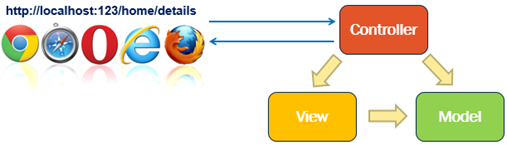
### Controller in ASP.NET Core MVC

we will discuss, **what is Controller and it's role in ASP.NET Core MVC**.  
  
  
**Controller in MVC**  
  


* Controller in MVC is a class and it inherits from Microsoft.AspNetCore.Mvc.Controller
* The controller class name is suffixed with word **"Controller"**. For example HomeController, EmployeeController.
* 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.
* Controller class contains a set of public methods. These public methods in the Controller class are called **action methods**. It is these controller action methods that handle and process the incoming http request.
* Let's say the user has typed the following URL in the browser address bar and hit the ENTER keyhttp://localhost:12345/home/details
* The URL "/home/details" is mapped to the "Details" public action method in the HomeController. This mapping is done by the **routing rules**defined in our application.
* We will discuss Routing in ASP.NET Core MVC in detail later.
* The request arrives at a controller action method. As part of processing that request, the controller creates the **Model**.
* To retrieve model data, the controller depends on a service.
* For example, in our case to retrieve Employee data, HomeController depends on IEmployeeReporsitory service.
* IEmployeeReporsitory service is injected into the HomeController using the constructor. This is called Dependency Injection.

Notice, we are assigning the injected dependency to a read-only field. This is a good practice as it prevents accidentally assigning another value to it inside a method.

* Once the controller has the required model data, it can simply return that model data if we are building a RESTful service or an API.

**Controller returns JSON data**  
  
The following example returns JSON data. Notice, the return type of the Details() method is set to JsonResult as we are explicitly returning JSON data. In this case, Details() method always returns JSON data. It does not respect content negotiation and ignores the **Accept Header**.

public class HomeController : Controller  
{  
    private readonly IEmployeeRepository \_employeeRepository;  
  
    public HomeController(IEmployeeRepository employeeRepository)  
    {  
        \_employeeRepository = employeeRepository;  
    }  
  
    public JsonResult Details()  
    {  
        Employee model = \_employeeRepository.GetEmployee(1);  
        return Json(model);  
    }  
}

**Controller returns ObjectResult**  
  
The following example respects content negotiation. It looks at the **Request Accept Header**and if it is set to **application/xml**, then XML data is returned. If the Accept header is set to **application/json**, then JSON data is returned.

public class HomeController : Controller  
{  
    private IEmployeeRepository \_employeeRepository;  
  
    public HomeController(IEmployeeRepository employeeRepository)  
    {  
        \_employeeRepository = employeeRepository;  
    }  
  
    public ObjectResult Details()  
    {  
        Employee model = \_employeeRepository.GetEmployee(1);  
        return new ObjectResult(model);  
    }  
}

**Please note :**To be able to return data in XML format, we have to add Xml Serializer Formatter by calling AddXmlSerializerFormatters() method in ConfigureServices() method in Startup.cs file.

public void ConfigureServices(IServiceCollection services)  
{  
    services.AddMvc().AddXmlSerializerFormatters();  
}

**Controller returns View**  
  
The following example returns a **View**. Notice we have set ViewResult as the return type for the Details method as we are returning a view.

public class HomeController : Controller  
{  
    private IEmployeeRepository \_employeeRepository;  
  
    public HomeController(IEmployeeRepository employeeRepository)  
    {  
        \_employeeRepository = employeeRepository;  
    }  
  
    public ViewResult Details()  
    {  
        Employee model = \_employeeRepository.GetEmployee(1);  
        return View(model);  
    }  
}

At this point if we run the application and navigate to http://localhost:49119/home/details, we get the following error. This  is because we do not have the required View file created yet.

InvalidOperationException: The view 'Details' was not found. The following locations were searched: /Views/Home/Details.cshtml /Views/Shared/Details.cshtml /Pages/Shared/Details.cshtml

**Summary**

* 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.
* The controller builds the model AND
* Returns the Model data to the caller if we are building an API  OR
* Select a View and pass the model data to the view
* The View then generates the required HTML to present the data