Unit 1 – Introduction to Web API (Easy Explanation)

Why API?

- Suppose you start a new business and create a **website** to show your products using technologies like **ASP.NET MVC, PHP, JSP**, etc.
- You also need a **database** (like SQL Server, MySQL, Oracle, etc.) to store product data.
- The website and database together make a **dynamic website** that users can interact with.
- Later, your business grows — now you also want **Android** and **iOS** apps.
- That means you have **three different applications** (Website, Android, iOS) — but **only one database**.

So, how will all three apps talk to the same database safely?

Problems Without Web API

- Without Web API, every app (website, Android, iOS) would need its own connection to the database.
- This creates **security risks** and **code duplication**.
- The better way: create **one Web API project** that connects to the database.
- Then, all three apps talk **only to the Web API**, not directly to the database.

✅ The **Web API** acts like a **bridge (mediator)** between:

- **Frontend (Website / App)** and
- **Backend (Database)**

All business logic stays in the Web API.

Introduction to Web API

- **API** = *Application Programming Interface*
- **Web API** is a framework to build **HTTP services** that can be used by:
    - Web browsers
    - Mobile apps
    - Other servers
- It allows **different software systems** to **communicate** over the Internet using **HTTP**.
- It's very useful for **data exchange** and **integration** between different applications.

Advantages of Web API

1. **No Code Duplication**
   Write logic once in Web API; all apps can reuse it.
2. **Easily Extend Applications**
   You can later add more platforms (like mobile apps) without rewriting logic.
3. **Abstraction (Hiding Logic)**
   Frontend developers can't see backend logic — only use API endpoints.
4. **Security**
   Applications can't access the database directly — safer and more controlled.

Key Characteristics of Web APIs

1. **HTTP-based Communication**
   Uses standard **HTTP methods** (GET, POST, PUT, DELETE).
2. **Data Exchange Formats**
   Usually uses **JSON** or **XML** for sending/receiving data.
   JSON is more popular because it's simple.
3. **RESTful Architecture**
   Follows **REST principles** – stateless, resource-based, and uses URLs.
4. **Authentication & Authorization**
   Uses security methods like:
   - API Keys
   - OAuth
   - JWT (JSON Web Tokens)

HTTP (HyperText Transfer Protocol)

1. How Browser and Server Communicate

- When you type a URL and press Enter:
  - Browser sends a **Request** to the server.
  - Server sends back a **Response**.
- This communication happens through **HTTP Protocol**.

Client: Browser or mobile app
Server: Web server
Request → Response = HTTP Communication

---

2. What is HTTP?

- Full form: **HyperText Transfer Protocol**

- It’s an **application layer protocol** used to communicate on the web.
- It sends and receives **web pages, images, videos, files**, etc.
- Works on a **request-response** model:
    - Client sends a request.
    - Server sends a response.

---

3. HTTP Request Components

- A **request** is sent from client → server.
- Components of a Request:
    - **URL**
    - **HTTP Verb (GET, POST, etc.)**
    - **Header**
    - **Body**

4. HTTP Response Components

- A **response** is sent from server → client.
- Components of a Response:
    - **Status Code**
    - **Response Header**
    - **Data (Body)**

---

4. HTTP Verbs / Methods

Used to perform actions on resources:

Verb	Meaning
GET	To read/fetch data
POST	To create new data
PUT	To update existing data
DELETE	To remove data

---

5. HTTP Status Codes

These are 3-digit numbers that tell the result of a request.

Type	Range	Meaning
1XX	Informational	Request received, still processing
2XX	Success	Request successful (e.g., 200 OK)
3XX	Redirection	Need to take further action (e.g., 301 Moved)
4XX	Client Error	Problem in request (e.g., 404 Not Found)
5XX	Server Error	Problem on server (e.g., 500 Internal Error)

JSON (JavaScript Object Notation)

- A **lightweight format** to share data between client and server.
- Easy to **read, write, and understand**.
- Uses text format like {}, [], :, and "".
- Data is stored in **name-value pairs**.

Example:

```
{
  "StudentID": 101,
  "Name": "Naimish",
  "City": "Rajkot"
}
```

Used For: Sending data between server and app instead of XML.

Common Libraries in .NET:

- Newtonsoft.Json
- System.Text.Json

Serialization & Deserialization

- **Serialization:**
  Convert **object → JSON string** (for storing/sending).
- **Deserialization:**
  Convert **JSON string → object** (for reading/using again).

Example:

```
string jsonData = JsonConvert.SerializeObject(std); // Serialize
var student = JsonConvert.DeserializeObject<Student>(jsonData); // Deserialize
```

Different JSON Data Types

Type	Description
Object	{} contains key-value pairs
Array	[] contains multiple values
String	Text inside quotes
Number, Boolean, Null	As usual

Creating and Consuming Web API

Steps to Create a Web API Project in Visual Studio

1. Open Visual Studio → Create new project → choose **ASP.NET Core Web API**
2. Set Project Name & Location → Next
3. Select Framework = **.NET 8.0 (LTS)**
4. Check **Enable OpenAPI Support** → Create

Steps to Build a Working Web API

1. **Create New Project**
   Delete default files:
   - WeatherForecast.cs
   - WeatherForecastController.cs
2. **Install Required Packages (via NuGet)**
   - Microsoft.EntityFrameworkCore
   - Microsoft.EntityFrameworkCore.SqlServer
   - Microsoft.EntityFrameworkCore.Tools
3. **Create Models Folder**
4. **Use Package Manager Console:**
5. Scaffold-DbContext "server=Naimish; database=NRVDemo; trusted_connection=true; TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

➤ This command generates models and DbContext from an existing database (Database-First approach).

6. **Edit DbContext file** – remove unnecessary lines.

7. **Add Connection String** in appsettings.json.
8. **Add Dependency Injection** in Program.cs.
9. **Create API Controller (e.g., StudentAPIController.cs)** inside **Controllers** folder.
10. Write your API logic inside that controller.