**Resumen – Creación del proyecto en Visual Studio 2022**

1. **Abrir y crear el proyecto**
   * En Visual Studio 2022 → **Crear nuevo proyecto** → buscar **App** → seleccionar **ASP.NET Core Web App** (versión **C#**, no F#).
   * Este curso se centra en **desarrollo backend**.
2. **Ubicación del proyecto**
   * Guardar en una carpeta **fuera de la nube** (OneDrive, Google Drive, Dropbox) para evitar lentitud por sincronización.
3. **Configuración inicial**
   * Usar **.NET 8** (aunque la mayoría funciona desde .NET 5).
   * Asegurarse de que esté marcada la opción **Usar controladores** (no trabajar en modo minimalista).
4. **Ejecución del proyecto**
   * Correr con HTTPS → se abre una consola mostrando los **puertos** (HTTP y HTTPS).
   * El navegador abrirá **Swagger**, que sirve para documentar la API y facilitar el trabajo al frontend.
   * Swagger muestra un método de ejemplo (WeatherForecast) creado por defecto.
5. **Prueba del método inicial**
   * El endpoint GET de WeatherForecast devuelve un **JSON de ejemplo**, confirmando que el proyecto funciona correctamente.

**Resumen – Controladores en ASP.NET Core**

1. **Qué es un controlador**
   * Es una **clase** que **hereda de ControllerBase** (o Controller) en ASP.NET.
   * Representa un **punto de entrada** para que un usuario o proceso solicite algo a la aplicación.
   * Devuelve una **respuesta** que puede ser: número, cadena, vista HTML, JSON, XML, etc.
   * Es la base de un **endpoint**, es decir, una URL que ejecuta una función específica.
2. **Creación de un nuevo controlador**
   * Borrar controladores creados por defecto (WeatherForecast).
   * En la carpeta **Controllers** → **Agregar** → **Controlador de API en blanco**.
   * Nombrar el archivo con el sufijo Controller (ej.: OperationController).
   * Por convención, la URL para acceder será:

https://localhost:puerto/api/Operation

(sin incluir la palabra Controller).

1. **Estructura básica de un controlador**
   * Es una clase normal en POO, pero con funcionalidades propias de ASP.NET gracias a ControllerBase.
   * Puede contener métodos que se mapean a endpoints.
2. **Creación de un método en el controlador**
   * Ejemplo: método Add que retorna un **decimal**.
   * Recibe dos parámetros (decimal a, decimal b).
   * Retorna la suma de ambos.
   * Decorado con [HttpGet] para indicar que se accede mediante **HTTP GET**.
3. **Prueba del endpoint**
   * Ejecutar el proyecto y abrir Swagger para ver el método disponible.
   * Para probar en el navegador:
   * https://localhost:puerto/api/Operation?a=1&b=2
   * El resultado será la suma de los parámetros (ej.: 3).
   * Cambiando los valores (a=10&b=2) retorna 12.

**Resumen – Métodos de petición HTTP y uso de Postman**

1. **Métodos HTTP más comunes**
   * También llamados *verbos*.
   * Principales:
     + **GET** → Obtener un recurso.
     + **POST** → Crear o modificar en el servidor (insertar datos, crear archivos, etc.).
     + **PUT** → Actualizar un recurso existente.
     + **DELETE** → Eliminar un recurso.
   * Otros: PATCH, OPTIONS, etc.
2. **Asociación de verbos con métodos del controlador**
   * Un método en el controlador se asocia con un verbo usando atributos como [HttpGet], [HttpPost], [HttpPut], [HttpDelete].
   * El verbo definido en la solicitud determina **qué método se ejecuta**.
   * Ejemplo en OperationController:
     + GET → suma (a + b).
     + POST → resta (a - b).
     + PUT → multiplicación (a \* b).
     + DELETE → división (a / b).
3. **Acceso a endpoints**
   * Formato de URL:

https://localhost:puerto/api/Operation?a=10&b=22

* + Los parámetros se pasan con ?param=valor y & para separarlos.

1. **Pruebas con Swagger**
   * Swagger documenta y permite probar la API rápidamente.
   * Muestra cada método con un color según el verbo.
   * Permite ingresar valores y ejecutar la solicitud con *Try it out*.
2. **Pruebas con Postman**
   * Herramienta más completa para probar y organizar solicitudes HTTP.
   * Permite guardar, agrupar en colecciones y compartir solicitudes.
   * Más usada en entornos reales que Swagger para pruebas de APIs.

**Resumen – Envío de datos en el Body de la solicitud HTTP**

1. **Partes de una solicitud HTTP**
   * **Ruta (URL)** → dónde se envía la solicitud.
   * **Verbo o método HTTP** → acción a realizar (GET, POST, PUT, DELETE).
   * **Body (cuerpo)** → datos enviados junto con la solicitud, comúnmente en formato **JSON**.
2. **Uso del Body**
   * Permite enviar información más estructurada (ej.: datos de una venta, nuevo cliente, etc.).
   * Los datos enviados en el Body se usan principalmente en **POST** y **PUT** (también posible en DELETE, pero poco común).
   * En **GET** no se suelen enviar datos en el body (no estándar).
3. **Ejemplo con JSON y clase en C#**
   * Se crea una clase Numbers con propiedades decimal A y decimal B.
   * En un método POST del controlador, se recibe un parámetro de tipo Numbers.
   * Postman envía un JSON en el Body:

{

"a": 15,

"b": 50

}

* + ASP.NET **deserializa automáticamente** el JSON y lo convierte en un objeto Numbers.

1. **Parámetros por URL vs Body**
   * URL: ?a=10&b=22 → parámetros simples.
   * Body: JSON estructurado → más flexible y limpio.
   * Es posible combinarlos, pero lo ideal es:
     + **GET** → parámetros en la URL.
     + **POST / PUT** → datos en el Body.
2. **Integración con Swagger y Postman**
   * Swagger detecta que el método ahora recibe un objeto y muestra el formato del Body requerido.
   * Postman configura automáticamente el Content-Type: application/json en los headers al seleccionar formato JSON.

**Resumen – Encabezados (Headers) en solicitudes HTTP**

1. **Qué son los encabezados (Headers)**
   * Información adicional incluida en la solicitud HTTP.
   * Pueden indicar:
     + Origen de la conexión.
     + Tipo de cliente (navegador, Postman, etc.).
     + Autenticación.
     + Configuración de contenido (tipo, tamaño, etc.).
   * Ejemplo: User-Agent, Host, Content-Length.
2. **Obtención de headers en ASP.NET**
   * Se usan atributos como [FromHeader] para recibir valores de headers.
   * Si el nombre del parámetro coincide con el nombre del header, se mapea automáticamente.

[FromHeader] string host

* + Si el nombre del header tiene caracteres especiales (como guiones), se especifica el nombre:

[FromHeader(Name = "Content-Length")] int contentLength

1. **Ejemplo de headers comunes**
   * Host → se obtiene directamente.
   * Content-Length → requiere Name en [FromHeader] por tener guion.
2. **Headers personalizados**
   * No definidos en el estándar oficial HTTP.
   * Se recomienda iniciar el nombre con X- (convención).
   * Ejemplo:
     + Header enviado: X-Sum: Pato
     + Lectura en backend:

[FromHeader(Name = "X-Sum")] string sum

1. **Casos de uso**
   * Muchos servicios externos utilizan headers personalizados para enviar datos específicos.
   * Útiles para autenticación, control de acceso, configuración extra o intercambio de datos sin incluirlos en el body o en la URL.