1. MVC

- Model: class the represents data of the app, also contains validation logic and methods related to the functionality of the

- View: component that displays the UI of the app, usually it is related to a model in order to display data to the user

- Controller: it handles the browser requests, it is like an intermediary between the model(data) and the view(displaying of the data). It responds to user interaction by handling input and reacting accordingly

2. Naming conventions: when it come to names, all models, controllers and views have Pascal Case(e.g MyClassName). Models should have short and suggestive names since they represent the data, Controllers related to models have the Model’s full names suffixed by “Controller”(e.g MyClassNameController) and controller actions can have any name, but the rules state that they should also be Pascal Cased(e.g IsLoggedIn, RegisterUser). When it come to Views, the standard 2 folders are Home containing the Index.cshtml and related views and Shared that usually contains \_Layout.cshtml which represent the common layout for the whole app. Other folders can have any names respecting the same conventions as everything else and they contain related views(Account folder would have Login and Register views).

3. 2 ways of passing data from controller to view:

- By using ViewBag, which is a dynamic dictionary in which we can put data which can be accessed in the view using ViewBag.Variable

- By using ViewData similar to ViewBag, we access it by ViewData[“variable”]

- By using a Model that we pass when we return View in the controller. For this we have to specify in the View what model are we using and the we can access data as ModelName.variable

4. By default, routes are of the type “controller”/”action”/”id” so in the Controller we can map actions to different routes and for this we have multiple functions(e.g RedirectToAction(action) or RedirectToAction(action, controller)). Also in the View we can use @Html.ActionLink(name, action, controller)

5. For this we can use Attributes like [HttpPost] or [HttpGet] to specifiy that a specific action demands a certain type of request. Moreover we can use [Authorize] to restrict access to actions by demanding some kind of authorization.

6. To make an action available only through a form you can use the [ChildOnlyAction] attribute which will ensure that the action is not publicly routable via URL and that it can only be accessed by the related form

7. Data validation is defined in the model as DataAnnotation attributes(e.g [Required], [Range(x,y)] over the properties. After defining those, in the view we can link the fields of a form to each property in order to create, for example, a new object of type Model. Then, in the controller we can check the integrity of the data with Model.isValid and then do the future action according to the result of this.