# Introduction to DOM

* Stands for Document Object Model
* When the browser finishes creating the website using HTML, CSS, JS, etc. It will create a DOM of the page.
* This is how we can select one or more elements in HTML (Remember about CSS Selectors).
* It treats HTML elements as objects.
  + You can change the properties of elements.
  + You can have methods to change/manipulate elements.
  + The events for elements.
* So essentially, we will use the DOM to get, change, add, or delete elements in our webpage to dynamically change it depending on what the user is doing.

## Dom Events

* Events (like C# events) it will give a response whenever a certain event that happens such as click a button, scrolling your wheel, or pressing a key on your keyboard.

# Introduction to MVC

* Stands for Model-View-Controller
* It is a design pattern we used to separate our application into three main groups.

## Model

* The model process data / user input.
* It represents the current state of your application.
* It is composed of data structures and any other logic that is involved with data processing.
  + It will also be comprised of your BL, DL.

## View

* It is what the end-user sees and interacts with.
* Essentially the webpage you see when visiting a website.
* So, it is composed of your HTML, CSS, JS, or assets.

## Controller

* The controller handles the client requests.
* It will call upon appropriate business logic (from your model) to process what the client wants from the server.

## ASP.Net MVC

* It is a web application framework in .NET 5 that implements the MVC design pattern to create a website.
* Since it uses the MVC design pattern, your web application is separated into model, a view, a controller.

# Model

* It will hold the data structure that we will be using for our view.

## View Model

* View models are as the name states, data structures used in views.
* There are some data you would not want to necessarily present to the user. Or sometimes you would want to aggregate the data that was separated out in your business logic models because they are processed separately.
* Note that to take advantage of tying your views to models, you would want to repackage some of your data into a data model since by default, views do not recognize composition relationships between objects.
  + **They can only scaffold value type properties** and not the complex ones (classes / reference types).

## Data Annotations

* Data annotations are used to add validation to your view models.
* With data annotations you will be able to add client-side validations based on business rules you have set in your models (quantities cannot be negative or first name has to be filled).
* In your views we will be using tag helpers.

## Model Binding

* When you tightly couple a view to a model.
* When you use a model as a parameter in your controller actions. You model bind to a particular model client side when you use the validations set on your models via the data annotations to impose client side validation. Server side, you check the validity of your model being passed in the request body via ModelState.IsValid property.

# View

* The view is the template for the actual html page we will be serving to the end-user.

## Types of Views

* Strongly Typed
  + Views are tied to a model. This allows the advantage of strong type checking. Every variable and constant has an explicitly defined type.
  + Specify model type with @model at the top of the view.
* Weakly Typed
  + Views are not tied to a model.
* Dynamically Typed
  + You pass models to it, but it is not tightly coupled to that specific model.
  + Use @model dynamic.
  + You can still access the properties via @model, but the Model you are passing must have properties referred to similarly.

## Partial Views

* Reduce code duplication by manage reusable parts of the view.

## View components

* Like partial views. The view content requires code to run on the server to render the webpage.

# Razor

* Markup syntax used to embed C# logic in webpages.

## Razor view engine

* What compiles your razor files to be able to dynamically generate web content on the server.

# Helpers

* Are used to tie controller and model logic to the view.

## Tag Helpers

* Look like HTML tags. Enable server side code to participate in creating and rendering HTML elements in razor files. Bind specific elements and its attributes. Server side rendering while still preserving the HTML editing experience. Markups better in performance in comparison to html helpers.

## HTML Helpers

* Similar to Tag helpers. Invoked as methods that are mixed with HTML inside your Razor views.
* Uses C# under hood.

## View Discovery

* Determines which view file is used based on the view name. Default to return view with the same name as the action method. Searches view in the view folder allocated to the controller then searches for the view in the shared folder.

# Controller

* Responsible for the initial processing of the request and instantiation of the model.

## Action

* Public method of the controller. Parameters on action are bound to request data and are validated using model binding.
* Return ActionResults.

## Action Verbs

* Used when you want to control the selection of an action method based on a HTTP request method.