MVC Pattern

There are three approaches to building modern web UIs with ASP.NET Core

We can build apps that:

- Render User Interface (UI) from the server. The web app is dynamically rendered on the server and arrives at the client ready for display.
    - The UI's HTML and CSS are dynamically generated on the server in response to a browser request.

Why Render UI from the server?

- The advantage of using this approach is that the client requirements are minimal because the server takes care of the logic and page generation.
- This is great for building for low end devices, users with low bandwidth connections and multiple browser versions.

There's a downside of course:

- The downside is that it is expensive as the server handles all the work and user interaction is time consuming because every update has to be generated and rendered by the server.

- This approach is commonly used for static sites.
- Render the User Interface(UI) from the client's browser. The UI will be dynamically rendered in the client's browser by manipulation the DOM (Document Object Model)

accordingly, which will allow us to write code that can change the document structure, style, and content.

Why Render UI client side?

- The advantage of this approach is that the app is highly interactive as the UI updates are handled locally and it is cheaper as the cost of rendering the UI is offloaded to the client.
- It also supports building progressive web apps as the offline mode will still provide interactivity.

There has to be a balance

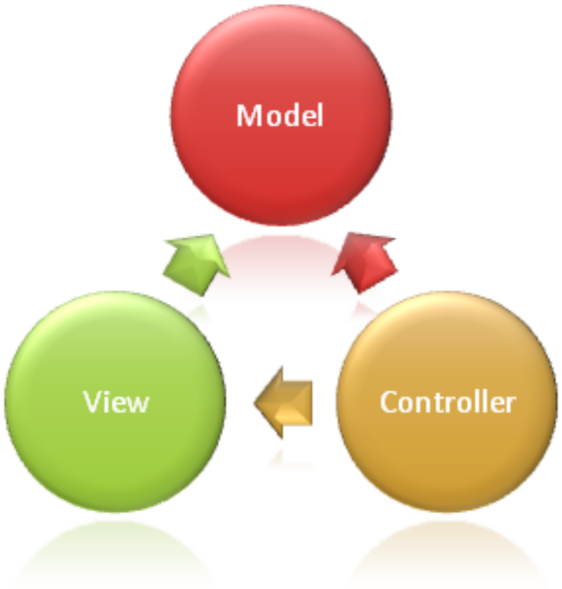
- The downside is that it will exclude clients that have low end devices or low bandwidth and when it comes to load time of the logic users may experience latency as the code needs to be downloaded onto the client.

- This approach is commonly used for interactive dashboards or collaborative apps.
- Hybrid approach that uses both the client and server user interface rendering

When using the Server Rendered UI we can choose between Razor Pages or Model View Controller (MVC).

MVC

The Model View Controller pattern separates the app into Models, Views and Controllers.



In a Nutshell:

Using the MVC pattern for websites, requests are routed to a Controller that is responsible for working with the Model to perform actions and/or retrieve data. The Controller chooses the View to display and provides it with the Model. The View renders the final page, based on the data in the Model. It supports decoupling of low level details and for this reason, It is great for building scalable large apps.

The game players

- Controllers are responsible for working with the Model to perform user actions. The controller chooses the view to display and provides it with data required.

💡 Controllers shouldn't be overly complicated by too many responsibilities.
To keep controller logic from becoming overly complex, push business logic out of the controller and into the domain model.

- The Model represents the state of the application and any business logic or operations that should be performed by it. It is used to retrieve and store model state in a database such as SQL Server database. It is also used for business logic separation from the data in the application.

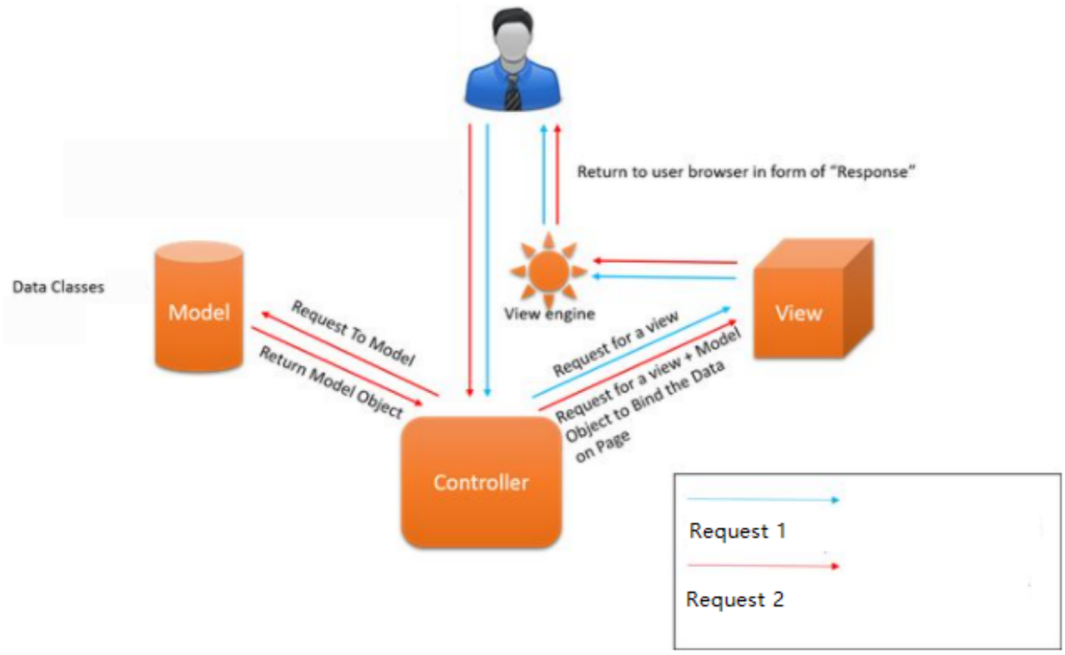
- The View handles data presentation and user interaction. A view is an HTML template with embedded Razor markup. Razor markup is code that interacts with HTML markup to produce a webpage that's sent to the client.
  In ASP.NET Core MVC, views are .cshtml files that use the C# programming language in Razor markup. Usually, view files are grouped into folders named for each of the controllers.

Why MVC?

- It manages application complexity by dividing an application into the model, view and controller.
- It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application.
- It provides better support for test-driven development.

- It is suitable for large scale developer team and web applications.
- It provides high degree of control to the developer over the application behavior.

Different MVC User Interactions (Requests)



- Request 1 (Blue) returns a view only without data from the Model. When the user issues a request, it is handled by the MVC Handler and redirected to the Controller which will call an Action Method i.e. [***HttpGet***]. If the method returns an instance of a **ViewResult** then it will find the suitable view from the View Directory from the web site solution and then the V**iew Engine (Razor)** will render the view in HTML for the user's browser.
- Request 2(Red) is when a user requests a **View** + data from the **Model.** When the user makes a request, It is handled by the MVC Handler and redirected to the Controller which will directly call an Action Method i.e. [***HttpGet***]. If the method returns an instance of **ViewResult** with a **Model Object,** a request to the Model Class for an object that we need to use on the view. A **ViewResult(ModelObject)** will be returned and handled by the **Model** property of the View Class.