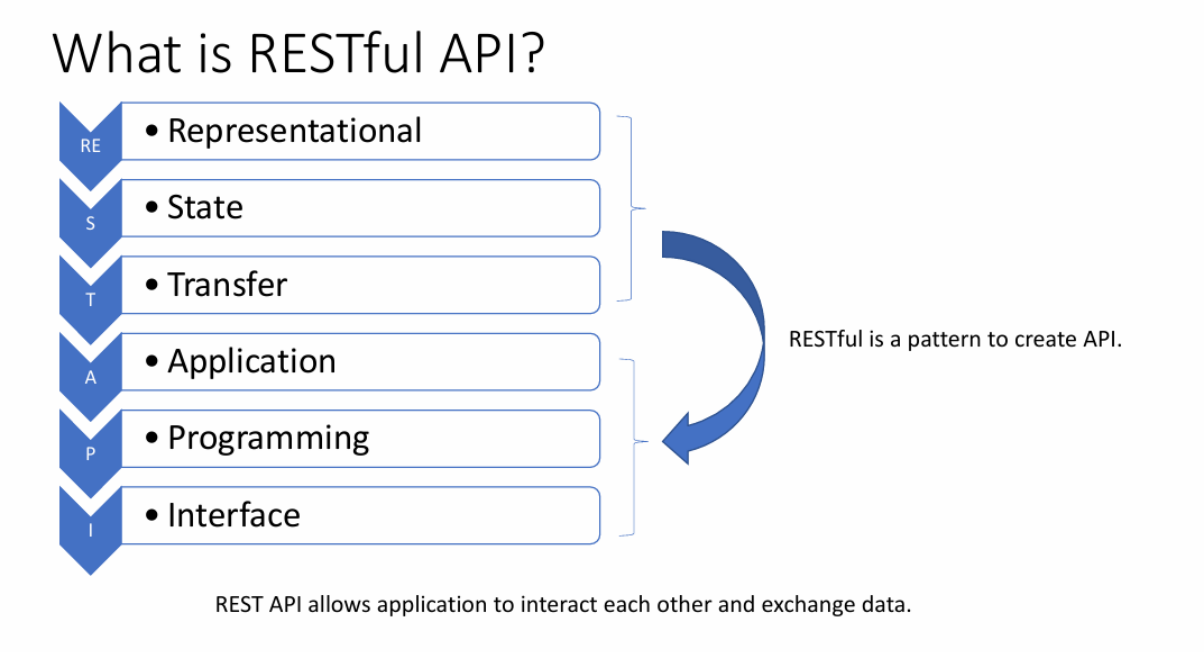
ASP.Net Core Minimal APIs Day 5

Our Path - empty project → hosting → middleware → routing → model binding → minimal APIs

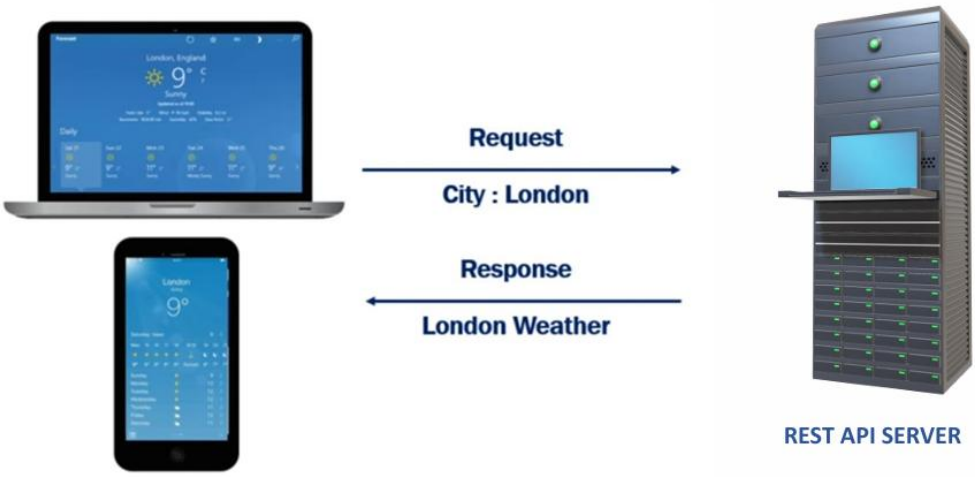
These notes explain:
✓ Concepts
✓ Architecture
✓ SQL connection handling
✓ Insert / Select / Update / Delete
✓ Parameterization (security)
✓ Error handling
✓ Best practices

ASP.NET Core – Topic 5: Minimal APIs

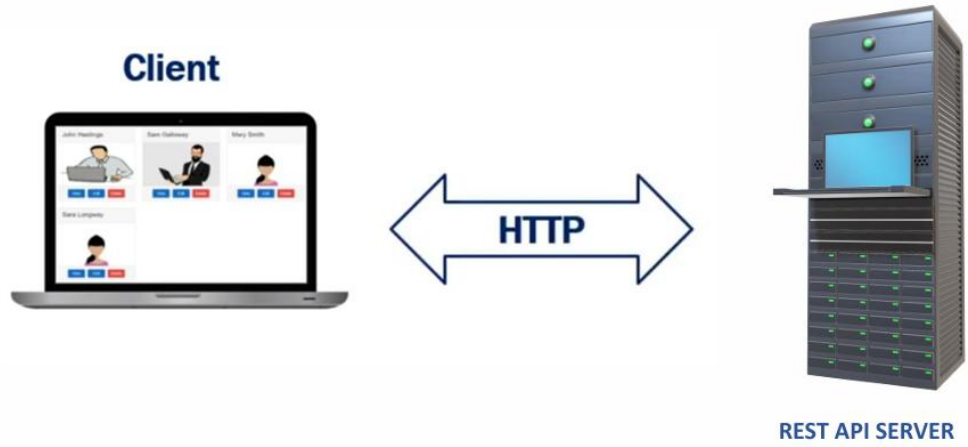
REST API



Example



Our Example



REST Constraints

Uniform Interface

- Define interface between server and client. How the REST API deliver, this constraint take care about this. Lets understand this but should know about
- Following technical terms.

Resource	HTTP Verb
➢ Product	➢ Get
➢ Employee	➢ Post
➢ Customer	➢ Put
➢ Order	➢ Delete

Resource and Verb Related to each other. See how Resource and Verb work together

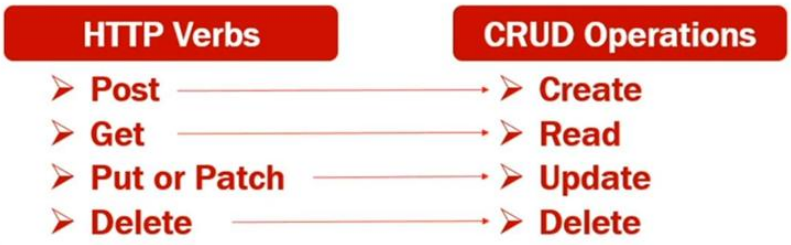
- https://sircltech.com/api/employees

Protocol | Domain Name | End Point

Each resource has an Unique Resource Identifier. So https://sircltech.com/api/employees is an URI where list of employees are available.

HTTP VERB

- HTTP verb decides what does the API “what to do with resource”. There are following http verbs with corresponding CRUD operation…



Note

- Combination of HTTP Verb and URI request to the server “ What to do with resource”

URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees/1	GET	Gets employee with Id = 1
.../api/employees	POST	Creates a new employee
.../api/employees/1	PUT	Updates employee with Id = 1
.../api/employees/1	DELETE	Deletes employee with Id = 1

1. What are Minimal APIs? (Simple Definition)

Minimal APIs allow you to build web APIs in ASP.NET Core using very little code, without needing controllers, MVC, or extra files. Everything is written inside Program.cs.

Minimal API is the **simplest & fastest** way to start building APIs.

2. Why Minimal APIs Were Introduced?

Because for small APIs, microservices, or learning purposes:

- MVC is too heavy
- Too many files (Controller, Model, Startup)
- Simple endpoints become too verbose

Minimal APIs solve this by providing:

✓ Quick setup
✓ Fewer files
✓ Easy learning path
✓ Fast performance
✓ Perfect for microservices

3. Where Minimal APIs Work Well?

- CRUD operations
- Microservices
- Internal APIs
- Learning routing + model binding
- Small/medium applications
- Prototyping

4. Minimal API Basic Structure

The minimal API structure inside Program.cs:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

// endpoints come here

app.Run();
```

Everything happens between:

```
var app = builder.Build();
app.Run();
```

5. Minimal API for GET Requests

Example 1 — Basic GET:

```
app.MapGet("/", () => "Hello from Minimal API!");
```

Example 2 — GET with Route Parameter:

```
app.MapGet("/hello/{name}", (string name) =>
{
   return $"Hello, {name}";
});
```

6. Minimal API with Multiple Parameters

```
app.MapGet("/add/{a}/{b}", (int a, int b) =>
{
   return a + b;
});
```

7. Minimal API with Query String Binding

```
app.MapGet("/search", (string? keyword, int? page) =>
{
   return $"Keyword = {keyword}, Page = {page}";
});
```

8. Minimal API with Route + Query Together

```
app.MapGet("/product/{id}", (int id, string? color) =>
{
   return $"Product ID: {id}, Color: {color}";
});
```

9. Minimal API with JSON Body (POST)

```
public class Student
{
   public string Name { get; set; }
   public int Age { get; set; }
}
```

Now create a POST endpoint:

```
app.MapPost("/student/add", (Student student) =>
{
   return $"Student {student.Name} (Age {student.Age}) added!";
});
```

Send JSON Body:

```
{
   "name": "Ravi",
   "age": 20
}
```

10. Binding Collections in Minimal API

```
app.MapGet("/tags", (string[] items) =>
{
   return items;
});
```

URL:

```
/tags?items=a&items=b&items=c
```

11. Minimal API with Headers

```
app.MapGet("/device", (HttpRequest req) =>
{
   var agent = req.Headers["User-Agent"].ToString();
   return $"User-Agent: {agent}";
});
```

12. Minimal API with Dependency Injection

```
builder.Services.AddSingleton<TimeService>();


public class TimeService
{
   public string Now() => DateTime.Now.ToString();
}
```

Use it in endpoint:

```
app.MapGet("/time", (TimeService svc) =>
{
   return svc.Now();
});
```

13. Minimal API Return Types

Minimal APIs automatically format the response based on the returned type:

Return Type	Output
string	**text response**
int, double	**number**
object	**JSON**
array/list	**JSON array**
IResult	**rich response (status codes etc.)**

Examples:

```
return Results.Ok(student);
return Results.NotFound("Student not found");
return Results.Created("/student/1", student);
```

14. Status Code Responses Using Results Class

```
app.MapGet("/check/{age}", (int age) =>
{
   if (age < 18) return Results.BadRequest("Too young!");
   return Results.Ok("Valid age");
});
```

15. Minimal API CRUD Example

1. Create in-memory list:

```
List<Student> students = new();
```

2. GET All

```
app.MapGet("/students", () => students);
```

3. GET by ID

```
app.MapGet("/students/{id}", (int id) =>
{
   var s = students.FirstOrDefault(x => x.Id == id);
   return s is null ? Results.NotFound() : Results.Ok(s);
});
```

4. POST add

```
app.MapPost("/students", (Student s) =>
{
   students.Add(s);
   return Results.Ok(s);
});
```

5. PUT update

```
app.MapPut("/students/{id}", (int id, Student updated) =>
{
   var s = students.FirstOrDefault(x => x.Id == id);
   if (s is null) return Results.NotFound();

   s.Name = updated.Name;
   s.Age = updated.Age;
```

```
    return Results.Ok(s);
});
```

6. DELETE

```
app.MapDelete("/students/{id}", (int id) =>
{
    var s = students.FirstOrDefault(x => x.Id == id);
    if (s is null) return Results.NotFound();

    students.Remove(s);
    return Results.Ok(s);
});
```

16. Minimal APIs Internals (How They Work)

Request → Kestrel → Routing → Model Binding → Minimal API Lambda → Response

Minimal APIs are simply:

- A **lambda function**
- Registered as an **endpoint**
- Using **model binding** to fill parameters
- Returning **IResult or object**

17. Differences: Minimal API vs MVC

Feature	**Minimal API**	**MVC**
Files	Only Program.cs	Multiple files
Speed	Faster	Slightly heavier
Best for	Microservices, simple APIs	Large applications
Architecture	Functional endpoints	Controller-based
Views	No	Yes
Filters	Limited	Full support
Attributes	Limited	Full

18. Minimal API Pros and Cons

✓ **Pros:**

- Minimal coding
- Super-fast
- Best for small services
- Clean routing
- Great for learning core concepts

❌ **Cons:**

- Harder to manage large apps
- No built-in filters
- Limited attribute usage
- Hard to organize many endpoints

Exercises for Students

Exercise 1:

Create /square/{n} endpoint → return square of number.

Exercise 2:

Create /user/register POST endpoint → bind User JSON → return welcome message.

Exercise 3:

Create /sum?x=10&y=20 → return sum.

Exercise 4:

Create /deviceinfo → read User-Agent header.

Exercise 5:

Build full CRUD for Book model (in memory).

ASP.NET Core – Minimal APIs with ADO.NET

1. Why Teach ADO.NET Before Dapper or EF Core?

Because ADO.NET shows the **real internal working** of database access:

- How to open a connection
- How SQL commands are executed
- How SqlDataReader returns rows
- Why parameterized queries matter
- Why ORMs exist (to reduce boilerplate)

Understanding ADO.NET makes students better developers even if they later use Dapper or EF Core.

2. What is ADO.NET? (Simple Definition)

ADO.NET is a low-level data access technology in .NET that allows applications to connect to a database, execute SQL commands, and read results.

It uses core classes such as:

- SqlConnection
- SqlCommand
- SqlDataReader
- SqlParameter
- SqlDataAdapter

Minimal APIs can work directly with ADO.NET because everything is just C#.

3. Database Setup for Examples

Create a simple table in SQL Server:

```
CREATE TABLE Students (
   Id INT IDENTITY(1,1) PRIMARY KEY,
   Name NVARCHAR(100),
   Age INT
)
```

4. Add Connection String

In **appsettings.json**:

```
{
  "ConnectionStrings": {
    "Default": "Server=.;Database=MyDb;Trusted_Connection=True;TrustServerCertificate=True"
  }
}
```

Read it in Program.cs:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();


string connStr = builder.Configuration.GetConnectionString("Default");
```

5. Minimal API + ADO.NET – Basic Concept

Minimal API handler:

- Receives request
- Opens DB connection
- Executes SQL command
- Returns response

Example skeleton:

```
app.MapGet("/testdb", () =>
{
    using var conn = new SqlConnection(connStr);
    conn.Open();


    return "Connection OK";
});
```

6. READ (SELECT) – GET All Students

```
app.MapGet("/students", () =>
{
    var list = new List<Student>();
```

```
    using var conn = new SqlConnection(connStr);
    conn.Open();

    using var cmd = new SqlCommand("SELECT Id, Name, Age FROM Students", conn);

    using var reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        list.Add(new Student
        {
            Id = reader.GetInt32(0),
            Name = reader.GetString(1),
            Age = reader.GetInt32(2)
        });
    }

    return list;
});
```

✓ Demonstrates reading rows
✓ Shows how mapping works manually
✓ Allows students to appreciate ORMs later

7. READ (SELECT) – GET Student by ID

IMPORTANT: Always use **parameterized queries** to prevent SQL injection.

```
app.MapGet("/students/{id}", (int id) =>
{
    using var conn = new SqlConnection(connStr);
    conn.Open();

    using var cmd = new SqlCommand("SELECT Id, Name, Age FROM Students WHERE Id = @Id",
conn);
    cmd.Parameters.AddWithValue("@Id", id);
```

```
    using var reader = cmd.ExecuteReader();
    if (reader.Read())
    {
        var s = new Student
        {
            Id = reader.GetInt32(0),
            Name = reader.GetString(1),
            Age = reader.GetInt32(2)
        };

        return Results.Ok(s);
    }

    return Results.NotFound();
});
```

8. CREATE (INSERT) – POST with JSON Body

```
public class Student
{
    public int Id { get; set; }  // ignored for insert
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Endpoint:

```
app.MapPost("/students", (Student s) =>
{
    using var conn = new SqlConnection(connStr);
    conn.Open();

    using var cmd = new SqlCommand("INSERT INTO Students (Name, Age) VALUES (@Name, @Age)",
conn);
    cmd.Parameters.AddWithValue("@Name", s.Name);
```

```
    cmd.Parameters.AddWithValue("@Age", s.Age);

    int rows = cmd.ExecuteNonQuery();
    return rows > 0 ? Results.Ok("Inserted") : Results.BadRequest("Insert failed");
});
```

✓ Shows how model binding + SQL INSERT works together

9. UPDATE – PUT Request

```
app.MapPut("/students/{id}", (int id, Student s) =>
{
    using var conn = new SqlConnection(connStr);
    conn.Open();

    using var cmd = new SqlCommand(
        "UPDATE Students SET Name = @Name, Age = @Age WHERE Id = @Id", conn);

    cmd.Parameters.AddWithValue("@Id", id);
    cmd.Parameters.AddWithValue("@Name", s.Name);
    cmd.Parameters.AddWithValue("@Age", s.Age);

    int rows = cmd.ExecuteNonQuery();

    return rows > 0 ? Results.Ok("Updated") : Results.NotFound();
});
```

✓ Teaches update logic
✓ Introduces 404 Not Found

10. DELETE – DELETE Request

```
app.MapDelete("/students/{id}", (int id) =>
{
    using var conn = new SqlConnection(connStr);
    conn.Open();
```

```
    using var cmd = new SqlCommand("DELETE FROM Students WHERE Id = @Id", conn);
    cmd.Parameters.AddWithValue("@Id", id);

    int rows = cmd.ExecuteNonQuery();

    return rows > 0 ? Results.Ok("Deleted") : Results.NotFound();
});
```

✓ Shows delete pattern
✓ Teaches parameterized queries

11. Notes on Using ADO.NET in Minimal APIs

✓ Use using statements

Ensures connection closing.

✓ Use Parameterized Queries

Avoids SQL injection.

✓ Avoid Inline SQL Concatenation

BAD:

"WHERE Name = '" + name + "'"

GOOD:

"WHERE Name = @Name"

✓ Don't Open Connection Too Early

Open → Execute → Close.

✓ Minimal APIs Should Not Become Too Big

One file approach may get messy.

This helps justify switching to **Dapper** later.

12. Complete Minimal API + ADO.NET CRUD Structure

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

string connStr = builder.Configuration.GetConnectionString("Default");
```

```
// GET all
app.MapGet("/students", ...);

// GET by id
app.MapGet("/students/{id}", ...);

// POST insert
app.MapPost("/students", ...);

// PUT update
app.MapPut("/students/{id}", ...);

// DELETE remove
app.MapDelete("/students/{id}", ...);

app.Run();
```

Minimal API + ADO.NET Request Flow

Browser / Postman

↓

Request → Routing → Model Binding

↓

ADO.NET (open conn → SQL → reader → map)

↓

Minimal API Handler Returns Result

↓

Response → Client

15. Student Exercises

Exercise 1:

Create endpoint /students/search?name=Ravi → return students whose Name contains given text (LIKE query).

Exercise 2:

Create endpoint /students/age/{min}/{max} → return students between age range.

Exercise 3:

Handle invalid SQL operations gracefully (try-catch).

Exercise 4:

Add a created timestamp to table and return last inserted ID using:

SELECT SCOPE_IDENTITY();