Unit 1: Introduction to Web API

- **Why API?**
  - A website + database works fine at first.
  - But when you also need Android/iOS apps, all must connect to the same database.
  - Giving direct DB access to every app is risky.
  - **Solution → Web API acts as a mediator** (apps talk to API, API talks to DB).
- **Problems without Web API:**
  - Apps directly connect to DB → security risks, code duplication, maintenance problems.
  - With API → one central logic for all apps.
- **What is Web API?**
  - **API (Application Programming Interface):** A set of rules for communication.
  - **Web API:** API built over HTTP (web protocol) so apps can exchange data (usually JSON).
  - Used by browsers, mobile apps, servers, etc.
- **Advantages:**
  - Avoids code duplication
  - Abstraction (frontend doesn't see DB logic)
  - Security (DB hidden)
  - Extendability (easily add more clients later)
- **HTTP Basics:**
  - Works on **Request–Response** model.
  - Client → sends **Request** (URL, headers, body).
  - Server → sends **Response** (status code, headers, data).
- **HTTP Methods (verbs):**
  - GET (read), POST (create), PUT (update), DELETE (remove).
- **Status Codes:**
  - 1xx → Info, 2xx → Success (200 OK),
  - 3xx → Redirection, 4xx → Client error (404 Not Found),
  - 5xx → Server error (500 Internal Server Error).
- **JSON:**

- Lightweight data format (key–value, arrays).
- Easy to read/write, used instead of XML.
- Example:

```
{ "id": 1, "name": "Riya" }
```

Unit 2: Entity Framework Core (EF Core) & LINQ

- **What is EF Core?**
  - An **ORM (Object Relational Mapper)** → lets us use C# classes instead of SQL queries.
  - Example:
    - SQL: SELECT * FROM Students WHERE StudentId=1;
    - EF Core: _context.Students.Find(1);
- **Advantages of ORM:**
  - Less SQL code
  - Easy CRUD
  - Type safety
  - Migrations (DB auto-sync with code)
- **Approaches:**
  - **Code-First:** Start with classes → DB created automatically.
  - **Database-First:** Start with DB → classes generated.
- **Key Components:**
  - **DbContext:** Bridge to DB (manages connection + CRUD).
  - **DbSet<T>:** Represents a table.
- **Migrations:**
  - Track changes in model → update DB schema.
  - Commands: Add-Migration, Update-Database.
- **Async/Await:**
  - Allows non-blocking DB/API calls.
  - Example:

```
var students = await _context.Students.ToListAsync();
```

LINQ (Language Integrated Query):

- Query DB/objects/XML directly in C#.

- Two styles:
    - Query syntax → looks like SQL.
    - Method syntax → uses lambdas (Where, Select, OrderBy).

LINQ Common Operators:

- Filtering: Where
- Sorting: OrderBy, ThenBy
- Projection: Select, SelectMany
- Aggregation: Count, Sum, Average, Min, Max
- Distinct, Join, First/Single

Unit 3: Routing, Versioning & Fluent Validation

- **Routing:**
    - Maps URLs → controller actions.
    - **Convention-based:** /controller/action/id.
    - **Attribute-based:** Add [Route("api/products/{id}")] above method.
    - Parameters can be optional.
- **API Versioning:**
    - Needed because APIs evolve.
    - Types:
        1. **URL Versioning:** /api/v1/products
        2. **Query String:** /api/products?version=1
        3. **Header:** version: 1 in request header
        4. **Consumer-based:** Version fixed per client.
- **Fluent Validation:**
    - A library for building validation rules.
    - More flexible than [Required] data annotations.
    - Steps:
        1. Install FluentValidation.AspNetCore
        2. Create Validator class
        3. Write rules using RuleFor()
        4. Register in Program.cs

- Example:

```
RuleFor(x => x.Age).GreaterThan(18);
```

- **Conditional Validation:**
  - When() and Unless() for applying rules only under conditions.

Unit 4: AJAX & jQuery

- **JavaScript Basics:**
  - Makes pages interactive (forms, animations, events).
  - Data types: string, number, boolean, array, object, null, undefined.
  - Functions, loops, events.
- **jQuery:**
  - A JavaScript library (shorter, easier syntax).
  - **Selectors:** $("#id"), $(".class"), $("p").
  - **Events:** .click(), .dblclick(), .hover(), .on().
- **AJAX (Asynchronous JavaScript and XML):**
  - Updates page content **without reloading**.
  - Uses XMLHttpRequest / fetch() / jQuery $.ajax().
  - Example:

```
$.ajax({
 url: "/api/students",
 type: "GET",
 success: function(data){ console.log(data); }
});
```

- **Advantages:**
  - Faster (fetches only required data).
  - Smooth user experience (like Gmail, Google Maps).
  - Works with APIs (ASP.NET Core, jQuery, Fetch API).

Unit 5: Middleware, Authentication & Authorization

- **Middleware:**
  - Components in the HTTP request pipeline.

    - Each can process request, modify it, or pass to next.
    - Example: Logging → Authentication → Routing → Endpoint.
    - Order matters!
- **Built-in Middleware:**
    - Authentication, Routing, Static files, Error handling, etc.
- **Action Filters:**
    - Run before/after controller actions.
    - Types: Authorization, Action, Result, Exception.
    - Example: [Authorize].
- **Custom Middleware:**
    - Create your own by writing a class with InvokeAsync(HttpContext context).
- **Authentication vs Authorization:**
    - **Authentication:** Who are you? (login, credentials).
    - **Authorization:** What can you do? (permissions, roles).
- **JWT (JSON Web Token):**
    - Self-contained token for authentication.
    - Contains header, payload (user info), signature.
    - Stored in client, sent in each request.
- **Role-Based Access Control (RBAC):**
    - Assign permissions to roles (Admin, User, Guest).
    - Users get roles → roles decide access.
    - Implemented via [Authorize(Roles="Admin")].
- **Global Error Handling:**
    - Middleware can catch exceptions and return user-friendly messages.