

ASP.NET WEB API CORE

Lesson 03: Introduction
to .Net Core Repository
Pattern



Lesson Objectives

- In this lesson we will cover the following
 - Handling Errors
 - Repository and Unit Of Work Pattern
 - MSTest / XUnit
 - Basic Unit Test
 - Postman Utility
 - Working with swagger



Handling errors in an ASP.NET Core Web API

- API controllers may contain a lot of validation such as parameters check, model state check etc.

```
[HttpGet]
```

```
[SwaggerOperation("GetDevices")]
```

```
[ValidateActionParameters]
```

```
public IActionResult Get([FromQuery][Required]int page,  
[FromQuery][Required]int pageSize)  
{  
    if (!this.ModelState.IsValid)  
    {  
        return new BadRequestObjectResult(this.ModelState);  
    }  
    return new ObjectResult(deviceService.GetDevices(page, pageSize));  
}
```



Handling Errors in Core (Contind...)

- API controllers can be easily cleaned by creating validation model attribute.

Example below contains simple model validation check.

Basically all you need to do is to create a custom Action Filter which will check if the ModelState is valid, and if not it returns a BadRequestObjectResult containing the ModelState.

```
public class ValidateModelAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if (!context.ModelState.IsValid)
        {
            context.Result = new BadRequestObjectResult(context.ModelState);
        }
    }
}
```



Handling Errors in Core (Contd...)

Hook it up to your Controller:

```
[Authorize]
```

```
[Route("api/properties")]
```

```
[ValidateModel]
```

```
public class PropertiesController : Controller
```

```
{
```

```
    // code omitted for brevity
```

```
}
```

Now every time I make a call to one of the actions in that controller, and the model validation fails, it will automatically return 400 (Bad Request) response, and the body of the response will contain the errors, for example:

Handling Errors in Core (Contnd..)



```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

namespace DeviceManager.Api.ActionFilters
{
    /// <summary>
    /// Intriduces Model state auto validation to reduce code duplication
    /// </summary>
    /// <seealso cref="Microsoft.AspNetCore.Mvc.Filters.ActionFilterAttribute" />
    public class ValidateModelStateAttribute : ActionFilterAttribute
    {
        /// <summary>
        /// Validates Model automaticaly
        /// </summary>
        /// <param name="context"></param>
        /// <inheritdoc />
        public override void OnActionExecuting(ActionExecutingContext context)
        {
            if (!context.ModelState.IsValid)
            {
                context.Result = new BadRequestObjectResult(context.ModelState);
            }
        }
    }
}
```

Repository and Unit Of Work Pattern in .NET core

Repository Pattern

- The repository pattern is used to create an abstraction layer between the DAL (data access layer) and the BAL (business access layer) to perform CRUD operations.
- The Repository pattern is intended to create an abstraction layer between the data access layer and the business logic layer of an application.
- It is a data access pattern that prompts a more loosely coupled approach to data access.
- We create the data access logic in a separate class, or set of classes, called a repository, with the responsibility of persisting the application's business model.



Repository Pattern

Generic

Specific

Add()
Remove()
GetById(id)
Find(predicate)

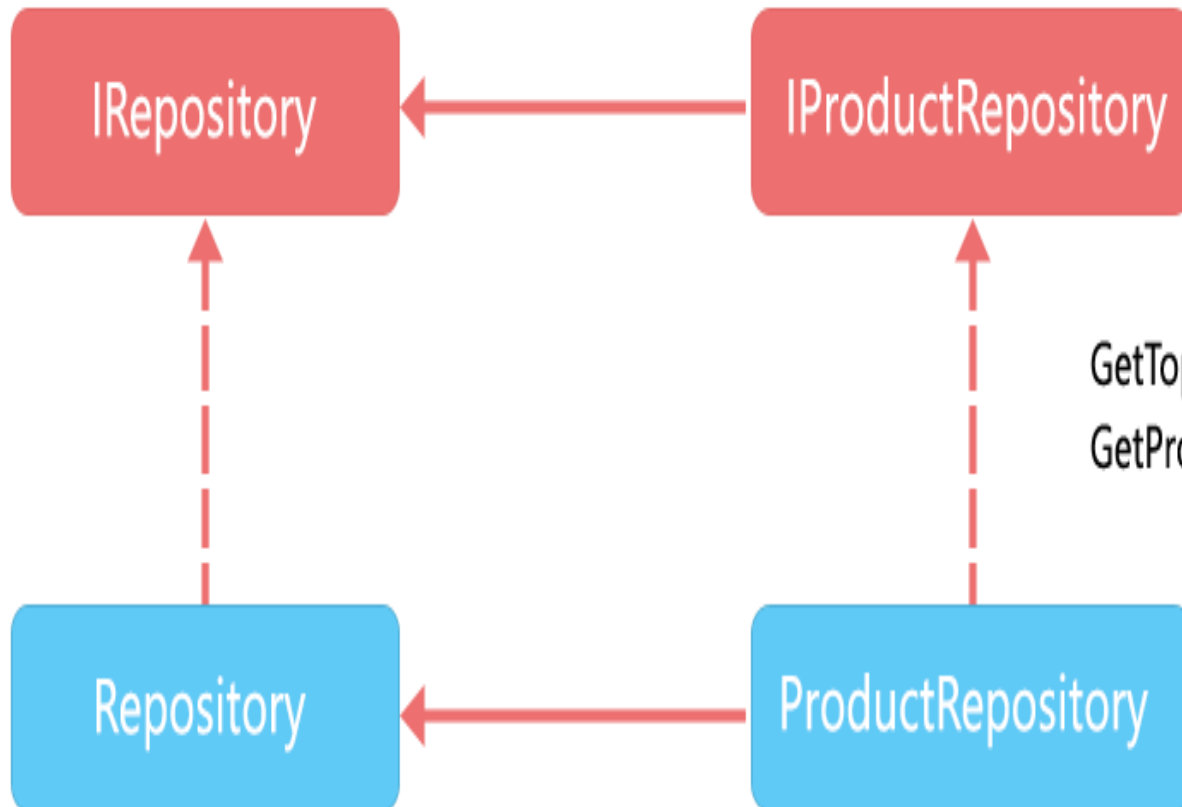
IRepository

IProductRepository

GetTopSellingProducts ()
GetProductsWithCategories ()

Repository

ProductRepository





Repository Pattern Implementation

Generic Repository Pattern

A generic repository implementation is used to define common database operations (like Create, Retrieve Update, Delete etc.) for all the database entities in a single class.

```
public interface IRepository where TEntity : class
{
    IEnumerable GetAll();
    IEnumerable Find(Expression> predicate);
    TEntity Get(object Id);

    void Add(TEntity entity);
    void AddRange(IEnumerable entities);

    void Update(TEntity entity);

    void Remove(object Id);
    void Remove(TEntity entity);
    void RemoveRange(IEnumerable entities);
}
```

Repository Pattern Implementation (Contd..)



// implementation

```
public class Repository : IRepository where TEntity : class
{
    protected readonly DbContext db;

    public Repository(DbContext _db)
    {
        db = _db;
    }

    public IEnumerable GetAll()
    {
        return db.Set().ToList();
    }

    public IEnumerable Find(Expression> predicate)
    {
        return db.Set().Where(predicate);
    }
}
```

Repository Pattern Implementation(Contd..)



```
public TEntity Get(object Id)
{
    return db.Set().Find(Id);
}
```

```
public void Add(TEntity entity)
{
    db.Set().Add(entity);
}
```

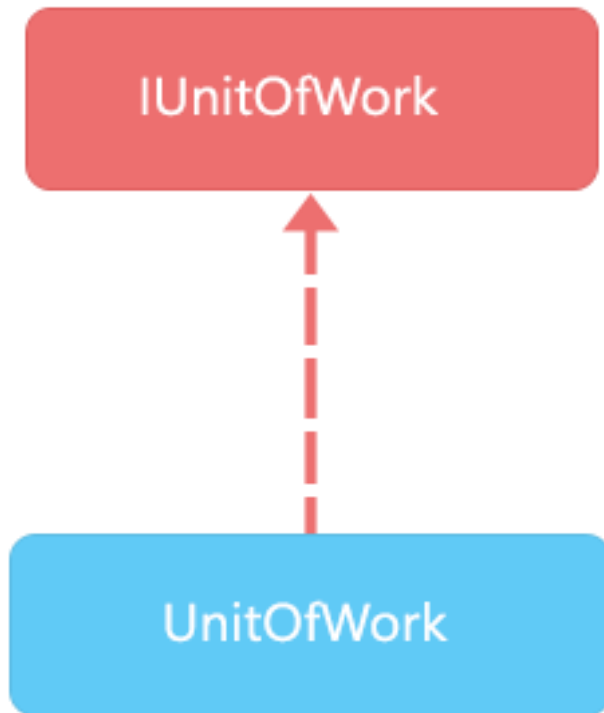
```
public void AddRange(IEnumerable entities)
{
    db.Set().AddRange(entities);
}
```

```
public void Remove(TEntity entity)
{
    db.Set().Remove(entity);
}
```



Unit of work Pattern

- The unit of pattern implementation manage in-memory database CRUD operations on entities as one transaction. So, if one of the operation is failing then entire db operations will be rollback.



```
IProductRepository Products { get ; }  
IOrderRepository Orders { get ; }  
void SaveChanges () ;
```

Entity Framework and Repository and Unit Of Work Patterns

- Entity Framework is based on the repository and unit of work patterns to perform CRUD operations on an entity.

Entity Framework

<< repository >>

DbSet

```
Add( obj )  
Remove( obj )  
Find( id )  
Where( predicate )
```

<< unit of work >>

DbContext

```
DbSet  
DbSet  
DbSet  
SaveChanges()
```

Entity Framework and Repository and Unit Of Work Patterns (Contind...)

- The DbSet class is based on the repository design pattern which provides us a set of method to perform CRUD operations on an entity.
- The DbContext class is based on unit of work pattern which includes all the DbSet entities.
- The DbContext class manages in-memory database operations on these entities and later saves all these updates as one transaction into database.





Advantages of Repository and Unit Of Work Design Patterns

- Abstract Data Access Layer and Business Access Layer from the Application.
- Manage in-memory database operations and later saves in-memory updates as one transaction into database.
- Facilitates to make the layers loosely-coupled using dependency injection.
- Facilitates to follow unit testing or test-driven development (TDD).



What is xUNIT?

- The xUnit is an open-source unit testing tool for the .NET framework that simplifies the testing process and allows us to spend more time focusing on writing our tests.
- It has an in-memory collection which we are going to fill up with our dummy data.
- We will decorate test methods with the [Fact] attribute, which is used by the xUnit framework, marking them as the actual testing methods.
- Besides the test methods, we can have any number of helper methods in the test class as well.
- When writing unit tests it is usually the practice to follow the AAA principle (Arrange, Act and Assert):
 - Arrange – this is where you would typically prepare everything for the test, in other words, prepare the scene for testing (creating the objects and setting them up as necessary)
 - Act – this is where the method we are testing is executed
 - Assert – this is the final part of the test where we compare what we expect to happen with the actual result of the test method execution



What is xUNIT?

- Test method names should be as descriptive as possible. In most of the cases, it is possible to name the method so that it is not even necessary to read the actual code to understand what is being tested.
- we use the naming convention in which the first part represents the name of the method being tested, the second part tells us more about the testing scenario and the last part is the expected result.
- Generally, the logic inside our controllers should be minimal and not so focused on business logic or infrastructure (ec. data access).
- We want to test the controller logic and not the frameworks we are using.
- We need to test how the controller behaves based on the validity of the inputs and controller responses based on the result of the operation it performs.
- The first method we are testing is the Get method and there we will want to verify Whether the method returns the `OkObjectResult` which represents 200 HTTP code response and Whether returned object contains our list and all of our items.



What is xUNIT?

- We create an instance of the Controller object in the test class and that is the class we want to test.
- It is important to note here that this constructor is called before each test method, meaning that we are always resetting the controller state and performing the test on the fresh object.
- This is important because the test methods should not be dependant on one another and we should get the same testing results, no matter how many times we run the tests and in which order we run them.
- Testing the GetById method by verifying that the controller will return 404 status code (Not Found) if someone asks for the non-existing item.
- Secondly, we test if 200 code is returned when the existing object is asked for and lastly we check if the right object is returned.
- Testing the Add Method testing that the right objects are returned when someone calls the method.
- Among other things, we are testing if ModelState is validated and the proper response is returned in the case that the model is not valid.
- But to achieve this, it is not enough to just pass the invalid object to the Add method.



What is xUnit?

- That wouldn't work anyway since model state validation is only triggered during runtime.
- It is up to integration tests to check if the model binding works properly.
- Applications are more light weightWhat we are going to do here instead is add the `ModelError` object explicitly to the `ModelState` and then assert on the response of the called method.
- Testing the remove method is pretty straightforward i.e Remove method tests take care that valid response is returned and that object is indeed removed from the list.
- Unit tests should be readable.
- No one wants to spend time trying to figure out what is that your test does. Ideally, this should be clear just by looking at the test name.
- Unit tests should be maintainable
- We should try to write our tests in a way that minor changes to the code shouldn't make us change all of our tests.



What is xUnit?

- The DRY (don't repeat yourself) principle applies here, and we should treat our test code the same as the production code.
- This lowers the possibility that one day someone gets to the point where he/she needs to comment out all of our tests because it has become too difficult to maintain them.
- Unit sets should be fast
- If tests are taking too long to execute, it is probable that people will run them less often.
- That is certainly a bad thing and no one wishes to wait too long for tests to execute.
- Unit tests should not have any dependances
- It is important that anyone who is working on the project can execute tests without the need to provide access to some external system or database.
- Tests need to run in full isolation.



What is xUnit?

- Make tests trustworthy rather than just aiming for the code coverage
- Good tests should provide us with the confidence that we will be able to detect errors before they reach production.
- It is easy to write tests that don't assert the right things just to make them pass and to increase code coverage.
- But there is no point in doing that. We should try to test the right things to be able to rely on them when time comes to make some changes to the code.



Postman

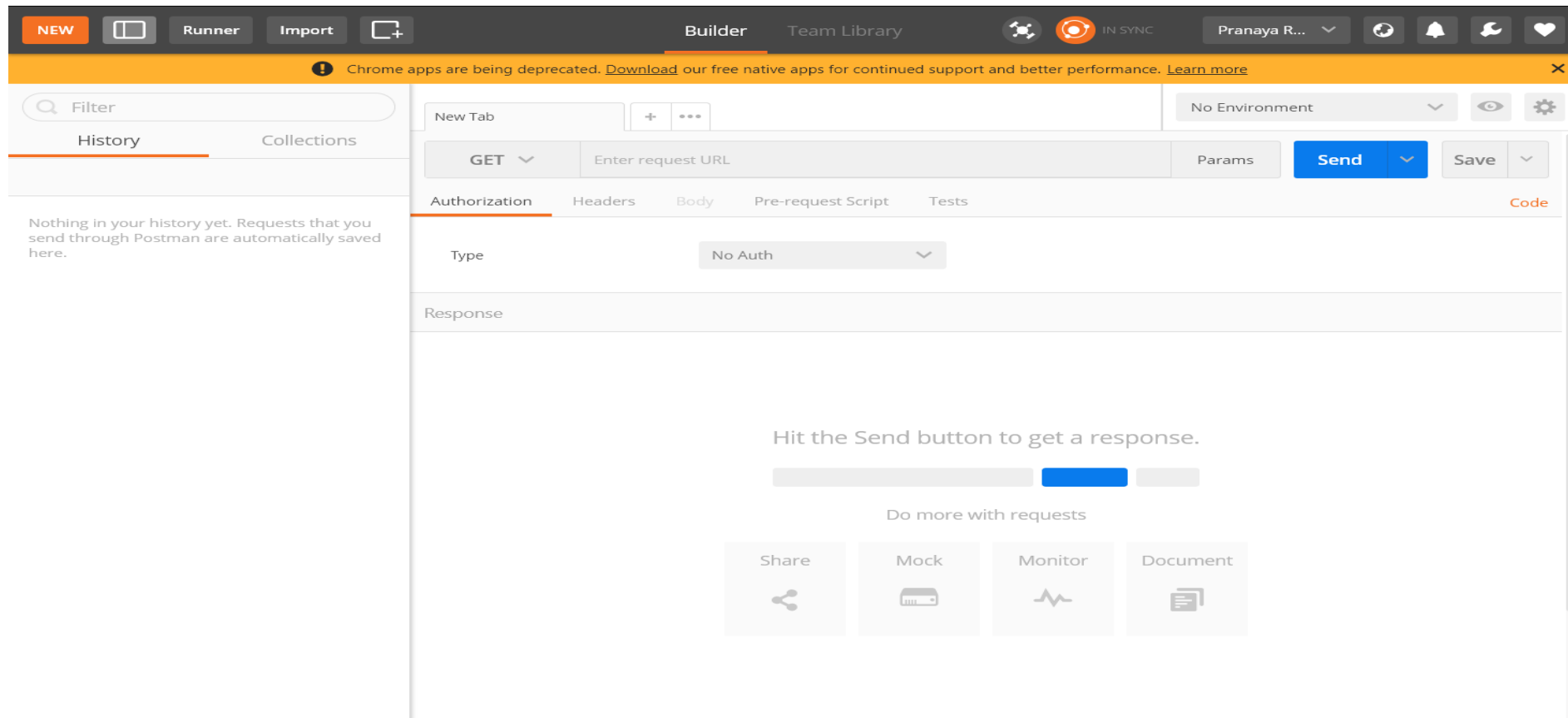
What is POSTMAN?

- The Postman is the most popular and the most powerful HTTP client for testing the restful web services. Postman makes it easy to test the Restful Web APIs, as well as it develops and documents Restful APIs by allowing the users to quickly put together both simple and complex HTTP requests. The Postman is available as both a Google Chrome in-browser app and Google Chrome Packaged App.
- The packaged app version of Postman provides many advanced features that include **OAuth 2.0** support and **bulk uploading/importing** that are not available with the Google Chrome in-browser app version.



Steps for installing the Postman

- **Step 1:** Download and install POSTMAN from [Postman](#)
- **Step 2:** Once the Postman is successfully installed, open the Postman. It will look like the image shown below.





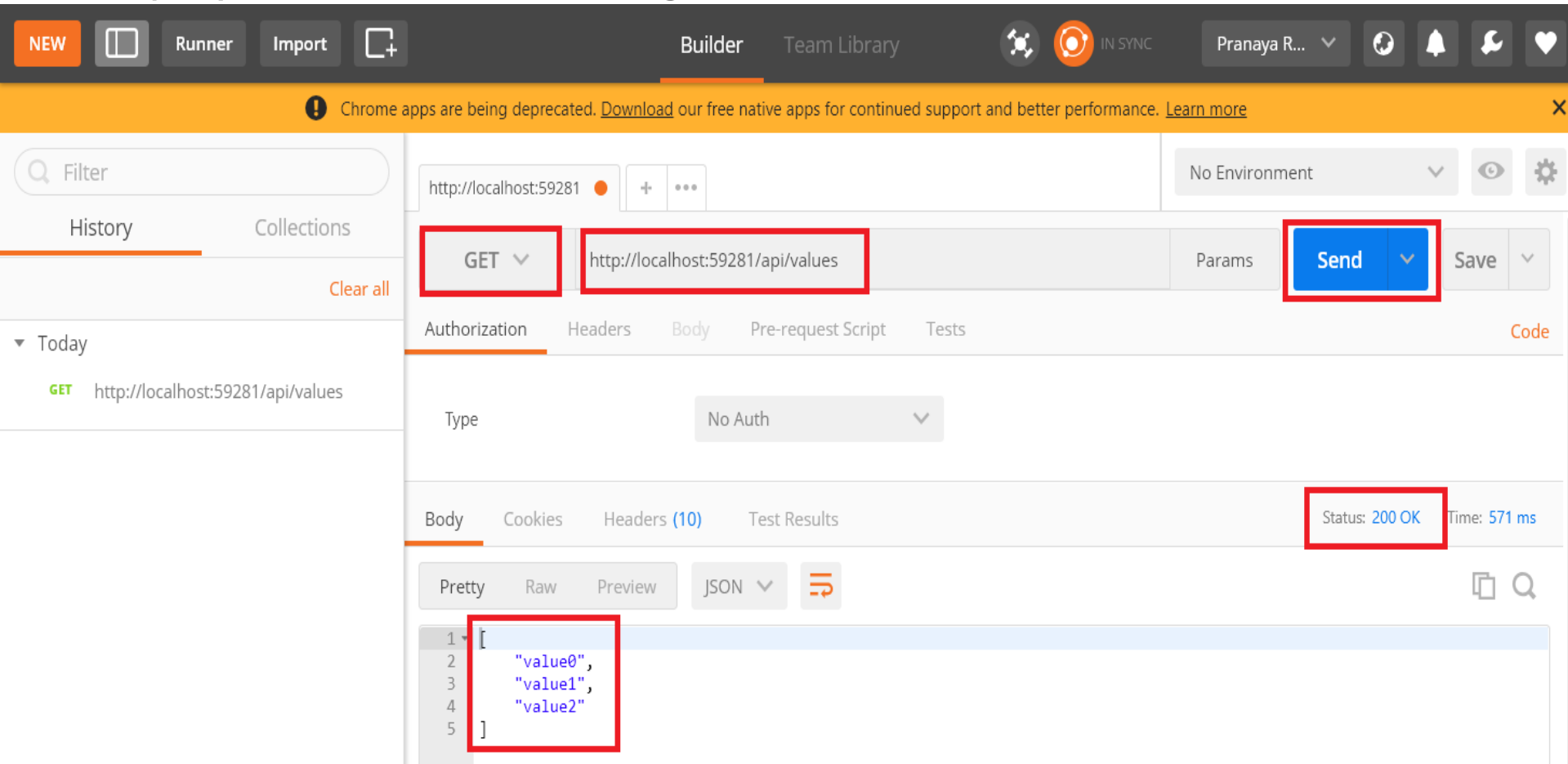
GET Request using Postman

- Select the HTTP Method as "GET" and enter the URL of your Web API as shown in the below image.

The screenshot displays the Postman application interface. At the top, there is a dark grey header bar with navigation buttons: 'NEW' (orange), a grid icon, 'Runner', 'Import', and a plus icon. To the right of these are 'Builder' and 'Team Library' tabs, followed by a 'IN SYNC' status indicator and a user profile dropdown for 'Pranaya R...'. On the far right of the header are icons for a globe, a bell, a moon, and a heart. Below the header is an orange banner with a warning icon and text: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The main interface is divided into a left sidebar and a main workspace. The sidebar has a 'Filter' search bar and two tabs: 'History' (active, underlined) and 'Collections'. Below the 'History' tab, it says 'Nothing in your history yet. Requests that you send through Postman are automatically saved here.' The main workspace shows a request configuration for 'http://localhost:59281'. The HTTP method 'GET' is selected in a dropdown and is highlighted with a red box. The URL 'http://localhost:59281/api/values' is entered in the text field and is also highlighted with a red box. To the right of the URL is a 'Params' tab. Further right is a blue 'Send' button, which is highlighted with a red box, and a 'Save' button. Below the request configuration are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests'. The 'Authorization' tab is active, showing a 'Type' dropdown set to 'No Auth'. At the bottom of the workspace is a 'Response' section.

GET Request using Postman Cotnd...

- Once you click on the Send button, an HTTP request is sent to the provided URL. The response coming from the Web API Server is displayed in the below image.



The screenshot displays the Postman application interface. At the top, there is a navigation bar with buttons for 'NEW', 'Runner', 'Import', 'Builder', and 'Team Library'. A notification banner indicates that Chrome apps are being deprecated. The main workspace shows a GET request to the URL 'http://localhost:59281/api/values'. The 'Send' button is highlighted with a red box. Below the request bar, the 'Body' tab is selected, and the response is displayed as a JSON array:

```
[
  "value0",
  "value1",
  "value2"
]
```

 The response status is '200 OK' and the time taken is '571 ms'. The response is also highlighted with a red box.



POST Request using Postman

- Choose the HTTP verb as **POST**
- Set the URL
- Set the Content-Type as **application/json**. To do this click on the Header tab and provide the key value as shown in the below image

The screenshot shows the Postman interface with a POST request configured. The URL is `http://localhost:59281/api/values`. The 'Headers' tab is selected, and a header with the key 'Content-Type' and value 'application/json' is added. The 'Body' tab is also visible.

Chrome apps are being deprecated. [Download](#) our free native apps for continued support and better performance. [Learn more](#)

Filter

History Collections

Clear all

▼ Today

GET `http://localhost:59281/api/values`

POST `http://localhost:59281/api/values`

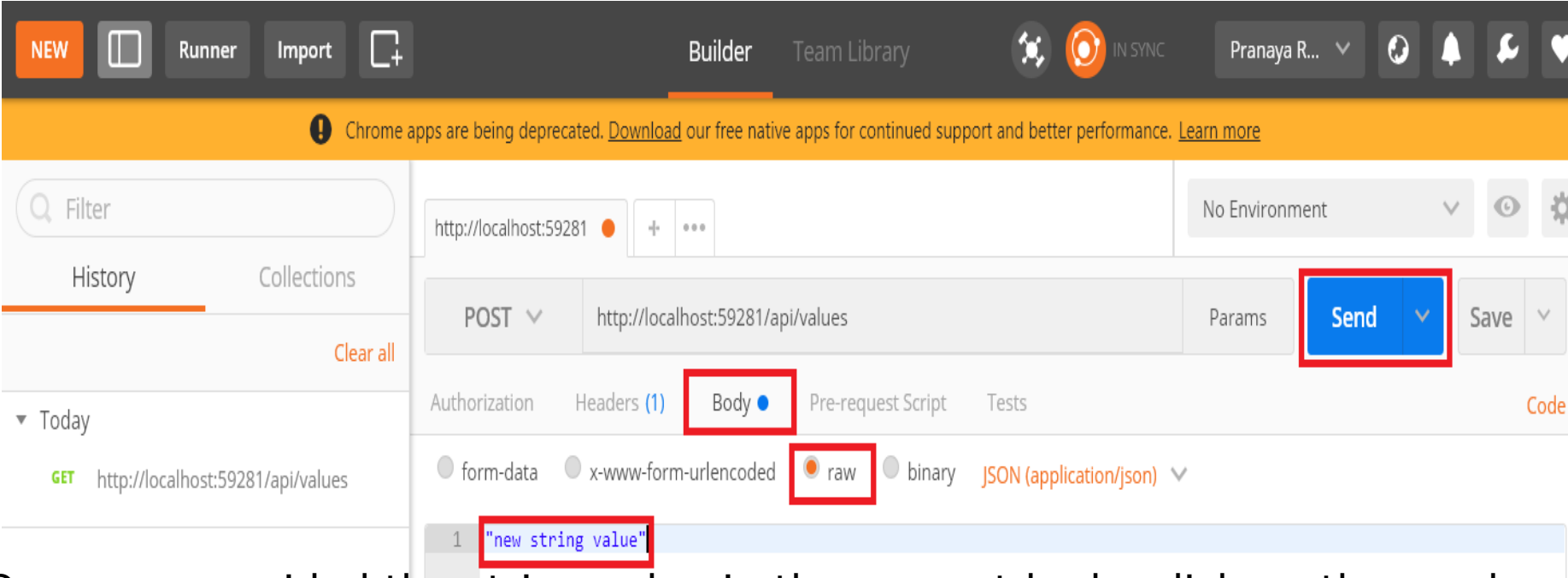
Authorization Headers (1) Body Pre-request Script Tests Code

Key	Value	Description
✓ Content-Type	application/json	
New key	Value	Description

Response

POST Request using Postman Contd..

- Next, we need to provide the string value that we want to add to the string array. We need to provide string value in the request body. To do so click on the body tab and provide the string value as shown below.



The screenshot displays the Postman application interface. At the top, there is a dark header bar with buttons for 'NEW', 'Runner', 'Import', and 'Builder'. Below this is an orange banner with a warning message: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The main interface is divided into three sections. On the left is a sidebar with a 'Filter' search bar, 'History' and 'Collections' tabs, and a list of requests including a 'GET http://localhost:59281/api/values' request. The central area shows a selected request: a 'POST' request to 'http://localhost:59281/api/values'. Below the URL bar are tabs for 'Authorization', 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests'. The 'Body' tab is selected and highlighted with a red box. Within the 'Body' tab, there are radio buttons for 'form-data', 'x-www-form-urlencoded', 'raw', and 'binary'. The 'raw' option is selected and highlighted with a red box. Below the radio buttons, the text 'JSON (application/json)' is visible. At the bottom of the 'Body' tab, there is a text input field containing the string 'new string value', which is also highlighted with a red box. On the right side of the request configuration area, there are buttons for 'Send', 'Save', and 'Code'. The 'Send' button is highlighted with a red box.

Once you provided the string value in the request body, click on the send button which will issue a post request to the web API. In the same way, you can test the PUT and DELETE Requests.



Swagger

- This is one of the best tool for testing web api method. It will also give very good user friendly definition of web api method.

Advantage

1. We can easily test web api method
2. We will get user friendly documented message about web api method
3. We can easily debug the web api method using this tool.

Integrate Swagger UI in an ASP.NET Core

- Swagger is an open-source software framework that helps developers design, build, document, and consume RESTful Web API
- Swagger is popular for its Swagger UI that allows developers to test their Web APIs. However, Swagger toolset supports automated documentation, code generation, and automated testing including test cases.

We have divided this Concepts into the following sections:

- The Need for Documenting our API
- Swagger/Open API
 - Swagger Specification
 - Swagger UI
- Integrating Swagger UI into our Applications
 - Installing the Package
 - Configuring the Swagger middleware
 - Exploring the Swagger UI



Integrate Swagger UI (Contind..)

- Extending and Customizing
 - Extending the documentation
 - Customizing the UI



Integrate Swagger UI (Contind..)

The Need for Documenting our API

- Developers who consume our API might be trying to solve important business problems with it. Hence it is very important for them to understand how to use our API effectively. This is where API documentation comes into the picture.
- The API documentation is the process of giving instructions about how to effectively use and integrate an API.
- Hence it can be thought of as a concise reference manual containing all the information required to work with the API, with details about the functions, classes, return types, arguments and more, supported by tutorials and examples.
- So having proper documentation for our API enables the consumers to integrate our APIs as quickly as possible and move forward with their development.
- Furthermore, this also helps them to understand the value and usage of our API, improve the chances for our API adoption and make our APIs easier to maintain and support.



Integrate Swagger UI (Contind..)

Swagger/Open API

- Swagger is a language-agnostic specification for describing REST APIs. The Swagger is also referred to as OpenAPI.
- It allows us to understand the capabilities of a service without looking at the actual implementation code.
- Swagger minimizes the amount of work needed while integrating an API. Similarly, it also helps API developers to document their APIs quickly and accurately.

Swagger Specification

Swagger Specification is an important part of the Swagger flow. By default, a document named swagger.json is generated by the Swagger tool which is based on our API.

It describes the capabilities of our API and how to access it via HTTP.



Integrate Swagger UI (Contind..)

Swagger UI

- Swagger UI offers a web-based UI that provides information about the service. This is built using the Swagger Specification and embedded inside the [Swashbuckle](#) package and hence it can be hosted in our ASP.NET Core app using middlewares.

Integrating Swagger UI into our Applications

- We can use the Swashbuckle package to easily integrate Swagger into our .NET Core Web API projects. It will generate the Swagger specification for our project. Additionally, the Swagger UI is also contained within Swashbuckle.

There are three main components in the Swashbuckle package:

Swashbuckle.AspNetCore.Swagger:

This contains the Swagger object model and the middleware to expose SwaggerDocument objects as JSON.



Integrate Swagger UI (Contind..)

- `Swashbuckle.AspNetCore.SwaggerGen`: A Swagger generator that builds `SwaggerDocument` objects directly from our routes, controllers, and models.
- `Swashbuckle.AspNetCore.SwaggerUI`: An embedded version of the Swagger UI tool. It interprets Swagger JSON to build a rich, customizable experience for describing the web API functionality.

Installing the Package

- The first step is to install the Swashbuckle package.
- We can execute the following command in the Package Manager Console window:
 - `Install-Package Swashbuckle.AspNetCore -version 5.0.0-rc4`
- This will install the Swashbuckle package in our application.



Integrate Swagger UI (Contind..)

Configuring the Swagger Middleware

The next step is to configure the Swagger Middleware.

Let's make the following changes in the ConfigureServices() method of the Startup.cs class:

```
public void ConfigureServices(IServiceCollection services)
{
    // Register the Swagger generator, defining 1 or more Swagger
documents
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version =
"v1" });
    });
    services.AddControllers();
}
```



Integrate Swagger UI (Contind..)

- This adds the Swagger generator to the services collection.
- In the Configure() method, let's enable the middleware for serving the generated JSON document and the Swagger UI:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // Enable middleware to serve generated Swagger as a JSON endpoint.
    app.UseSwagger();
    // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
    // specifying the Swagger JSON endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });
}
```

By executing these steps, The Swagger is configured and ready to use in our project.



Exploring the Swagger UI



First, we are going to create an Employee class:

```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string EmailId { get; set; }
}
```



```
[Route("api/[controller]")]
[ApiController]
public class EmployeeController : ControllerBase
{
    // GET: api/Employee
    [HttpGet]
    public IEnumerable<Employee> Get()
    {
        return GetEmployees();
    }
    // GET: api/Employee/5
    [HttpGet("{id}", Name = "Get")]
    public Employee Get(int id)
    {
        return GetEmployees().Find(e => e.Id == id);
    }
    // POST: api/Employee
    [HttpPost]
    [Produces("application/json")]
    public Employee Post([FromBody] Employee employee)
    {
        // Logic to create new Employee
        return new Employee();
    }
}
```



Swagger UI

➤ The Swagger UI can be found at `http://localhost: <port> /swagger:`

The screenshot displays the Swagger UI interface in a web browser. The browser's address bar shows the URL `localhost:44352/swagger/index.html`. The Swagger UI header includes the Swagger logo, the text "Supported by SMARTBEAR", and a dropdown menu labeled "Select a definition" with "My API V1" selected. Below the header, the main content area shows the API definition for "My API v1" (OAS3) at the path `/swagger/v1/swagger.json`. The "Employee" endpoint is expanded, showing four methods: GET, POST, PUT, and DELETE, each with its corresponding URL. The "Schemas" section is also expanded, showing the "Employee" model.

Swagger UI

localhost:44352/swagger/index.html

Swagger
Supported by SMARTBEAR

Select a definition: My API V1

My API v1 OAS3
`/swagger/v1/swagger.json`

Employee

- GET `/api/Employee`
- POST `/api/Employee`
- GET `/api/Employee/{id}`
- PUT `/api/Employee/{id}`
- DELETE `/api/Employee/{id}`

Schemas

- Employee >



Swagger UI (Contind...)

- Now we can explore the API via the Swagger UI and it will be easier to incorporate it into other applications.
- We can see each controller and its action methods listed here.
- Once we click on an action method, we can see detailed information like parameters, response, and example values. There is also an option to try out each of those action methods:



Swagger UI (Contind...)

A screenshot of a web browser displaying the Swagger UI for a REST API. The browser's address bar shows 'localhost:44352/swagger/index.html'. The page title is 'Employee'. The endpoint is 'GET /api/Employee'. The 'Parameters' section is empty, showing 'No parameters'. The 'Responses' section shows a '200' status code with a 'Success' description. A 'Media type' dropdown is set to 'text/plain'. Below it, there is a note 'Controls Accept header.' and a tab for 'Example Value' which displays a JSON array of employee objects. A 'Try it out' button is visible in the top right of the endpoint section.

Swagger UI

localhost:44352/swagger/index.html

Employee

GET /api/Employee

Parameters [Try it out](#)

No parameters

Responses

Code	Description	Links
200	<p>Success</p> <p>Media type</p> <p>text/plain</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>[{ "id": 0, "firstName": "string", "lastName": "string", "emailId": "string" }]</pre>	No links



Summary

- In this lesson you have learnt about:
- Handling Errors
 - Repository and Unit Of Work Pattern
 - MSTest / XUnit
 - Basic Unit Test
 - Postman Utility
 - Working with swagger

