MVC Overview and Model Binding	Understanding the MVC pattern in ASP.NET Core Model View Controller is a pattern that separates an app into three parts. **Model:** Application Data + Validation rules + domain Logic (via services) **View :** UI layer (razor.cshtml) that renders HTML **Controller:** Coordination requests, call services, select view/models.
	Roles of Models, Views, and Controllers **Models : C# types that represent data, decorated with Data annotations for validations.** **Views : Razor templates that use Taghelpers/HTML helpers to generate accessible, validated forms.** **Controllers : Classes with actions(methods) invoked by routing, they handle model binding, call service returns views or json.**
	Setting up an MVC project Step 1: Create an ASP.NET core Web App (MVC) dotnet new mvc -n MvcDemo Step 2: Creating a Model(class with properties and Data annotations) Step 3: Create controller (Index action method that returns a view Step 4: Create a view (Razor view based on index.cshtml) Step 5: Run application Step 6: Adding create Action method in the same controller Step 7 : Creating a respective view for the action method & Run application
	Model binding in MVC: simple and complex types: • Automatically maps HTTP request data (form fields, query strings, JSON) to # model properties. • It is primarily used in form submissions, API calls, and URL parameters. Here we are automatically mapping request data to C# objects. Submit the form -> Model binding maps fields to Product Properties. In case any validation fails, error are displayed. **Key features of MVC :** 1. **Attribute & conventional routing** app.MapControllerRoute(name: "default", pattern: "{controller=Home}/{action=Index}/{id?}") • **Maps URLs like /Products/Details/5 → ProductsController.Details(int id)** Attribute routing [Route(“product/[id:int}”)]

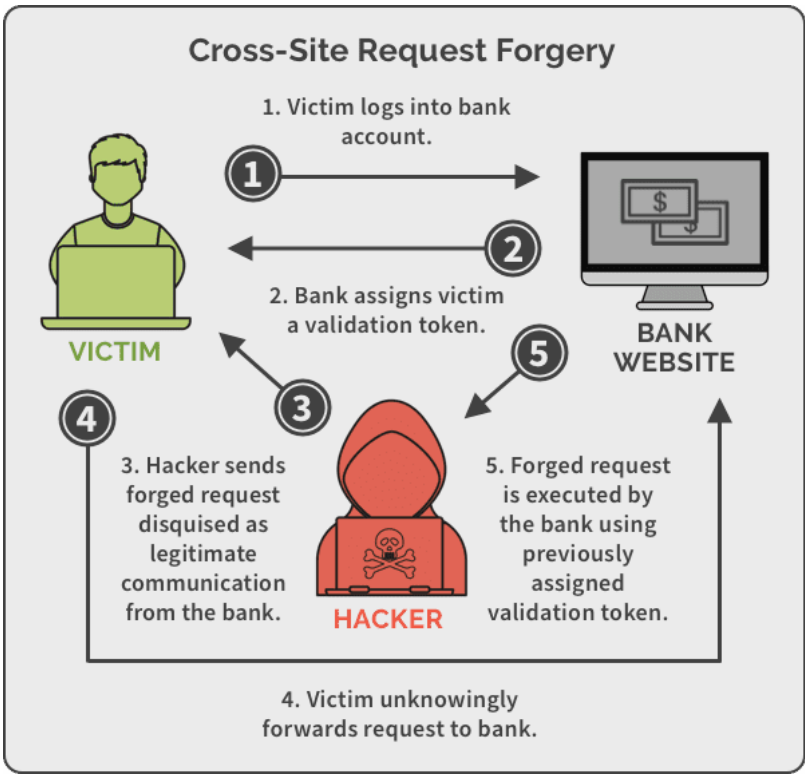
Public IActionResult Details(int id) {..}

- **Here we have more control over URL structure eg /products/5**

2. **Strongly typed views**
   a. Views can be bound to a specific model type(@model product)
   b. Compile time safety

```
@model Product// strongly types model
<h1> @model.Name</h1>  // compile time checking
```

3. **Built in antiforgery & validations**
   a. Anti forgery tokens[validateAntiForgeryToken] that prevent CSRF attacks.
   b. Model Validations(via [Required], [Range]



4. **Tag helpers & HTML helpers for ergonomic markup**
   a. **Use Tag helpers for beer readability and they are aligned with HTML**
   b. **HML helpers are useful or dynamic rendering**

	TagHelpers and HTML Helper Classes	Introduction to TagHelpers in ASP.NET Core
		Common TagHelpers for forms, links, etc.
		Using HTML Helper classes to generate HTML elements
		Custom TagHelpers and HTML Helpers
	Validations Overview and Data Annotations	Importance of data validation in web applications
		Using Data Annotations for server-side validation
		Common validation attributes and scenarios
	Server-Side and Client-Side Validation	Implementing server-side validation in MVC
		Setting up client-side validation with unobtrusive JavaScript
		Synchronizing server and client-side validations
		Build an MVC application incorporating cookies, sessions, and filters.

	RAzor	MVC
Architecture	Page based(page Model + View)	Controller - Based (MVC triad)
File organisation	Each Page has .cshtml + .cshtml.cs	Separate controllers, view and Models
Routing	Based on file structure(Pages folder)	Attribute or conventional routing
Default structure	@Page directive in CSHTML	Controller/Action route patterns
Code Separation	PageModel contains logic for page	Logic is split across controller/Model
Handler Methods	OnGet() and OnPost() ec	Action methods like Index() , Create()
Complexity	Simpler for page-focussed scenarios	Better for complex application
Testability	Less testability(tight coupling)	More testability(better separation)
Recommended use	Content-centric websites	API-driven or complex web apps
Dependency Injection(DI)	Supported in Page model	Supported in Controller
Initial Setup	Less boilerplate	More boilerplate
Model binding	Built into page model	Handled in controller
Lifecycle	Page Handlers(OnGet(), OnPost() etc)	Action filters, model binding etc

	GET method	POST method
Purpose	Retrieving data from the server	Submitting data to the server
Visibility	Data sent in URL (visible in address bar)	Data sent in HTTP request body(hidden)

Security	Less secure	More secure
Data length	Limited(2048 char due to URL Limits)	No strict limits
caching	Can be cached/bookmarked	Not cached by default
Use cases	Search queries, page navigation	Form submissions, file uploads & logins
back/refresh	safe(No data resubmission)	Browser warns about resubmission
Speed	Faster(no request body processing)	Slightly slower(requires body parsing)
Ex	?search=apple(URL parameter)	<form method = “post> hidden in the body

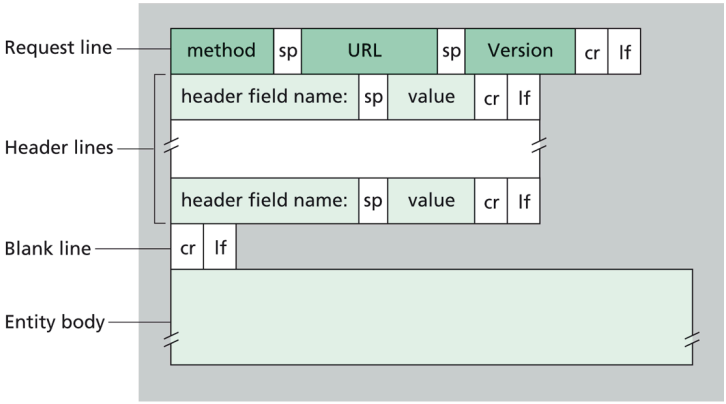
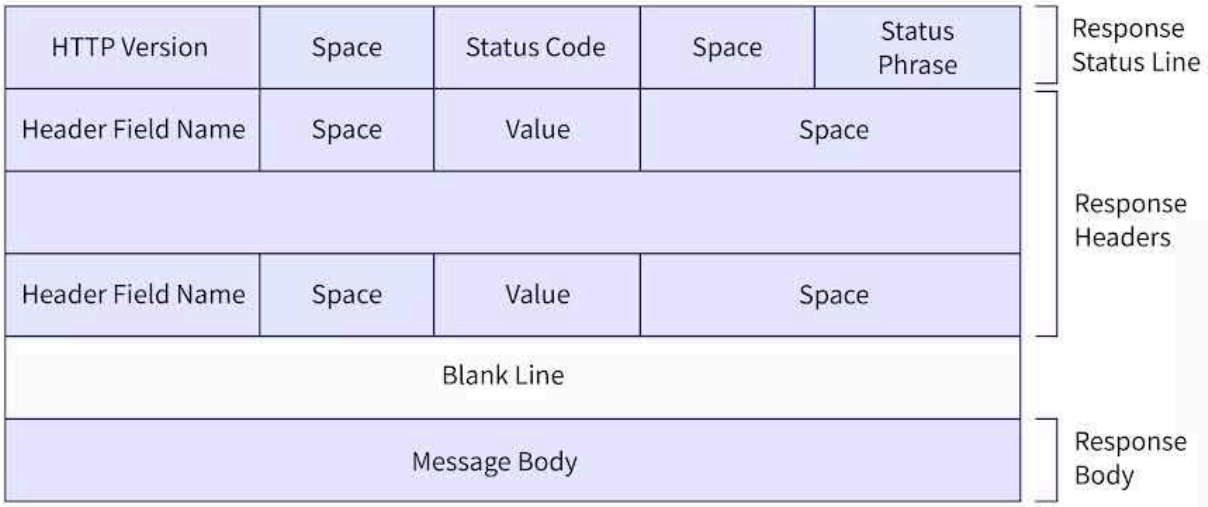


Figure 2.8 ♦ General format of a request message



Best practices while working with MVC application :

1. *Routing : prefer Attribute Routing for APis, Conventional for web apps*
2. *Views : Always use strongly-typed views(@model)*
3. *Security: Always use [validateAniForgeryToken] on POST*
4. *Validation : Combine Data annotations + client side validations*
5. *Tag helpers : Favour over HTML helpers for cleaner syntax*
6. *Separation of concern : keep controllers thin(move logic to services)*

Case Study : Bookstore management system with following capabilities :

- *Razor pages for managing books(CRUD operations)*
- *MVC for handling author management.*
- *Usage of model binding,routing, partial views and complex types.*

Step 1: Create a ASP.NET Core Web APP

Step 2: Adding MVCsupport

- *Defining book model(Models/Book.cs)*
- *Creating Razor page(Pages/Books/AddBook.cshtml)*
- *Page model (Pages.Books/AddBook.cshtml,cs)*

Step 3: Binding Complex ypes & Collection ie binding List<Books>

- *Here we can use [BindPoperty]*

Step 4: Creating reusable partial view

- *Reusing a book card display across multiple pages.*
- *Creating Pages/shared/_BookCard.cshtml and using this in Pages/Books/index.cshtml*

Step 5: Custom Routing in razor pages

- *.Books/Details?id=1 → /book/1*
  - *Adding route in Pages/Books/Details.cshtml*
  - *Generating links with asp-page*

Creating a book catalog using VS code where :

1. ***The main page shows you a book list.***
2. ***Reusable partial view for book cards***
3. ***Add new books using[Bind Property]***

Step 1: Crying a project using CLI and opening it inVS Code

Step 2: Creating a Book model

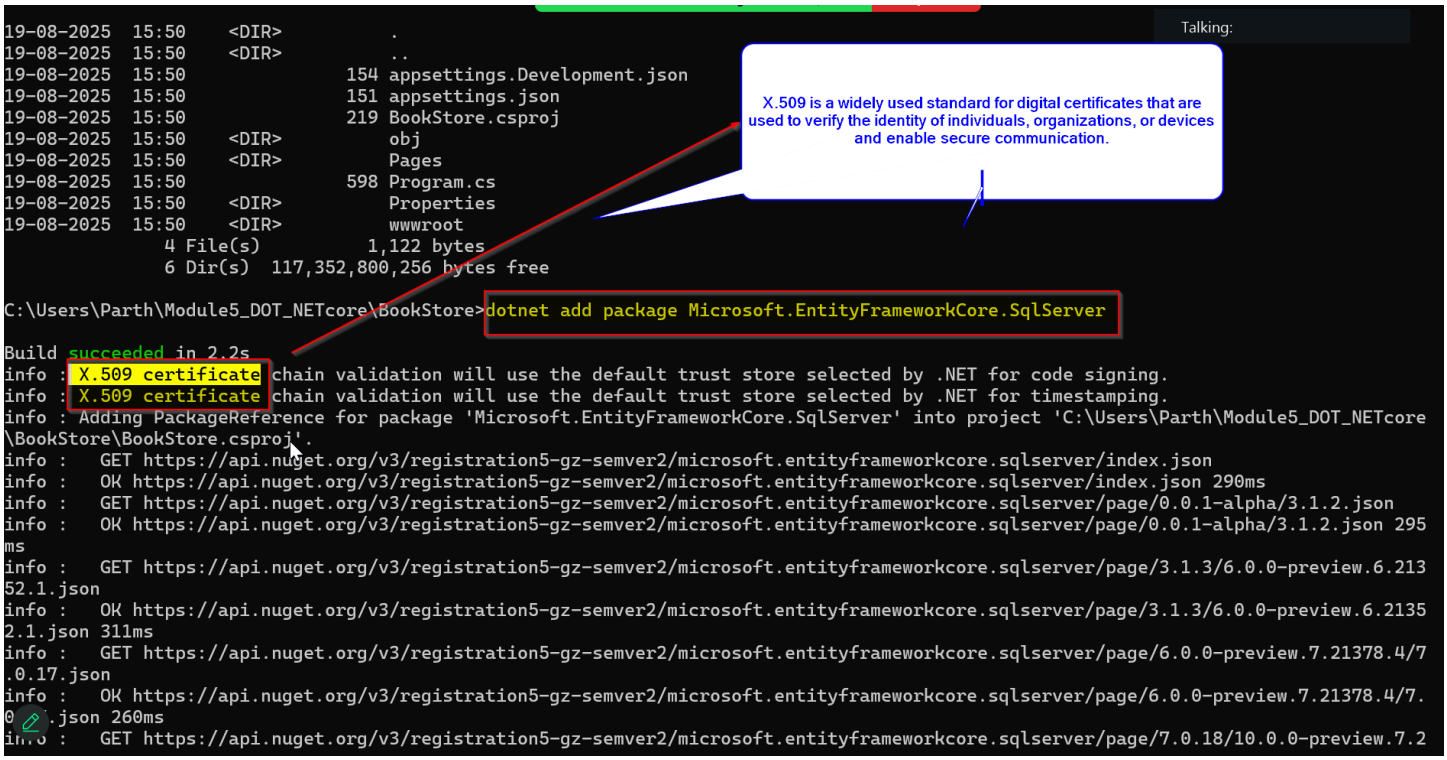
Step 3: Creating a Book Controller with Index() addBooks()

Step 4: Creating a View for index and a partial view for _Bookcard

Step 5: Updating layout for bootstrap

Step 6: Adding route

Step 7: Running the application



Major diff when we are creating .NET app in VStudio and VS code

	Visual studio	***VS code***
	GUI approach	*Builtin CLI*
Solution structure	*.sln is automatically created*	*No.sln*
intellisense	*Full-features, robust and predictive*	*Good, but slightly comprehensive*
Razor editor	*Advance razor editing with preview*	*Basic support(requires extensions)*
Project configuration	*GUI Property pages*	*Manual editing of .csproject file.*
Testing	*Built-in test explorer*	*Required test explorer extension*
Database tools	*SQLServer object explorer, EF tools*	*Limited (CLI based approach)*
Performance	*Heavier (RAM/CPU consumption is high)*	*Lightweight and fast*
Git integration	*Excellent built-in git tools*	*Good through extensions*
Deployment	*Builtin publish profiles and wizard*	*Manual or via extensions*