Phase 5

PREPARED BY BRIJESH KAMANI

Brijesh Kamani
FULL STACK DEVELOPER TRAINEE | RKIT

Table of Contents

25. Introduction to WEB Dev.

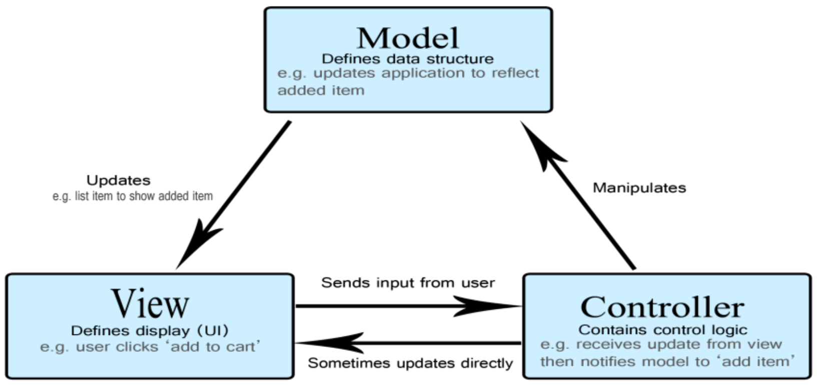
ASP.NET:

- *It is a web framework designed and developed by Microsoft. It is used to develop websites, web applications and web services. It provides fantastic integration of HTML, CSS and JavaScript. It was first released in January 2002. It is built on the Common Language Runtime (CLR) and allows programmers to write code using any supported .NET language.*
- *ASP.NET is a part of the Microsoft .NET Framework.*

25.1. MVC

- ***MVC** (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes the separation between the software's business logic and display. This "separation of concerns" provides for a better division of labour and improved maintenance. Some other design patterns are based on MVC, such as MVVM (Model-View-ViewModel), MVP (Model-View-Presenter), and MVW (Model-View-Whatever).*

- The three parts of the MVC software design pattern can be described as follows:
  1. ***Model**: Manages data and business logic.*
  2. ***View**: Handles layout and display.*
  3. ***Controller**: Routes commands to the model and view parts.*


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

The Model

- *The model defines what data the app should contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).*

- *Going back to our shopping list app, the model would specify what data the list items should contain — item, price, etc. — and what list items are already present.*

The View

- *The view defines how the app's data should be displayed.*

- *In our shopping list app, the view would define how the list is presented to the user, and receive the data to display from the model.*

The Controller

- *The controller contains logic that updates the model and/or views in response to input from the users of the app.*

ASP.NET Core MVC:

- *ASP.NET Core MVC is a rich framework for building web apps and APIs using the Model-View-Controller design pattern.*

- *The ASP.NET Core MVC framework is a lightweight, open-source, highly testable presentation framework optimized for use with ASP.NET Core.*
- *ASP.NET Core MVC provides a patterns-based way to build dynamic websites that enables a clean separation of concerns. It gives you full control over mark-up, supports TDD-friendly development and uses the latest web standards.*

25.2. Rest Web API

API:

- *An API is a set of definitions and protocols for building and integrating application software. It's sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).*
- *You can think of an API as a mediator between the users or clients and the resources or web services they want to get. It's also a way for an organization to share resources and information while maintaining security, control, and authentication—determining who gets access to what.*

RESTful API:

- *REST stands for representational state transfer and was created by computer scientist Roy Fielding.*
- *REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.*
- *When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.*
- *A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.*

REST API With ASP.NET:

- *ASP.NET makes it easy to build services that reach a broad range of clients, including browsers and mobile devices.*

- *With ASP.NET you use the same framework and patterns to build both web pages and services, side-by-side in the same project.*
- *ASP.NET was designed for modern web experiences. Endpoints automatically serialize your classes to properly formatted JSON out of the box. No special configuration is required. Of course, serialization can be customized for endpoints that have unique requirements.*
- *Secure API endpoints with built-in support for industry-standard JSON Web Tokens (JWT). Policy-based authorization gives you the flexibility to define powerful access control rules—all in code.*
- *ASP.NET lets you define routes and verbs in line with your code, using attributes. Data from the request path, query-string, and request body are automatically bound to method parameters.*

26. Starting with Project

26.1. Create New Web API Project

- *In Visual Studio > From the* ***File*** *menu, select* **New** > **Project**.
- *Enter Web API in the search box.*
- *Select the* **ASP.NET Core Web API** *template and select* **Next**.
- *In the* **Configure your new project dialogue**, *name the project TodoApi and select* **Next**.
- *In the* **Additional information** *dialogue:*
- *Confirm the* **Framework** *is* **.NET** ***6.0*** ***(*****Long-term support*****)***.
- *Confirm the checkbox for* **Use controllers(uncheck to use minimal APIs*****)*** *is checked.*
- *Select* **Create**.

26.2. Create Controller, Model

Adding a Model

- *If Solution Explorer is not already visible, click the* **View** *menu and select* **Solution Explorer**. *In Solution Explorer, right-click the Models folder. From the context menu, select* **Add** *then select* **Class**.

- *Name the class and write code like this,*

```
namespace ProductsApp.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

Adding a Controller

- *In* **Solution Explorer***, right-click the Controllers folder. Select* **Add** *and then select* **Controller***.*
- *In the* ***Add*** **Scaffold** *dialogue, select* **Web API Controller - Empty***. Click* **Add***.*
- *In the* **Add Controller** *dialogue, name the controller "ProductsController". Click* **Add***.*
- *The scaffolding creates a file named ProductsController.cs in the Controllers folder.*
- *If this file is not open already, double-click the file to open it. Replace the code in this file with the following:*
- *To keep the example simple, products are stored in a fixed array inside the controller class. Of course, in a real application, you would query a database or use some other external data source.*

26.3. Parameters (From URI, From Body)

- *Parameter binding is a type for catching values from the URI and the message body by the Web API. These depend on the type of the parameter.*

- *There are various rules for binding the parameters:*

- ***"Simple" type:*** *If the parameter is the simple type then it is a string convertible parameter that includes the preemptive data types such as Int, Double, Bool and so on with the Date-Time, Decimal, string and so on. By default, these are read from the URI.*
- ***"Complex" type:*** *If the parameter is a complex type then the Web API catches the value from the message body. It uses the media type formatters for catching the value from the body.*

➢ *We can also force Web API to get the parameter value from the request body or request URL by using FromUri and FromBody attribute.*

FromUri Attribute

- *It forces the Web API to read the parameter value from the requested URL. In the following example, I have defined the complex type and forced the Web API to read the value for the complex type from the requested parameter.*
- *Get method of Employee controller class shown below:*

```
public HttpResponseMessage Get([FromUri] TestData data)
{……
    return Request.CreateResponse(HttpStatusCode.OK, true);

}
```

- *A client can put the value of ID and Name property of TestData class in the query string. Web API uses them to construct TestData class.*
-

FromBody Attribute

- *It forces the Web API to read the parameter value from the requested body. In the following example, I forced the Web API to read the value for simple type from the requested body by using FromBody attribute.*

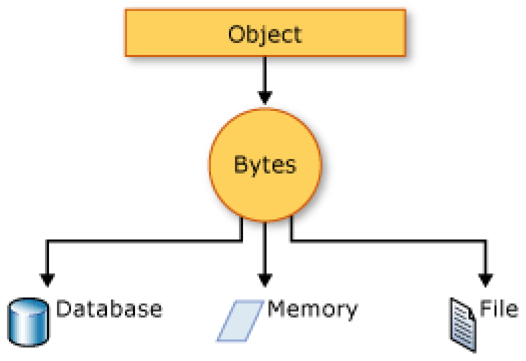
Example

```
[HttpPost]
public HttpResponseMessage Post([FromBody] string name)
{
    ……
    return Request.CreateResponse(HttpStatusCode.OK, true);
}
```

- *In the above example, I posted one parameter to the requested body, and Web API used a media-type formatter to read the value of the name from the requested body.*
- *To select the media formatter, Web API uses the content-type header. In the above example, the content type is set to "application/JSON" and the requested body contains a string value, so it binds to the parameter at Web API. It supports the JSON string, not the JSON object, so only one parameter is allowed to read from the message body.*

26.4. Serialization

➢ *Serialization in C# is the process of bringing an object into a form that it can be written on stream. It's the process of converting the object into a form so that it can be stored on a file, database, or memory; or, it can be transferred across the network. Its main purpose is to save the state of the object so that it can be recreated when needed.*



Deserialization:

- *As the name suggests, deserialization in C# is the reverse process of serialization. It is the process of getting back the serialized object so that it can be loaded into memory. It resurrects the state of the object by setting properties, fields etc.*

- ***Types***

  - *Binary Serialization*
  - *XML Serialization*
  - *JSON Serialization*

JSON serialization

- *The System.Text.Json namespace contains classes for JavaScript Object Notation (JSON) serialization and deserialization. JSON is an open standard that is commonly used for sharing data across the web.*
- *JSON serialization serializes the public properties of an object into a string, byte array, or stream that conforms to the RFC 8259 JSON specification. To control the way JsonSerializer serializes or deserializes an instance of the class:*
  - *Use a JsonSerializerOptions object*
  - *Apply attributes from the System.Text.Json.Serialization namespace to classes or properties*
  - *Implement custom converters*

Binary serialization

- *The System.Runtime.Serialization namespace contains classes for binary and XML serialization and deserialization.*
- *Binary serialization uses binary encoding to produce compact serialization for uses such as storage or socket-based network streams. In binary serialization, all members, read-only members, are serialized, and performance is enhanced.*

XML serialization:

- *It serializes the public fields and properties of an object, or the parameters and returns values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to XML. System.Xml.Serialization contains classes for serializing and deserializing XML. You apply attributes to classes and class members to control the way the XmlSerializer serializes or deserializes an instance of the class.*

26.5. Routing

- *Routing is responsible for mapping incoming browser requests to particular controller actions (or Razor Pages handlers). This section covers how routing differs between ASP.NET MVC (and Web API) and ASP.NET Core (MVC, Razor Pages, and otherwise).*
- *ASP.NET MVC offers two approaches to routing:*

  1. *The route table, which is a collection of routes that can be used to match incoming requests to controller actions.*
  2. *Attribute routing, which performs the same function but is achieved by decorating the actions themselves, rather than editing a global route table.*

Conventional routing

- *With conventional routing, you set up one or more conventions that will be used to match incoming URLs to* endpoints *in the app. In ASP.NET Core, these endpoints may be controller actions, like in ASP.NET MVC or Web API. The endpoints could also be Razor Pages, Health Checks, or SignalR hubs. All of these routable features are configured in a similar fashion using endpoints:*

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapHealthChecks("/healthz").RequireAuthorization();
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
    endpoints.MapRazorPages();
});
```

- *The preceding code is used (in addition to UseRouting) to configure various endpoints, including Health Checks, controllers, and Razor Pages. For controllers, the above configuration specifies a default routing convention, which is the fairly standard {controller}/{action}/{id?} pattern that's been recommended since the first versions of ASP.NET MVC.*

Attribute routing

- *Attribute routing in ASP.NET Core is the preferred approach for configuring routing in controllers. If you're building APIs, the [ApiController] attribute should be applied to your controllers. Among other things, this attribute requires the use of attribute routing for actions in such controller classes.*
- *Attribute routing in ASP.NET Core behaves similarly in ASP.NET MVC and Web API. In addition to supporting the [Route] attribute, however, route information can also be specified as part of the HTTP method attribute:*

```
[HttpGet("API/products/{id}")]
public async ActionResult<Product> Details(int id)
{
    // ...
}
```

- *As with previous versions, you can specify a default route with placeholders, and add this at the controller class level or even on a base class. You use the same [Route] attribute for all of these cases. For example, a base API controller class might look like this:*

```
[Route("API/{controller}/{action}/{id?:int}")]
public abstract class BaseApiController :
ControllerBase, IApiController
{
    // ...
}
```

- Using this attribute, classes inheriting from this type would route URLs to actions based on the controller name, action name, and an optional integer id parameter.

26.6. Config

- Web API supports code-based configuration. We created a simple Web API project. Web API project includes default WebApiConfig class in the App_Start folder and also includes Global.asax.
- The Global.asax file in an MVC application allows us to write code that we want to run at the application level, such as Application_BeginRequest, Application_error, Application_Start, session_start, session_end, etc.
- The App_Start folder of the MVC application is used to contain the class files which are needed to be executed at the time the application starts. The classes like BundleConfig, FilterConfig, IdentityConfig, RouteConfig, and Startup. Auth etc. are stored within this folder. So, in simple words, we can say that configuration related class files are stored in App_Start.
- The Web.config file of an MVC application is one of the most useful and important files which contains the application-level configurations.