**Web API Hands-On Guide with Examples**

**Submitted by – Richa Goyal(6365275)**

**Table of Contents**

1. [Introduction to Web API](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#introduction)
2. [Basic Web API Creation](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#basic-webapi)
3. [Swagger Integration](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#swagger-integration)
4. [Custom Model Classes and Controllers](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#custom-models)
5. [CRUD Operations](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#crud-operations)
6. [Authentication and Authorization](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#authentication)
7. [Custom Filters](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#custom-filters)
8. [CORS Configuration](https://claude.ai/chat/c2ccf778-dc66-4ba0-9844-a36382cbe942#cors-configuration)

**1. Introduction to Web API {#introduction}**

**What is Web API?**

Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is based on REST (Representational State Transfer) architecture.

**Key Features:**

* **Stateless**: Each request contains all information needed to process it
* **HTTP Methods**: GET, POST, PUT, DELETE for different operations
* **Content Negotiation**: Supports JSON, XML, and other formats
* **Cross-platform**: Works on Windows, Linux, and macOS

**HTTP Status Codes:**

* **200 OK**: Request successful
* **400 Bad Request**: Invalid request
* **401 Unauthorized**: Authentication required
* **404 Not Found**: Resource not found
* **500 Internal Server Error**: Server error

**2. Basic Web API Creation {#basic-webapi}**

**Step 1: Create .NET Core Web API Project**

**Input (Command):**

dotnet new webapi -n MyWebAPI

cd MyWebAPI

**Output (Project Structure):**

MyWebAPI/

├── Controllers/

│ └── ValuesController.cs

├── Properties/

│ └── launchSettings.json

├── appsettings.json

├── Program.cs

└── Startup.cs

**Step 2: Default ValuesController**

**Input (ValuesController.cs):**

[ApiController]

[Route("api/[controller]")]

public class ValuesController : ControllerBase

{

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

return new string[] { "value1", "value2" };

}

}

**Output (GET Request to /api/values):**

[

"value1",

"value2"

]

**3. Swagger Integration {#swagger-integration}**

**Step 1: Install Swashbuckle.AspNetCore**

**Input (Package Manager Console):**

Install-Package Swashbuckle.AspNetCore

**Step 2: Configure Swagger in Startup.cs**

**Input (Startup.cs - ConfigureServices):**

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "Web API Demo with Swagger",

Contact = new OpenApiContact()

{

Name = "John Doe",

Email = "john@example.com"

}

});

});

**Input (Startup.cs - Configure):**

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

**Output (Swagger UI):**

* Navigate to https://localhost:5001/swagger
* Interactive API documentation with "Try it out" functionality
* API endpoints listed with request/response examples

**4. Custom Model Classes and Controllers {#custom-models}**

**Step 1: Create Employee Model**

**Input (Models/Employee.cs):**

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Salary { get; set; }

public bool Permanent { get; set; }

public Department Department { get; set; }

public List<Skill> Skills { get; set; }

public DateTime DateOfBirth { get; set; }

}

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

**Step 2: Create EmployeeController**

**Input (Controllers/EmployeeController.cs):**

[ApiController]

[Route("api/[controller]")]

public class EmployeeController : ControllerBase

{

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

Department = new Department { Id = 1, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 1, Name = "C#" },

new Skill { Id = 2, Name = "JavaScript" }

},

DateOfBirth = new DateTime(1990, 1, 15)

}

};

}

[HttpGet]

[ProducesResponseType(200)]

public ActionResult<List<Employee>> Get()

{

return Ok(GetStandardEmployeeList());

}

}

**Output (GET Request to /api/employee):**

[

{

"id": 1,

"name": "John Doe",

"salary": 50000,

"permanent": true,

"department": {

"id": 1,

"name": "IT"

},

"skills": [

{

"id": 1,

"name": "C#"

},

{

"id": 2,

"name": "JavaScript"

}

],

"dateOfBirth": "1990-01-15T00:00:00"

}

]

**5. CRUD Operations {#crud-operations}**

**Step 1: Add PUT Method for Update**

**Input (EmployeeController.cs):**

[HttpPut("{id}")]

[ProducesResponseType(200)]

[ProducesResponseType(400)]

public ActionResult<Employee> Put(int id, [FromBody] Employee employee)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

var employees = GetStandardEmployeeList();

var existingEmployee = employees.FirstOrDefault(e => e.Id == id);

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

// Update employee data

existingEmployee.Name = employee.Name;

existingEmployee.Salary = employee.Salary;

existingEmployee.Permanent = employee.Permanent;

return Ok(existingEmployee);

}

**Input (POSTMAN - PUT Request):**

URL: PUT https://localhost:5001/api/employee/1

Headers: Content-Type: application/json

Body:

{

"id": 1,

"name": "John Updated",

"salary": 60000,

"permanent": true,

"department": {

"id": 1,

"name": "IT"

},

"skills": [],

"dateOfBirth": "1990-01-15T00:00:00"

}

**Output (Response):**

{

"id": 1,

"name": "John Updated",

"salary": 60000,

"permanent": true,

"department": {

"id": 1,

"name": "IT"

},

"skills": [],

"dateOfBirth": "1990-01-15T00:00:00"

}

**6. Authentication and Authorization {#authentication}**

**Step 1: Configure JWT Authentication**

**Input (Startup.cs - ConfigureServices):**

string securityKey = "mysuperdupersecret";

var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

**Step 2: Create AuthController**

**Input (Controllers/AuthController.cs):**

[ApiController]

[Route("api/[controller]")]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("token")]

public IActionResult GetToken()

{

var token = GenerateJSONWebToken(1, "Admin");

return Ok(new { token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new List<Claim>

{

new Claim(ClaimTypes.Role, userRole),

new Claim("UserId", userId.ToString())

};

var token = new JwtSecurityToken(

issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(10),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**Output (GET Request to /api/auth/token):**

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBZG1pbiIsIlVzZXJJZCI6IjEiLCJleHAiOjE2MjM5MjQ4MDAsImlzcyI6Im15U3lzdGVtIiwiYXVkIjoibXlVc2VycyJ9.example"

}

**Step 3: Protect Employee Controller**

**Input (Update EmployeeController.cs):**

[ApiController]

[Route("api/[controller]")]

[Authorize(Roles = "Admin")]

public class EmployeeController : ControllerBase

{

// ... existing methods

}

**Input (POSTMAN - Authorized Request):**

URL: GET https://localhost:5001/api/employee

Headers:

- Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

**Output (Unauthorized Request):**

{

"type": "https://tools.ietf.org/html/rfc7235#section-3.1",

"title": "Unauthorized",

"status": 401,

"traceId": "00-example-trace-id"

}

**7. Custom Filters {#custom-filters}**

**Step 1: Create Custom Authentication Filter**

**Input (Filters/CustomAuthFilter.cs):**

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var authHeader = context.HttpContext.Request.Headers["Authorization"];

if (authHeader.Count == 0)

{

context.Result = new BadRequestObjectResult("Invalid request - No Auth token");

return;

}

var authValue = authHeader.FirstOrDefault();

if (string.IsNullOrEmpty(authValue) || !authValue.StartsWith("Bearer"))

{

context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");

return;

}

base.OnActionExecuting(context);

}

}

**Step 2: Create Custom Exception Filter**

**Input (Filters/CustomExceptionFilter.cs):**

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

// Log exception to file

var logMessage = $"Exception: {exception.Message}\nStackTrace: {exception.StackTrace}\nTime: {DateTime.Now}";

File.AppendAllText("error.log", logMessage + "\n\n");

context.Result = new ObjectResult("An error occurred")

{

StatusCode = 500

};

context.ExceptionHandled = true;

}

}

**Input (Apply Filter to Controller):**

[CustomAuthFilter]

[ServiceFilter(typeof(CustomExceptionFilter))]

public class EmployeeController : ControllerBase

{

// ... methods

}

**Output (Without Authorization Header):**

{

"message": "Invalid request - No Auth token"

}

**8. CORS Configuration {#cors-configuration}**

**Step 1: Install CORS Package**

**Input (Package Manager Console):**

Install-Package Microsoft.AspNetCore.Cors

**Step 2: Configure CORS in Startup.cs**

**Input (Startup.cs - ConfigureServices):**

services.AddCors(options =>

{

options.AddPolicy("AllowAll",

builder =>

{

builder.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader();

});

});

**Input (Startup.cs - Configure):**

app.UseCors("AllowAll");

**Output (CORS Headers in Response):**

Access-Control-Allow-Origin: \*

Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS

Access-Control-Allow-Headers: Content-Type, Authorization

**Testing with POSTMAN**

**Collection Structure:**

Web API Tests/

├── Auth/

│ └── Get Token

├── Employee/

│ ├── Get All Employees

│ ├── Update Employee

│ └── Get Employee by ID

└── Values/

└── Get Values

**Sample Request Headers:**

{

"Content-Type": "application/json",

"Authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

**Sample Response Status Codes:**

* **200 OK**: Successful request
* **400 Bad Request**: Invalid input
* **401 Unauthorized**: Missing or invalid token
* **404 Not Found**: Resource not found
* **500 Internal Server Error**: Server error

**Summary**

This hands-on guide demonstrates:

1. **Basic Web API Creation**: Setting up a .NET Core Web API project
2. **Swagger Integration**: API documentation and testing interface
3. **Custom Models**: Creating complex data structures
4. **CRUD Operations**: Implementing Create, Read, Update, Delete operations
5. **JWT Authentication**: Securing APIs with JSON Web Tokens
6. **Custom Filters**: Implementing authentication and exception handling
7. **CORS Configuration**: Enabling cross-origin requests

Each section includes practical examples with actual input and output to help understand the implementation and testing process.