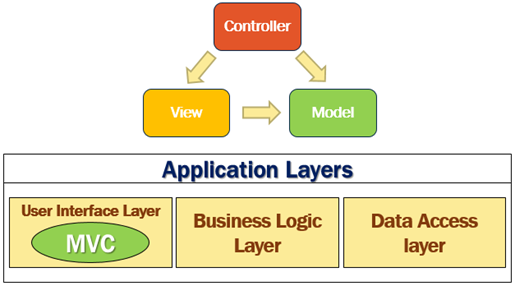
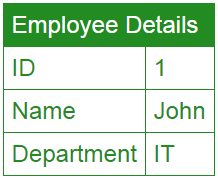
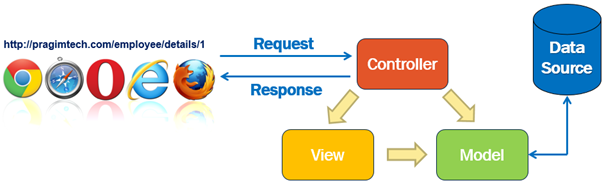
**ASP.NET Core MVC tutorial**

**What is MVC**   
  
   
  
MVC consists of three fundamental parts - **Model**, **View**and **Controller**. It's an architectural design pattern for implementing User Interface Layer of an application. A typical real world application usually has the following layers.

* User Interface Layer
* Business Logic Layer or Domain Layer
* Data Access layer

MVC design pattern is usually used for implementing the User Interface Layer of the application.   
  
**How MVC Works**   
  
Let us understand how the MVC design pattern works with an example. Let's say we want to retrieve a specific employee details (i.e an employee whose ID is 1) and display those details on a web page in an HTML table as shown below.   
  
   
  
So from the web browser we issue a request and the URL may look something like the following  
**http://Sample.com/employee/details/1**   
  


1. When this request arrives at our server, it is the Controller in the MVC design pattern that receives the request and handles it.
2. The controller creates the model. The model has the classes that describe the data.
3. In addition to the data itself, Model also contains the logic to retrieve data from the underlying data source such as a database.
4. In addition to creating the Model, the controller also selects a View and passes the Model object to that View.
5. The view is only responsible for presenting the model data.
6. It is the view, that generates the required HTML to present the model data i.e the employee data provided to it by the Controller.
7. This HTML is then sent over the network to the user who made the request.

**Model**   
  
So model in this case consists of the Employee class + the EmployeeRepository class that manages the employee data as shown below.

public class Employee  
{  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public string Department { get; set; }  
}  
  
public interface IEmployeeRepository  
{  
    Employee GetEmployee(int id);  
    void Save(Employee employee);  
}  
  
public class EmployeeRepository : IEmployeeRepository  
{  
    public Employee GetEmployee(int id)  
    {  
        // Logic to retrieve employee details  
    }  
  
    public void Save(Employee employee)  
    {  
        // Logic to save employee details  
    }  
}

We use the Employee class to hold the employee data and It is the EmployeeRepository class that retrieves and saves data in the underlying data source.    
  
So to generalise this, **a Model in MVC contains a set of classes that represent data and the logic to manage that data.** The class that represent the data is the Employee class and the class that manages the data is the EmployeeRepository class.  
  
If you are wondering why are we using the interface IEmployeeRepository. Can't we use just the EmployeeRepository class without the interface.  
  
Well, we can, but using the interface abstraction allows us to use dependency injection which in turn allows us to create systems that are loosely coupled  and easily testable.

**View**  
  
A View in MVC should only contain the logic to display the Model data provided to it by the Controller. You can think of a view as an HTML Template. Let's say in our example, we want to display Employee data in an HTML table.   
  
The view in this case will be provided with the Employee object. The Employee object is the model that carries the employee data to the view. The only responsibility of the view is to present the employee data in an HTML table. Here is the code in the view.

@model EmployeeManagement.Employee  
  
<html>  
<head>  
    <title>Employee Details</title>  
</head>  
<body>  
    <table>  
        <tr>  
            <td>ID</td>  
            <td>@Model.Id</td>  
        </tr>  
        <tr>  
            <td>Name</td>  
            <td>@Model.Name</td>  
        </tr>  
        <tr>  
            <td>Department</td>  
            <td>@Model.Department </td>  
        </tr>  
    </table>  
</body>  
</html>

In MVC, a View in only responsible for presenting the model data. There should be no complex logic in a view. To maintain a clear separation of concerns, the logic in a view must be very minimal and that too it must only be there for presenting data. If you get to a point where the presentation logic is getting too complicated, consider using a **ViewModel**or **View Component**. **View Components**are new in this version of MVC. We will discuss View Components in detail later.   
  
**Controller**  
  
When a request from the browser arrives at our application, it is the controller in the MVC design pattern, that handles the incoming http request and responds to the user action.    
  
In this case the user has issued a request to the URL (/employee/details/1), so this request is mapped to the Details action method in the EmployeeController, passing it the EMPLOYEE ID which in this case is 1. This mapping is done by the **Routing rules**defined in our application. We will discuss Routing in ASP.NET Core MVC in detail in our upcoming sessions.

public class EmployeeController : Controller  
{  
    private IEmployeeRepository \_employeeRepository;  
  
    public EmployeeController(IEmployeeRepository employeeRepository)  
    {  
        \_employeeRepository = employeeRepository;  
    }  
  
    public IActionResult Details(int id)  
    {  
        Employee model = \_employeeRepository.GetEmployee(id);  
        return View(model);  
    }  
}

As you can see, from the code in the **Details**action method, the controller builds the model, in this case the Model is the **Employee**object. To retrieve the Employee data from the underlying data source, the controller is making use of the **EmployeeRepository**class.   
  
Once the controller has constructed the Employee model object with the required data, it then passes that Employee model object to the view. The view then generates the required HTML to present the Employee data provided to it by the Controller. This HTML is then sent over the network to the user who made the request.  
  
  
**Summary**

* MVC is an architectural design pattern for implementing User Interface Layer of an application
* **Model :** Set of classes that represent data + the logic to manage that data. Example - Emplopyee class that represents the Employee data + EmployeeRepository class that saves and retrieves employee data from the underlying data source such as a Database.
* **View :** Contains the display logic to present the Model data provided to it by the Controller
* **Controller :** Handles the http request, work with the model, and selects a view to render that model.

As you can see, in the MVC design pattern we have a clear separation of concerns i. each component has a very specific task to do. we will discuss setting up the MVC middleware in our asp.net core application.