logic** and **convey user intentions** to the Controller

```
@model MvcMovie.Models.Movie
@{
    ViewData["Title"] = "Details";
}
<h1>Details</h1>
<div>
    <h4>Movie</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Title)</dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Title)</dd>
        <dt class="col-sm-2">@Html.DisplayNameFor(model => model.Genre) </dt>
        <dd class="col-sm-10">@Html.DisplayFor(model => model.Genre) </dd>
        @* ...more code *@
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

Topics

- Design Patterns
- **The MVC Design Pattern**
  - Model
  - View
  - **Controller**

Controller

Controllers **handle user actions**. For example, method *`Edit()`* will be executed when users edit a movie

```
public class MoviesController : Controller {
   private readonly MvcMovieContext _context;

   public IActionResult Edit(int id, Movie movie) {
      if (id != movie.Id) {
         return NotFound();
      }
      if (MovieService.IsValid(movie)) {
         _context.Update(movie);
         _context.SaveChangesAsync();

         return RedirectToAction(nameof(Index));
      }
      return View(movie);
   }
}
```

Controller

Controllers **select** which **models** and asks them to perform **sub-tasks** (e.g., validate, get and store necessary data)

```
public class MoviesController : Controller {
   private readonly MvcMovieContext _context;

   public IActionResult Edit(int id, Movie movie) {
      if (id != movie.Id) {
         return NotFound();
      }
      if (MovieService.IsValid(movie)) {
         _context.Update(movie);
         _context.SaveChangesAsync();

         return RedirectToAction(nameof(Index));
      }
      return View(movie);
   }
}
```

At this stage, don't care about what *MvcMovieContext* or *MovieService* is. Just know that they are all Models

Controller

Controllers **select** which **views** and ask them to generate the **respective representation** of the data

```
public class MoviesController : Controller {
   private readonly MvcMovieContext _context;

   public IActionResult Edit(int id, Movie movie) {
      if (id != movie.Id) {
         return NotFound();
      }
      if (MovieService.IsValid(movie)) {
         _context.Update(movie);
         _context.SaveChangesAsync();

         return RedirectToAction(nameof(Index));
      }
      return View(movie);
   }
}
```

At this stage, don't care about what *NotFound()*, *RedirectToAction()*, or *View()* is. Just know that they are all Views

Tasks

Based on the Get Cats example in the video, describe **what happens** for the 2 following scenarios:

1. **Searching books** in Amazon, and
2. **Displaying a book** in Amazon

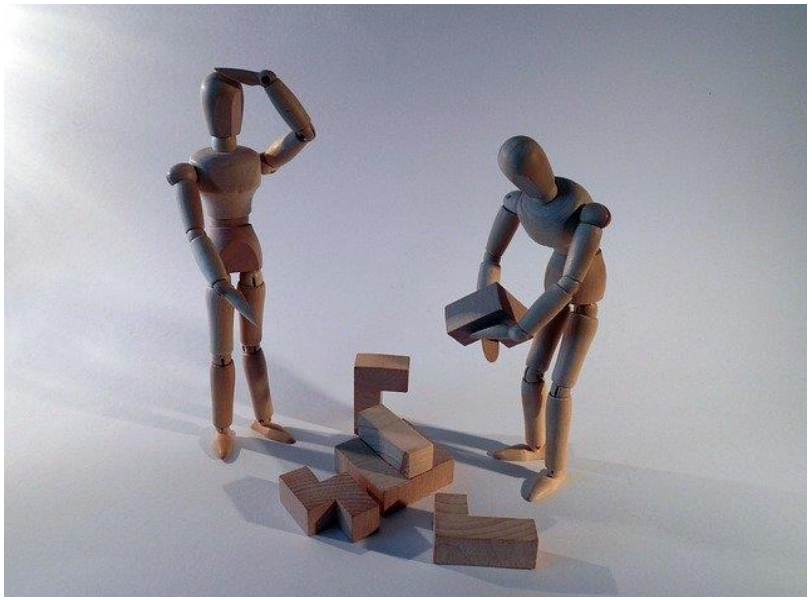


Image by succo from Pixabay

MVC Web Frameworks

In **ASP.NET Core MVC,** Model, View and Controller classes are all in **server side**

Benefits of MVC

1. **Separation of concerns**, so can focus on developing/debugging **different components**
2. **Decoupling models from views**, therefore
   - Controllers can select **different views** for the **same model**
   - Improve **testability** - since testing UI is generally difficult, we can focus on testing models

Task

When working on any MVC project, look at the code and try to answer the following questions:

1. What user actions are provided?
2. Given a user action, what controller is handling it?
3. What model(s) is the controller working with?
4. What data is the model(s) holding?
5. What view(s) is the controller working with? Is there only one or many different views? Based on what condition is a particular view selected?
6. How does the controller send model data to the view?
7. Given the model data, what representation are the views generating?

This slide would be useful in the long run!

Can you guess?

Have **MVC covered all** the **tasks** that servers need to do?

Image by Ranjat M from Pixabay

Readings

- ASP.NET Core In Action, Section 4.1, An Introduction to MVC, by *Andrew Lock*
- Overview of ASP.NET Core MVC https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0
- Tutorial: Create a web app with ASP.NET Core MVC https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-6.0&tabs=visual-studio