**CSPROJ-Project file**

Have a project file end with .csproj. It contains information about the target framework (the version of .NET) and package references ,nullable and implicitusing

**Hosting model**

How the application is hosted (in-process or out-of-process).

In-process: The program is hosted inside IIS Express.

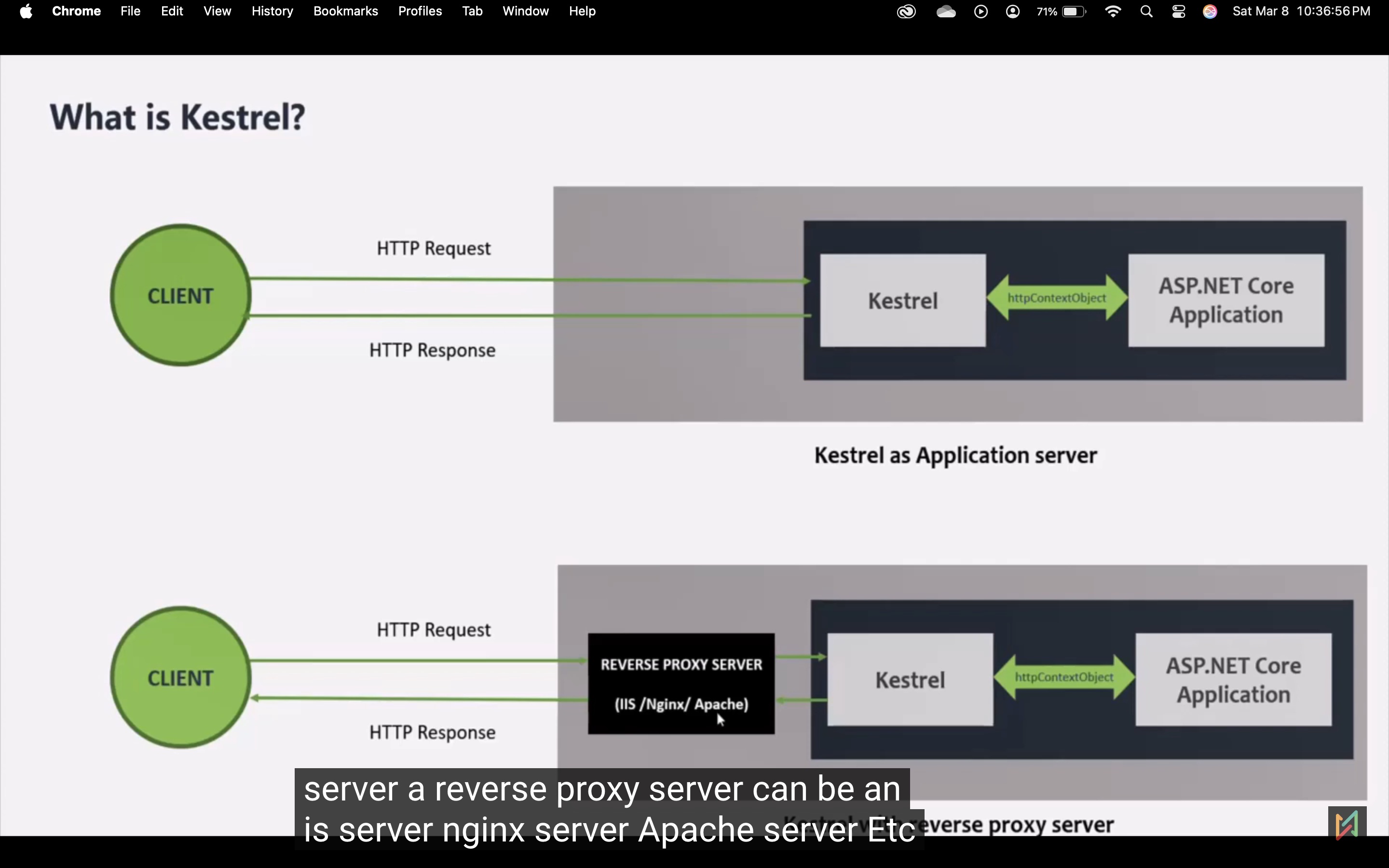
Out-of-process: The program is hosted internal and webserver.

Internal🡪kerstel(crsoo-platform webserver)

External🡪IIS,nginx,apache

Kestrel🡪cross platform webserver for .netcore Apps,iis is only for windows

It can also be a application server, reverse proxy server



In the diagram above, we’re discussing the development process. During development, we receive requests from the webpage. The Kestrel receives these requests, creates an HTTP context object, and sends it to the .NET Core application. After the application processes the request, the Kestrel receives the HTTP context object and sends a response back to the browser.

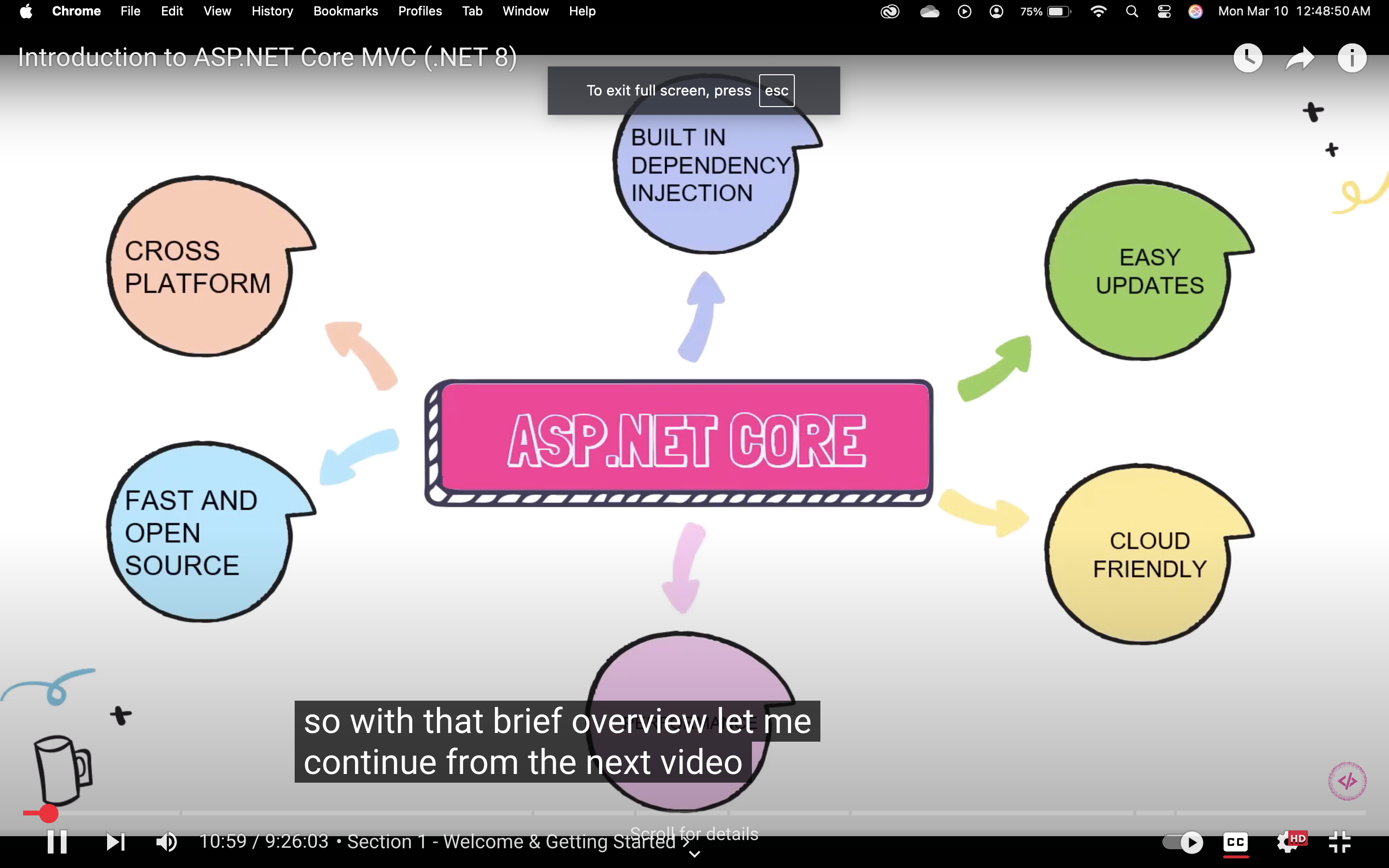
In a production environment, kestrel alone is not sufficient to handle browser requests. To address this, we also use a proxy server.

Once the “.NET Core Web Application” run (HTTP, hotload) button is clicked, a console application will be started. This application will then start the Kestrel server, where the application will be hosted.

**client server architecture**

<https://www.youtube.com/watch?v=KSPlGDxOilM>

<https://www.youtube.com/watch?v=a3LHYguyHkw&list=PL1BztTYDF-QPgfvPouABKLwfTKxB6z7gk&index=14>



**Lauchsettings.json**

Which profile should we use for the application? The profiles are http, https, and iisexpress. The URL details and environment details will be available.

**Wwwrott**

Static files, such as CSS and JavaScript, will be placed here.

**Appsettings.json**

Connection string, secret key

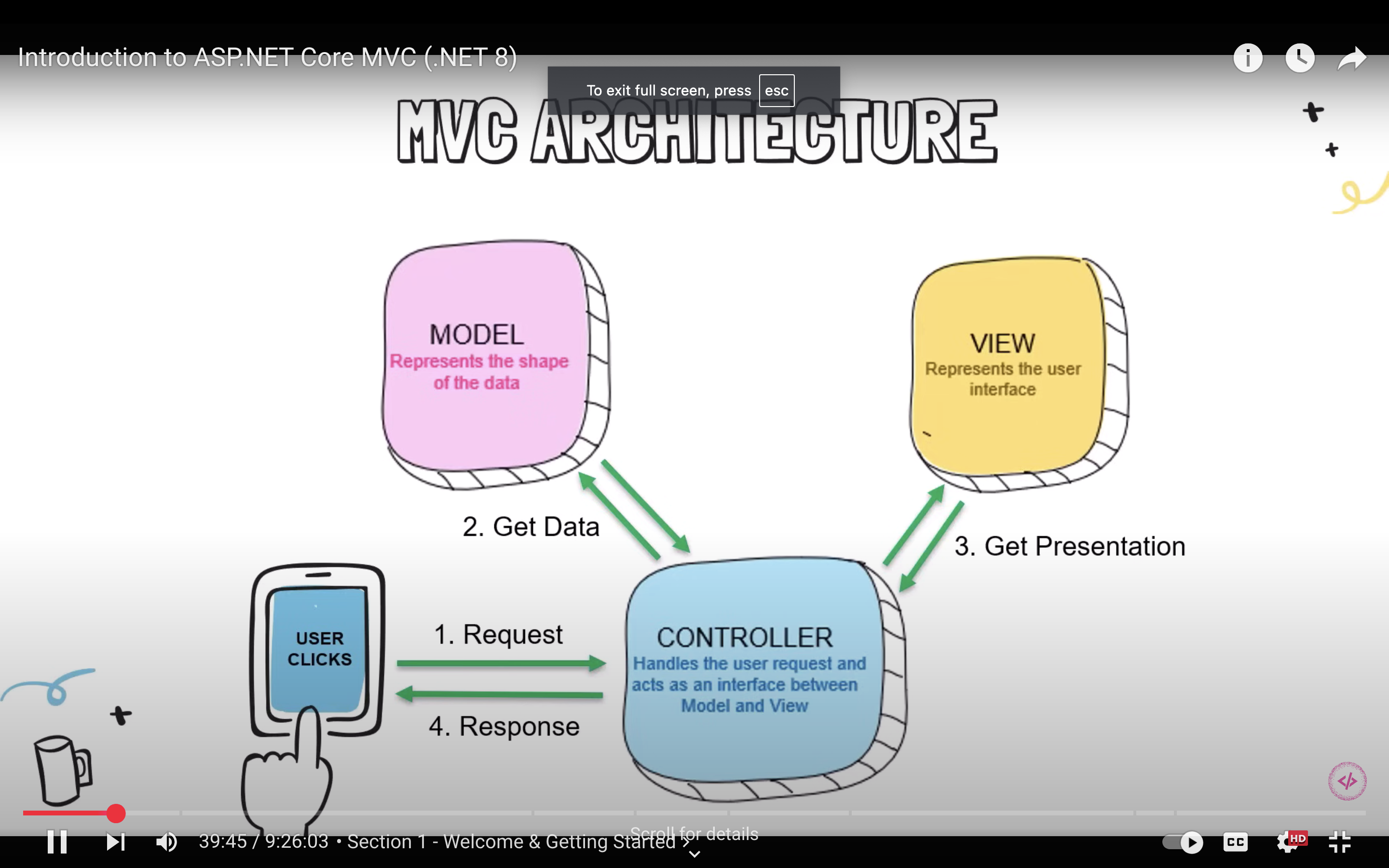
**Program.cs**

Configure a service, create a middleware

Model🡪Represents the shape of the data

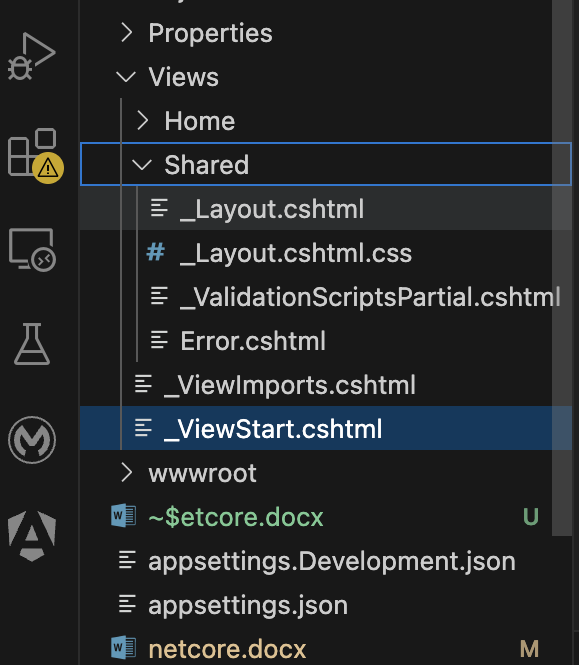
View🡪represent the user Interface

Controller🡪Handles the request and act as an interface between between model and view



Once the user clicks on the browser, the request goes to the controller. The controller retrieves data from the model and passes it back to the controller. The controller calls the view, which creates an HTML page representation based on the data. The view then passes the representation back to the controller. Finally, the controller sends back the response.

**Views**



\_Layout.cshtml: The master page contains the header, footer, and uses @RendorBody() to render the body content.

\_validationscriptpartial: This partial view adds client-side validation.

Error.cstml: This view handles error messages.

\_ViewStart.cshtml: This view specifies which page is using the layout.

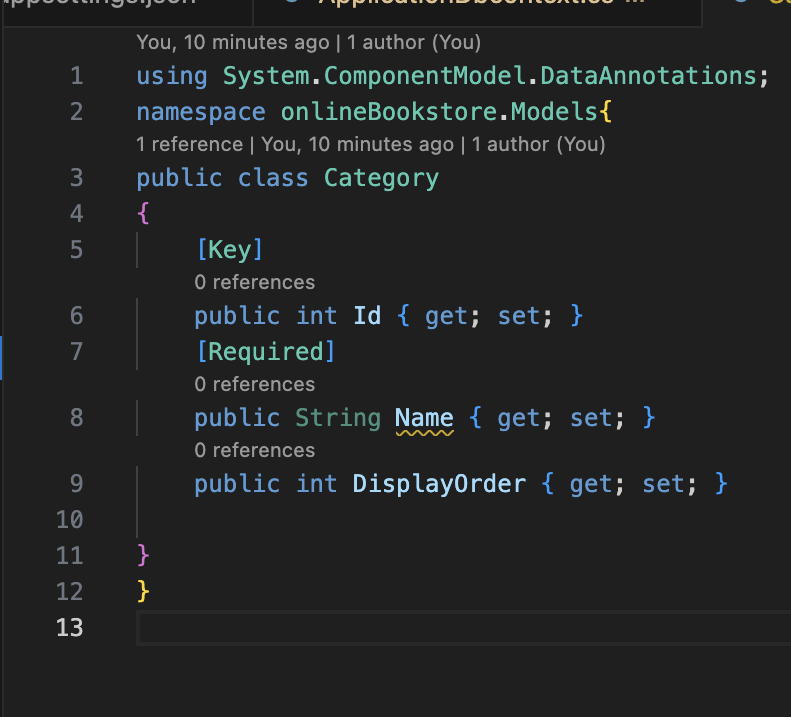
\_ViewImport.cshtml:Instead of importing each time in a different file, it uses a global import, eliminating the need to import in each file. **only available in views**

**Controller**

Return IACTIONRESULT

**Add a table suing Entity Framework**

1.Create model class



2.add a new folder Data , create ApplicationDbContext.cs



3.Add connection to the db in Program.cs

builder.Services.AddDbContext<ApplicationDbContext>(options=>options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

4. dotnet ef migrations add InitialCreate

dotnet ef database update

automatically the database will be created in sqlserver

**1. View**

A **View** is a complete HTML template that represents the UI for a specific action in a controller. It is used to display data and interact with users.

| **Feature** | **View** | **Partial View** |
| --- | --- | --- |
| **Purpose** | Renders a full HTML page | Renders a reusable UI section |
| **Layout Support** | Uses a layout page | Does not use layout by default |
| **Reusability** | Specific to a page | Can be reused across multiple views |
| **Rendering Methods** | return View() | @await Html.PartialAsync() or @await Html.RenderPartialAsync() |
| **Folder Location** | Stored in Views/ControllerName | Stored in Views/Shared or Views/ControllerName |

Clientside validation

If we want to show the validations on client side like html,if we don’t want go the controller and check validations.

Then we use client side validations

@section scripts{

<partial name="\_ValidationScriptsPartial"/>

}

**Pass id in query string**

<a asp-controller="category" asp-action="delete" asp-route-Id="@item.Id"><

**📌 Quick Comparison Table**

| **Feature** | **ViewData** | **ViewBag** | **TempData** |
| --- | --- | --- | --- |
| **Type** | Dictionary (ViewDataDictionary) | Dynamic (dynamic) | Dictionary (TempDataDictionary) |
| **Scope** | **Only for the current request** | **Only for the current request** | **Persists across redirects** |
| **Data Persistence** | ❌ **No** (Lost after request) | ❌ **No** (Lost after request) | ✅ **Yes** (Available for the next request) |
| **Access Type** | ViewData["key"] | ViewBag.key | TempData["key"] |
| **Use Case** | Passing data from controller to view | Similar to ViewData, but simpler syntax | Passes data across controller actions (Redirects) |
| **Performance** | Slightly faster | Slightly slower | Uses Session storage |
| **Underlying Type** | Dictionary (ViewDataDictionary) | Wrapper around ViewData | Uses Session internally |