**ASP.NET Web API Notes**

**Web API Controller Basics**

1. The name of a controller class must end with Controller.
2. Controllers must be derived from the System.Web.Http.ApiController class.
3. All public methods in a controller are called **action methods**.
4. Web API determines which controller and action method to execute based on the:
   * Incoming request URL.
   * HTTP verb (GET, POST, PUT, PATCH, DELETE).

**Action Methods:**

* Action methods can start with an HTTP verb or apply attributes such as [HttpGet], [HttpPost], etc., for methods that do not follow the verb naming convention.
* The return type of an action method can be any primitive or complex type.

**Routing in Web API**

Routing determines how HTTP requests map to specific controllers and actions.

**Types of Routing:**

1. **Convention-Based Routing:**
   * Uses route templates to determine the controller and action method.
   * Defined in WebApiConfig.Register.
   * config.Routes.MapHttpRoute(
   * name: "DefaultApi",
   * routeTemplate: "api/{controller}/{id}",
   * defaults: new { id = RouteParameter.Optional }
   * );
     + routeTemplate: Defines the URL pattern.
       - api: Fixed segment of the URL.
       - {controller}: Dynamically maps to the controller class.
       - {id}: Optional parameter.
     + defaults: Specifies default values for parameters if not provided in the request.
   * Advantages:
     + Less boilerplate.
     + Suitable for standard CRUD operations.
     + Acts as a fallback when attribute routes are not defined.
2. **Attribute Routing:**
   * Enabled using config.MapHttpAttributeRoutes().
   * Routes are defined directly on controllers or actions using attributes like [Route("api/products/{id}")].
   * Provides more flexibility compared to convention-based routing.
   * Example:
   * [Route("api/products/{id}")]
   * public IHttpActionResult GetProduct(int id)
   * {
   * // Action logic
   * }

**Parameter Binding**

1. **Primitive Types:**
   * Parameters like int, bool, string, etc., are bound from the query string.
2. **Complex Types:**
   * Parameters like objects are bound from the request body by default.
3. **Customization:**
   * Parameter binding can be customized using attributes like [FromUri] and [FromBody].

**HttpResponseMessage**

* Web API controllers return an HttpResponseMessage object to configure the response.
* Allows setting:
  + Status code.
  + Response content.
  + Error messages.

**Example:**

public HttpResponseMessage Get(int id)

{

var student = GetStudentFromDB(id);

if (student == null)

{

return Request.CreateResponse(HttpStatusCode.NotFound, id);

}

return Request.CreateResponse(HttpStatusCode.OK, student);

}

**IHttpActionResult (Web API 2+)**

* Introduced in Web API 2.
* Simplifies the response creation process.
* Similar to ActionResult in ASP.NET MVC.

**Example:**

public IHttpActionResult Get(int id)

{

var student = GetStudentFromDB(id);

if (student == null)

{

return NotFound();

}

return Ok(student);

}

**Web API Configuration**

* Web API configuration is done in the WebApiConfig.Register() method.
* Configurable components include:
  + Routes.
  + Formatters.
  + Filters.
  + DependencyResolver.
  + MessageHandlers.
  + ParameterBindingRules.
  + Properties and services.

**Configuration Example:**

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Enable attribute routing

config.MapHttpAttributeRoutes();

// Define convention-based routing

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

**Request/Response Data Formats**

* Web API supports multiple formats like JSON, XML, BSON, etc.
* **Media Type (MIME):**
  + Specified in HTTP request headers using Accept and Content-Type.
  + Accept: Specifies the expected response format (e.g., application/json).
  + Content-Type: Specifies the request body format (e.g., application/xml).

**Automatic Format Conversion:**

* Web API automatically converts:
  + Request data into CLR objects.
  + CLR objects into response data.
* Default formatters support JSON, XML, and form-urlencoded data.

This document organizes the unstructured notes into clear sections for better understanding and reference. Let me know if you need further refinements or additional details!