Customer Management System – Backend- User Guide

Backend : Dot Net Core, Web API, Entity Framework Core, SQL Server with Clear Architecture

✅ Steps to Setup and Run .NET Core Project (CMS)

▼ Step 1: Download or Clone the Project from GitHub

```
git clone <https://github.com/SheikGH/CMS_Backend.git>
```

Example: `git clone https://github.com/SheikGH/CMS_Backend.git`

📁 Step 2: Open the Solution in Visual Studio

1. Open **Visual Studio 2022+**
2. Click **"Open a project or solution"**
3. Navigate to the downloaded folder
4. Open `CMS.sln`

🔧 Step 3: Restore NuGet Packages

In **Visual Studio**:

- Right-click the **Solution → Restore NuGet Packages**
- Or use the terminal:

```
dotnet restore
```

⚙️ Step 4: Update `appsettings.json` (CMS.API/appsettings.json)

Example:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=YOUR_SERVER_NAME;Database=CMS;Trusted_Connection=True;MultipleActiveResultSets=
true"
  },
  "Jwt": {
    "Key": "Your_Secret_Key_Here",
    "Issuer": "CMSApp",
    "Audience": "CMSAppUsers",
    "DurationInMinutes": 60
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
```

```
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

🔐 Replace `YOUR_SERVER_NAME` with your actual SQL Server name (e.g., `localhost\\SQLEXPRESS`)
Replace `"Your_Secret_Key_Here"` with a **strong secret key**

□ Step 5: Create Database

1. Open **SQL Server Management Studio (SSMS)**
2. Run:

```
CREATE DATABASE CMS;
```

🎁 Step 6: Run Migrations and Update Database

In **Package Manager Console**:

1. Select **CMS.Infrastructure** as the default project
2. Run:

```
Add-Migration initialDb
Update-Database
```

Make sure `Startup Project` is set to `CMS.API`

🎁 Step 7: Install Required NuGet Packages

◆ CMS.Core

Install:

```
Install-Package MediatR -Version 12.2.0
Install-Package Microsoft.AspNetCore.Authentication.JwtBearer -Version 6.0.27
Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson -Version 6.0.31
Install-Package Microsoft.EntityFrameworkCore.Design -Version 7.0.16
Install-Package Newtonsoft.Json -Version 13.0.3
Install-Package Swashbuckle.AspNetCore -Version 6.5.0
```

◆ CMS.Infrastructure

```
Install-Package Microsoft.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.Design
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
```

◆ CMS.Application

```
Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection -Version 12.0.1
Install-Package MediatR -Version 12.2.0
```

```
Install-Package Microsoft.Extensions.Configuration.Abstractions -Version 7.0.0
Install-Package Microsoft.IdentityModel.Tokens -Version 6.35.0
Install-Package System.IdentityModel.Tokens.Jwt -Version 6.35.0
```

◆ CMS.API

```
Install-Package Microsoft.EntityFrameworkCore -Version 7.0.16
Install-Package Microsoft.EntityFrameworkCore.Design -Version 7.0.16
Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 7.0.16
Install-Package Microsoft.EntityFrameworkCore.Tools -Version 7.0.16
```

▶☐ Step 8: Run the Application

1. Set **CMS.API** as the **Startup Project**
2. Press `F5` or click **Run**
3. Swagger UI should open:

   `https://localhost:7067/swagger/index.html`

🔁 Step 9: Test Endpoints via Swagger or Postman

- ✅ **POST** `/api/Auth/login`
- ✅ **GET** `/api/Customers`
- ✅ **POST** `/api/Customers`
- ✅ **PUT** `/api/Customers/{id}`
- ✅ **DELETE** `/api/Customers/{id}`

✅ Project Structure Recap (Clean Architecture)

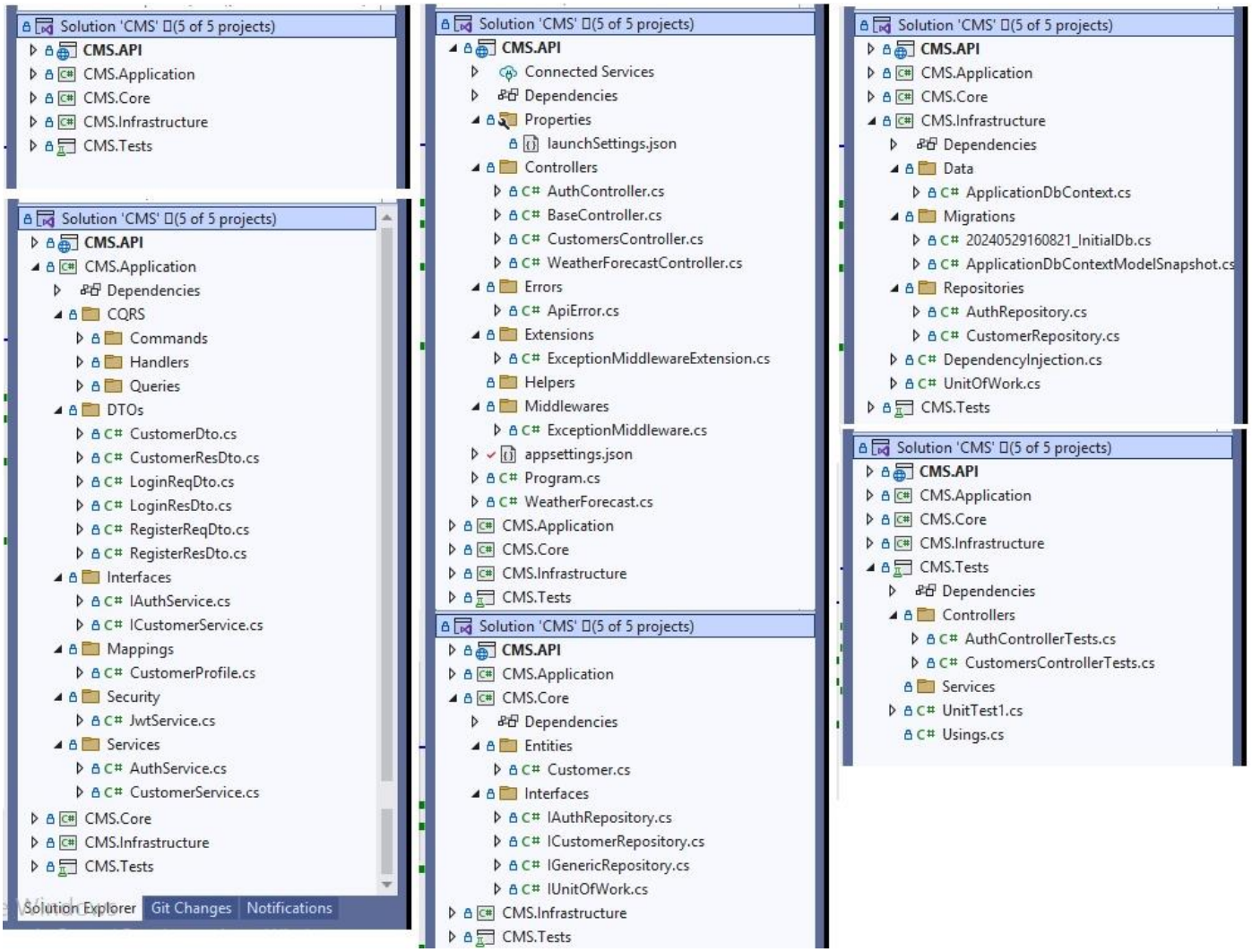
```
CMS/
├── CMS.Core/           → Domain Models, Interfaces
├── CMS.Infrastructure/ → EF Core, SQL, Repositories
├── CMS.Application/    → Services, DTOs, Mapping, Use Cases
├── CMS.API/            → Controllers, Middleware, Auth, Swagger
```

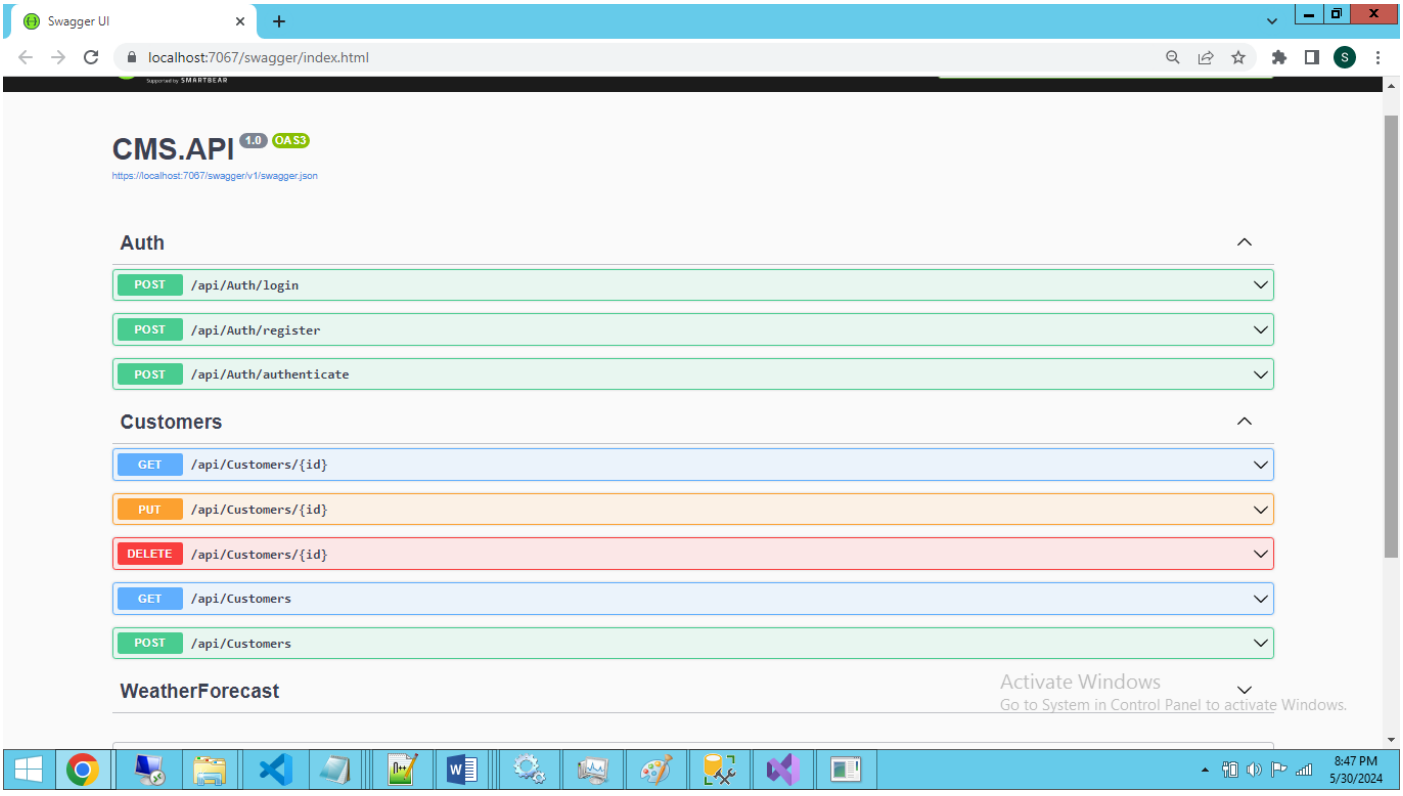
Backend: Dot Net Core, Web API, Entity Framework Core, SQL Server with Clear Architecture

✅ Backend API Summary

.NET Core Site URL: https://localhost:7067/swagger/index.html
Tech Stack: ASP.NET Core Web API + EF Core + SQL Server + JWT + Clean Architecture

Folder Structure: Clear Architecture



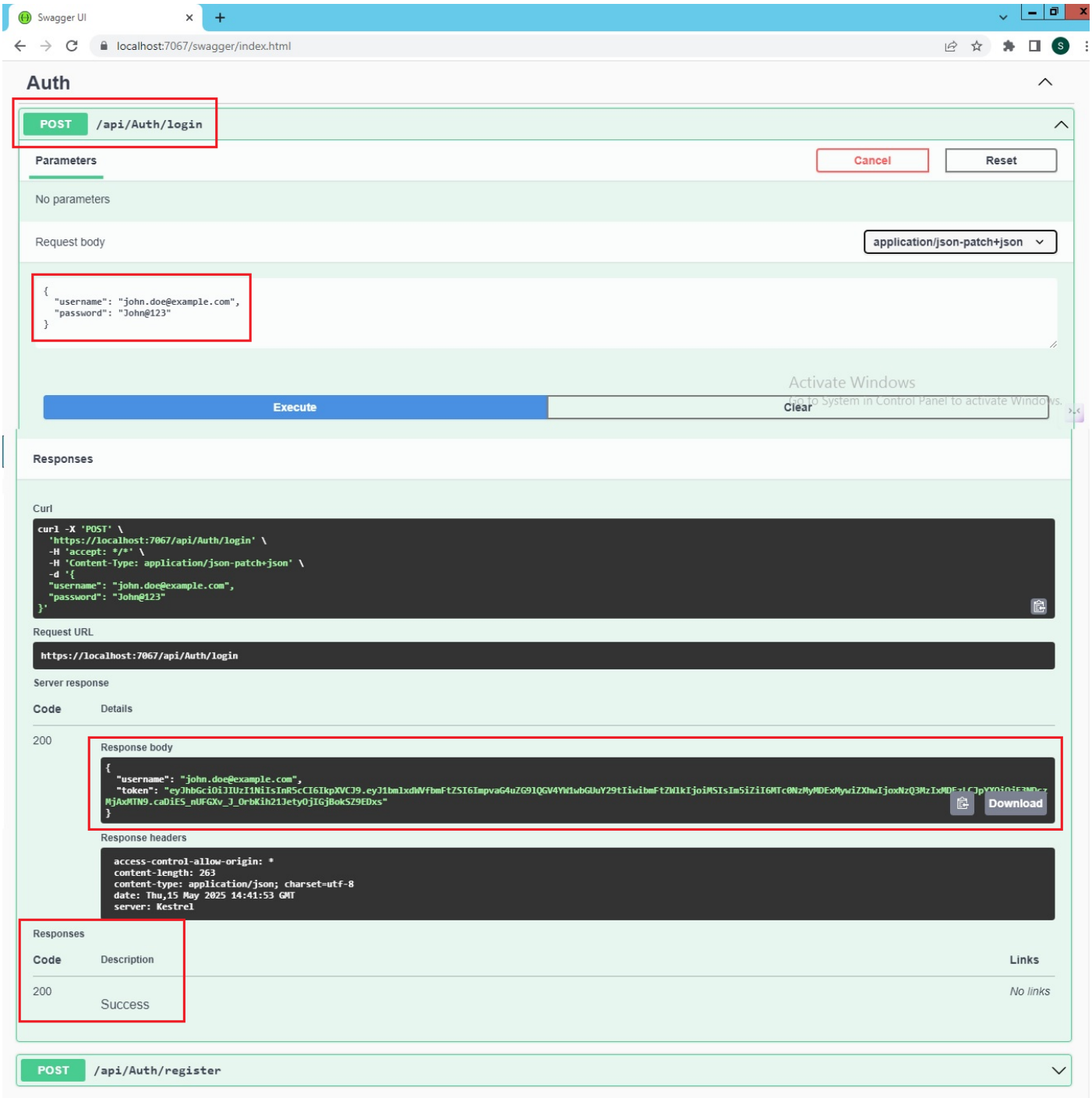


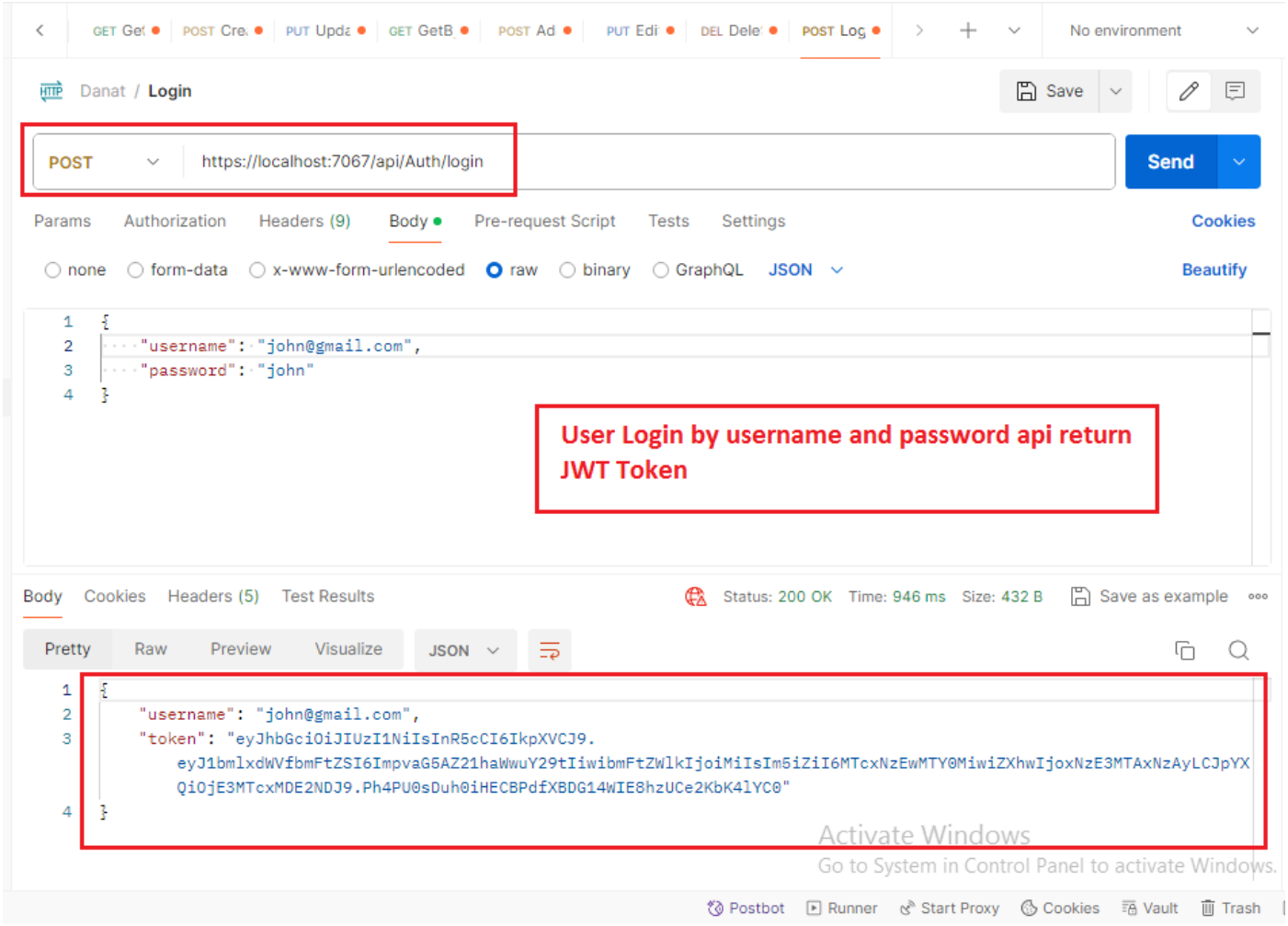
🔐 1. Authentication Endpoint

- **Purpose**: Login and generate JWT token
- **URL**: `GET https://localhost:7067/api/Auth/login`
- **Payload**:

```
{
    "username": "john@gmail.com",
    "password": "john"
}
```

- **Response**: JWT Token
- **Frontend Logic**:
    - Store token in `localStorage`
    - Send token in `Authorization` header for all protected requests





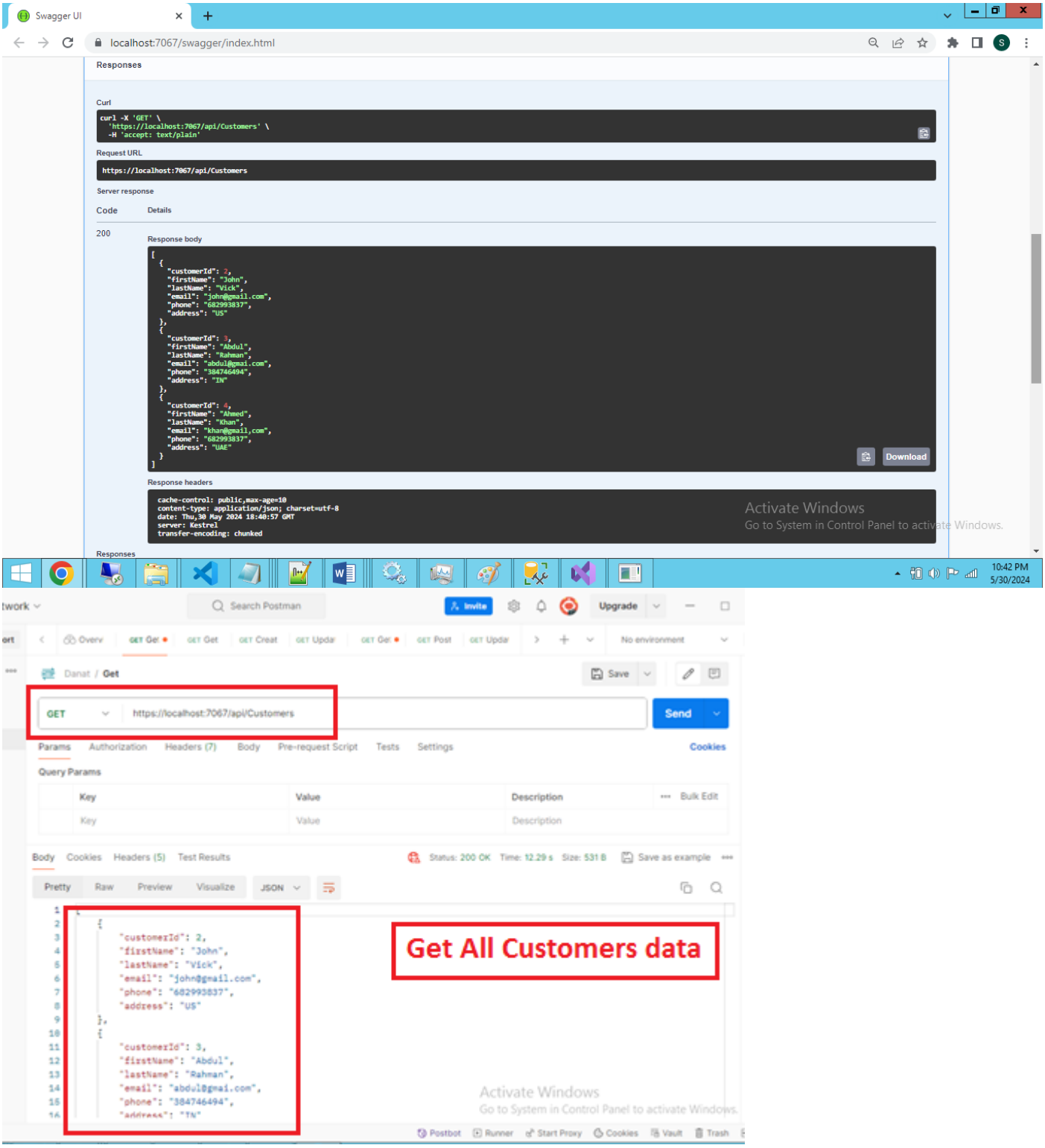
2. Get All Customers

- **Purpose**: Retrieve list of all customers
- **URL**: `GET https://localhost:7067/api/Customers`
- **Headers**:

```
Authorization: Bearer <token>
```

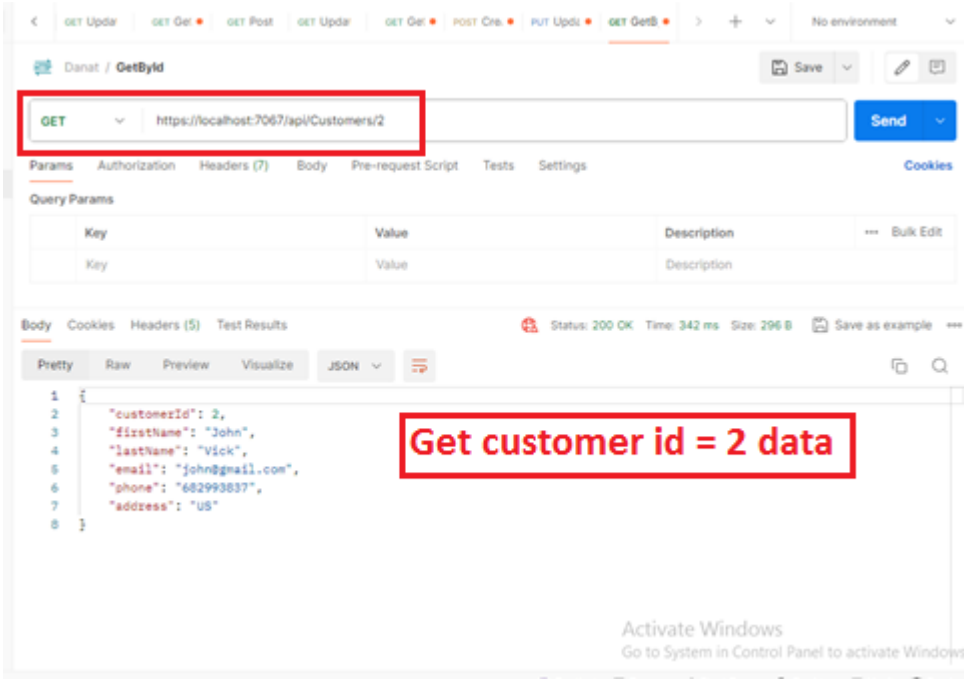
- **Response**:

```
[
  {
    "customerId": 1,
    "firstName": "John",
    "lastName": "Doe",
    "email": "john.doe@gmail.com",
    "phone": "1234567890",
    "address": "USA"
  },
  ...
]
```



🔍 3. Get Customer by ID

- **Purpose**: Retrieve a specific customer
- **URL**: `GET https://localhost:7067/api/Customers/{id}`
- **Example**: `GET /api/Customers/2`
- **Headers**: Requires JWT token

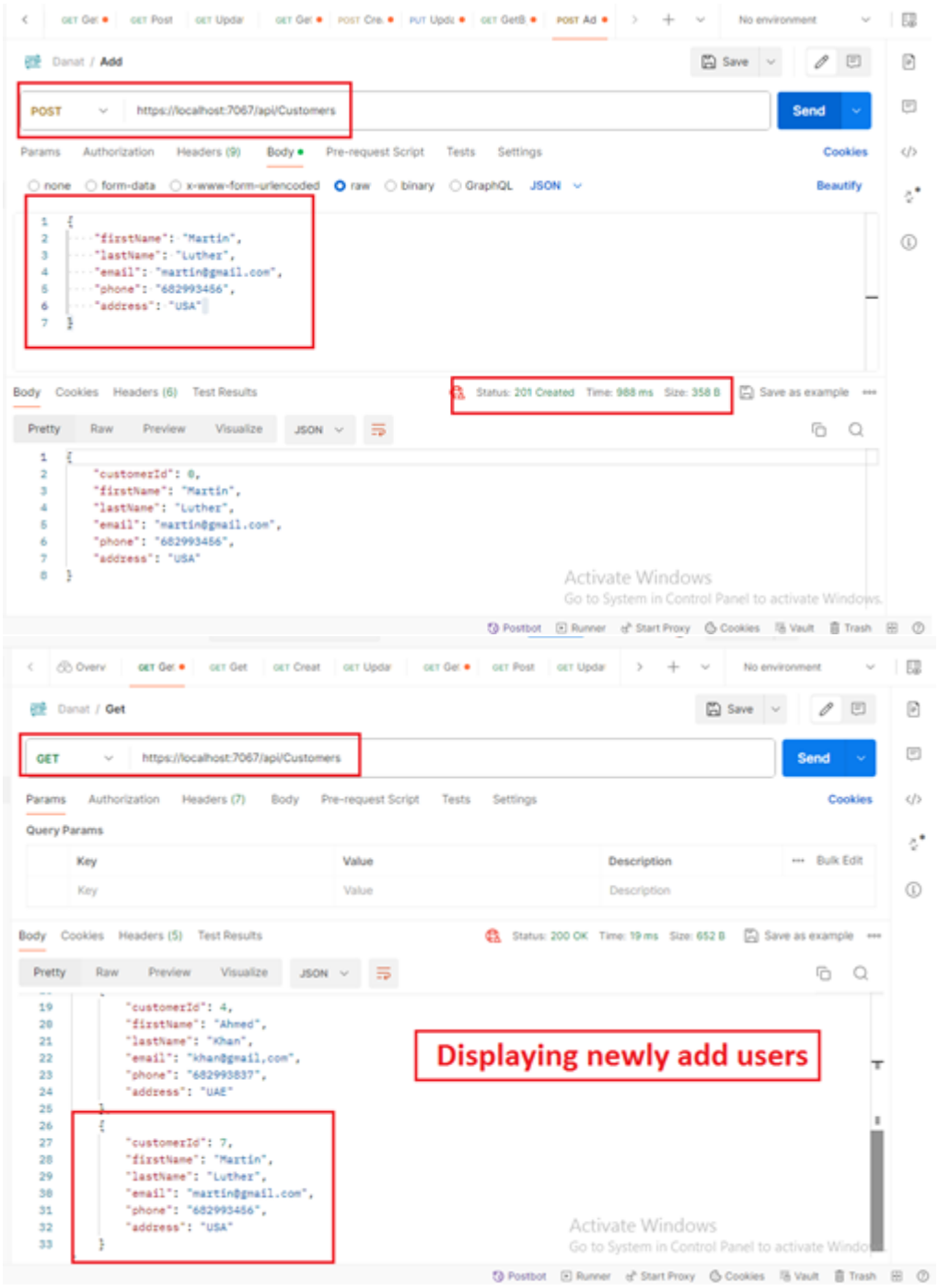


➕ 4. Add New Customer

- **Purpose**: Add a new customer
- **URL**: `POST https://localhost:7067/api/Customers`
- **Payload**:

```
{
    "firstName": "Martin",
    "lastName": "Luther",
    "email": "martin@gmail.com",
    "phone": "682993456",
    "address": "USA"
}
```

- **Validation**:
    - Required: `firstName`, `lastName`, `email`, `phone`, `address`
    - Unique: `email`

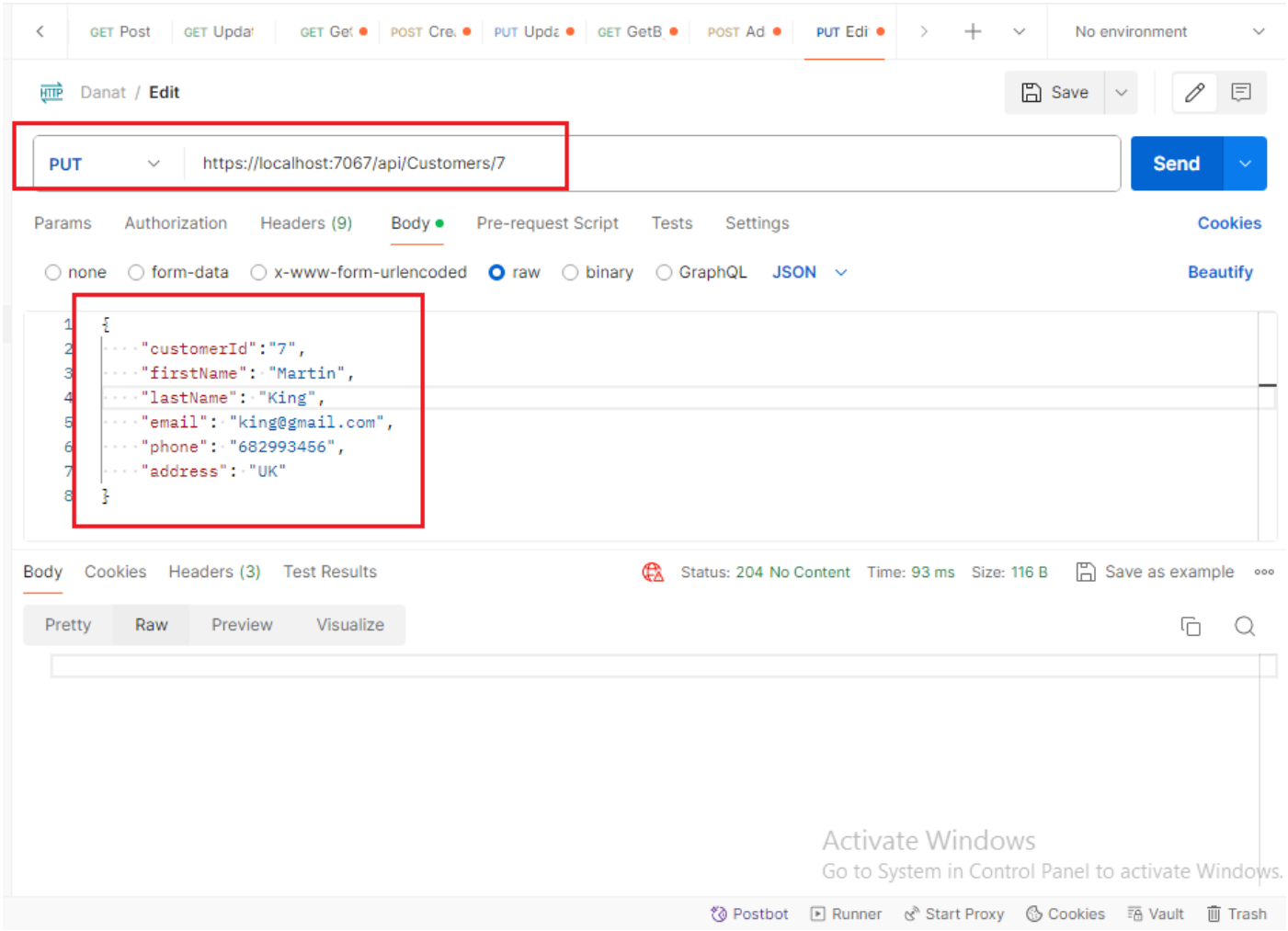


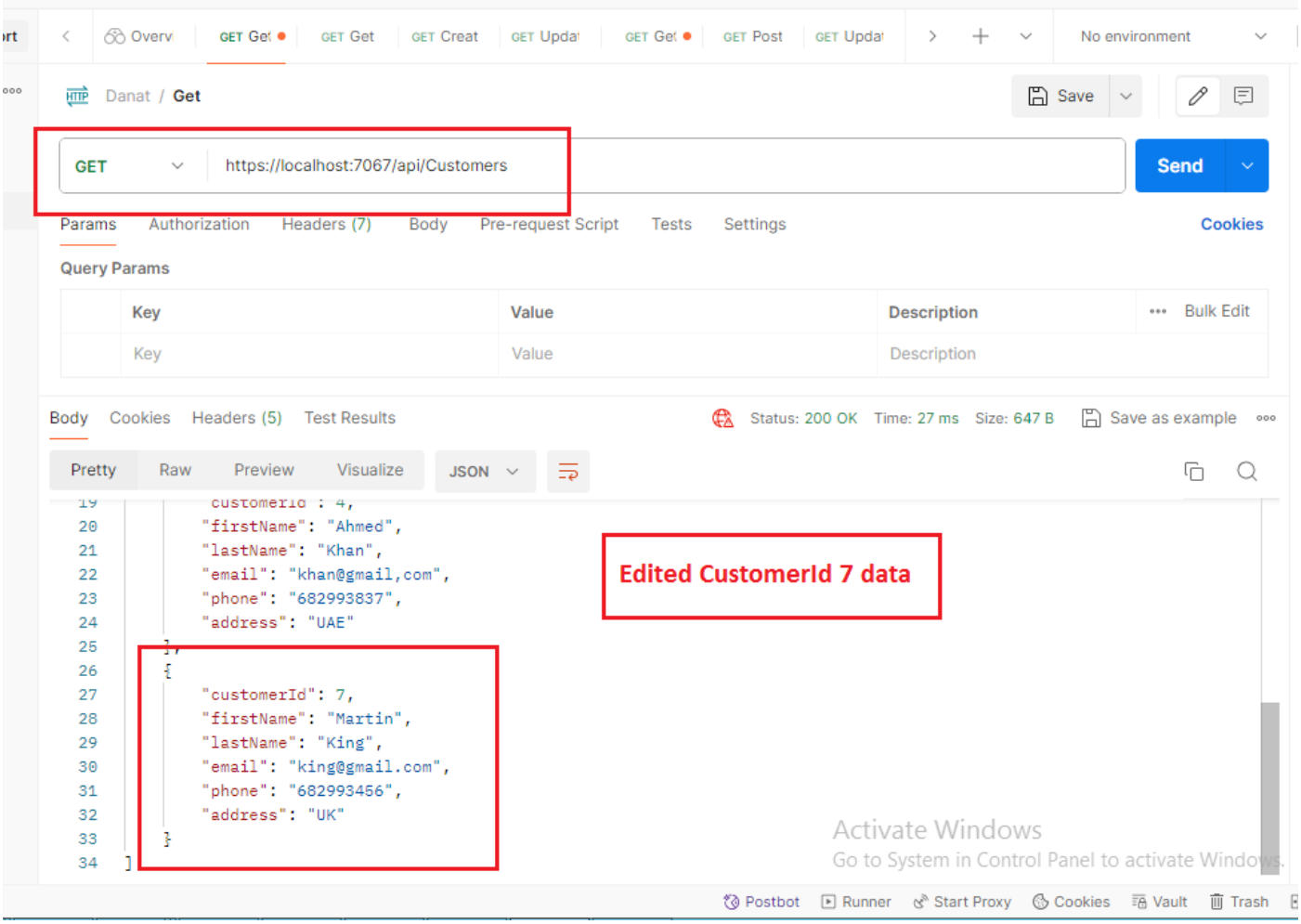
✏️ 5. Edit Existing Customer

- **Purpose**: Update customer details
- **URL**: `PUT https://localhost:7067/api/Customers/{id}`
- **Example**: `PUT /api/Customers/7`
- **Payload**:

```
{
    "customerId":"7",
    "firstName": "Martin",
    "lastName": "King",
    "email": "king@gmail.com",
    "phone": "682993456",
    "address": "UK"
}
```

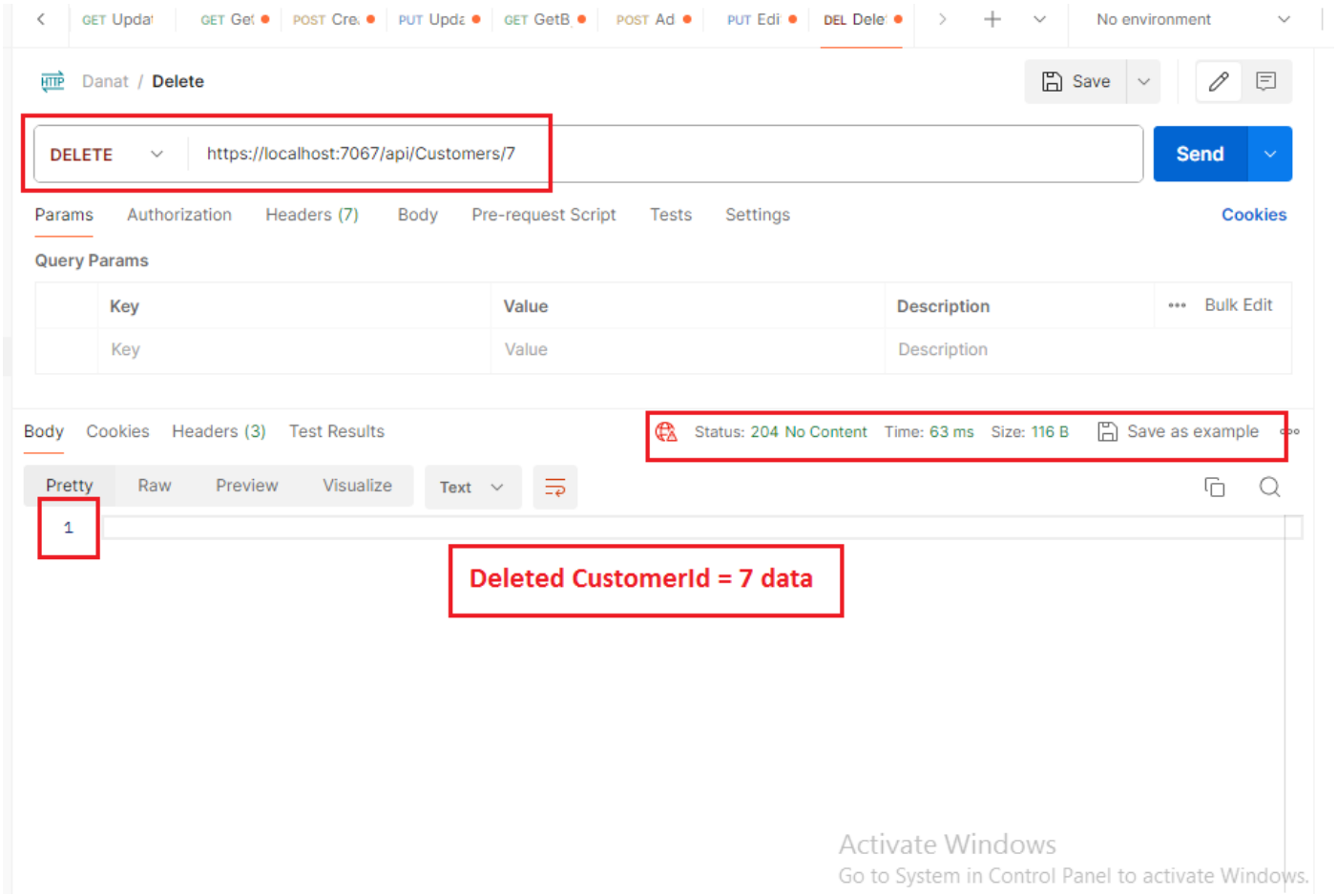
- **Note**:
  - Validates ID exists
  - Ensures email uniqueness





❌ 6. Delete Customer

- **Purpose**: Remove a customer by ID
- **URL**: `DELETE https://localhost:7067/api/Customers/{id}`
- **Example**: `DELETE /api/Customers/7`



✅ Clean Architecture Layers (Recommended)

Your architecture should be structured like:

```
/Core
  - Entities (Customer.cs)
  - Interfaces (ICustomerRepository.cs)
  - DTOs (CustomerDto.cs)
  - Validation (FluentValidation / DataAnnotations)

 /Application
  - Services (CustomerService.cs)
  - UseCases (e.g., GetAllCustomersQuery.cs)

 /Infrastructure
  - Data (AppDbContext.cs)
  - Repositories (CustomerRepository.cs)

 /API
  - Controllers (CustomerController.cs, AuthController.cs)
  - Middleware (JWT, ExceptionHandling)
  - Startup Configuration
```

🔐 Security

- Use `[Authorize]` attribute on controller or action level
- Validate JWT token in each request
- Return `401 Unauthorized` if the token is missing or invalid

⛓ Frontend ↔ Backend Connection

Frontend Route	Backend API Endpoint
`/register`	`POST /api/Customers`
`/login`	`GET /api/Auth/login`
`/customer-list`	`GET /api/Customers`
Inline Edit Customer	`PUT /api/Customers/{id}`
Delete Button Click	`DELETE /api/Customers/{id}`