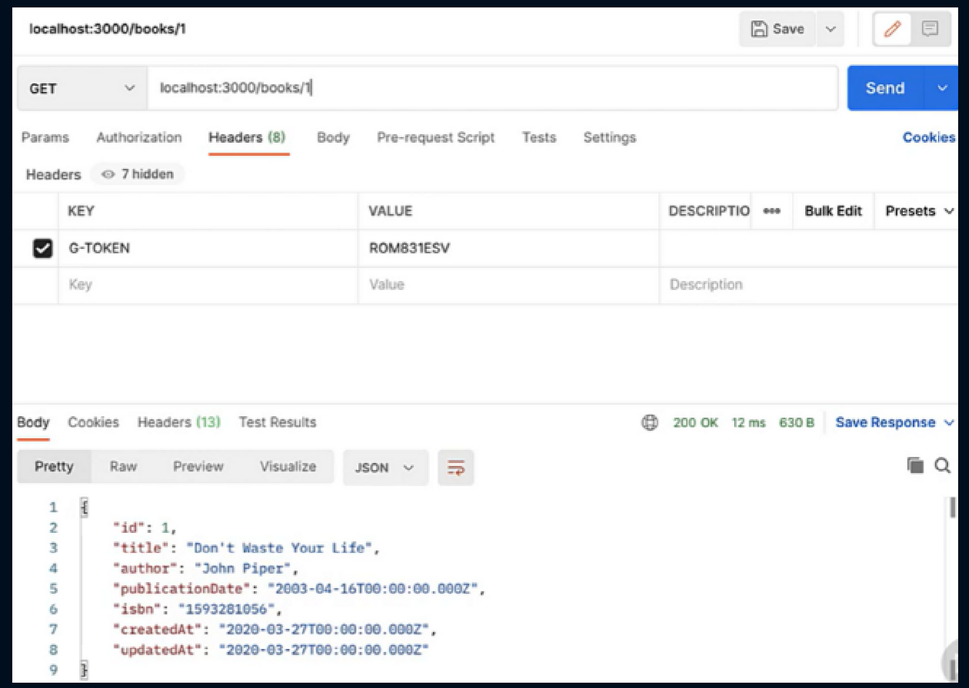
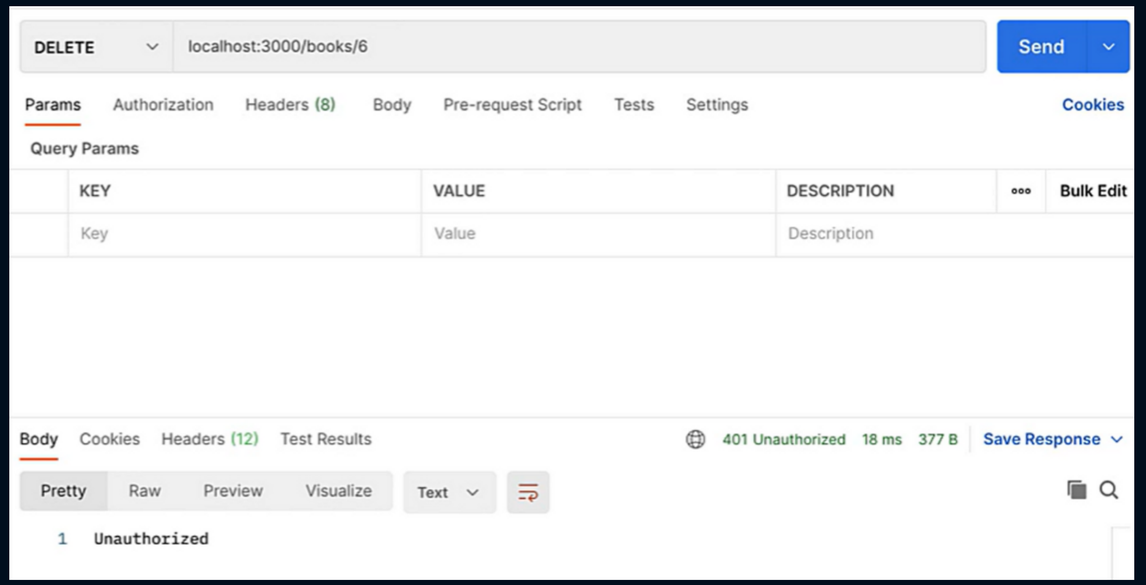
Postman overview

Postman supports various HTTP request types, including GET, POST, PUT, DELETE, PATCH, and more. You can easily switch between request types based on your API's requirements. You can add custom headers, query parameters, or URL parameters to your requests. Enables you to send different types of request bodies, such as raw text, JSON, XML, form-data, and binary data.



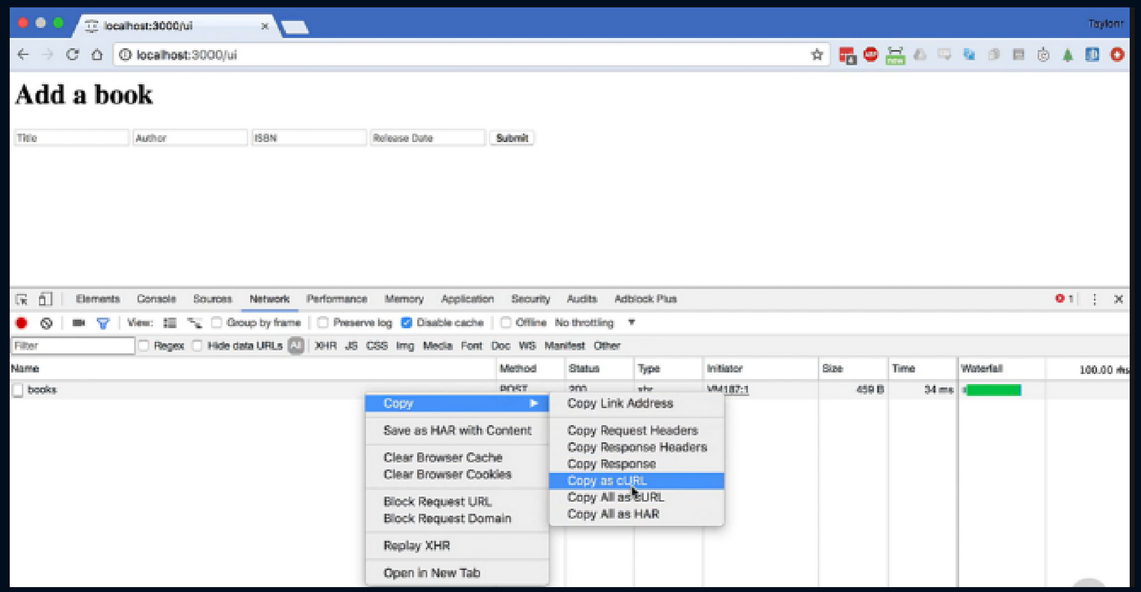
Get request



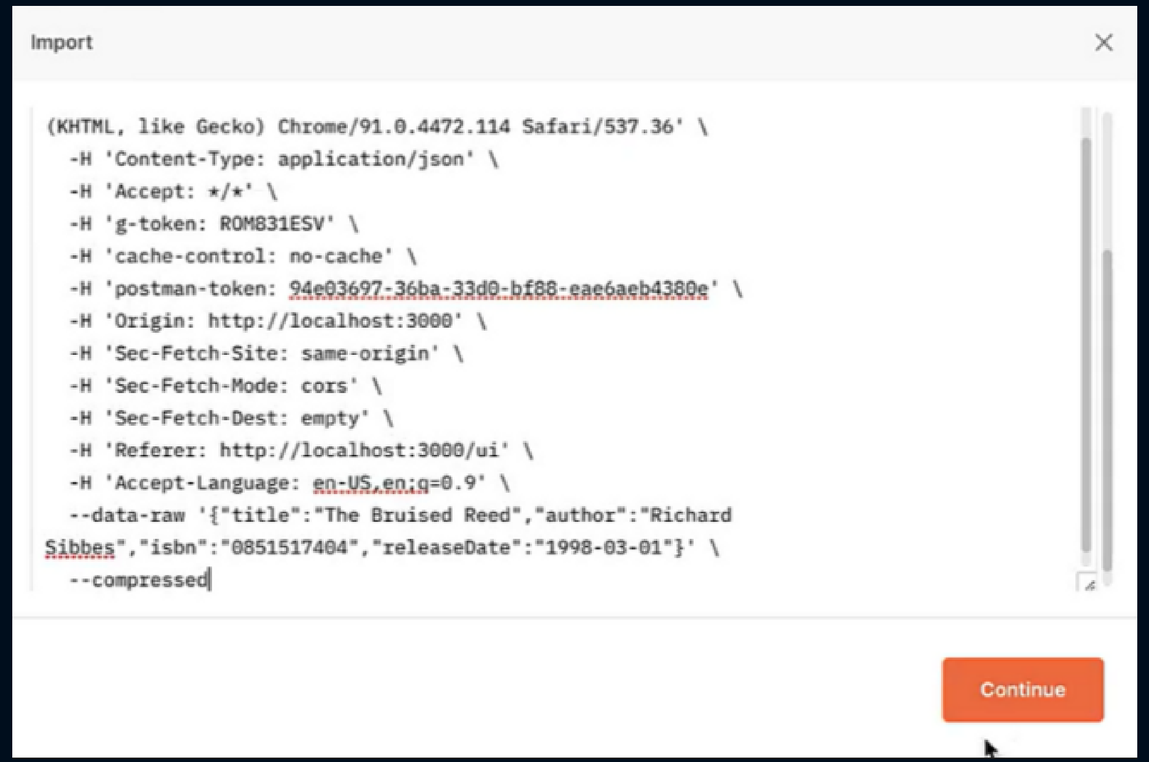
Delete request

Postman import tools

Importing Collection and Environment Files: Postman allows you to import collection files (in JSON or YAML format) and environment files (in JSON format), importing cURL Requests and importing Swagger/OpenAPI Definitions



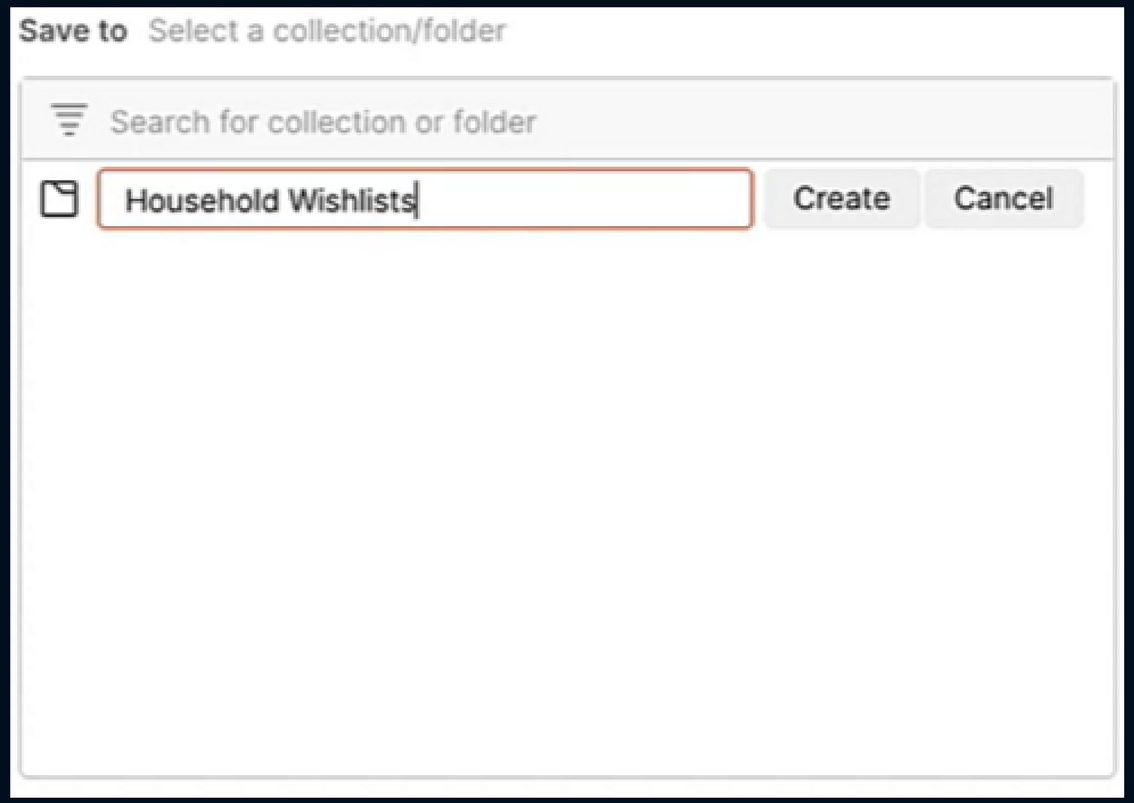
Copying a request from the browser as cURL



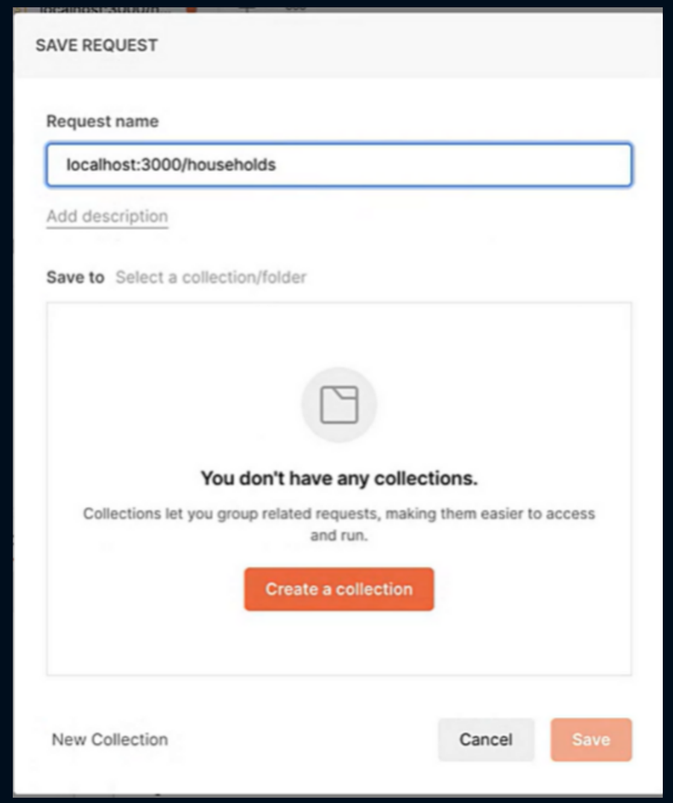
Importing cURL to Postman

Collections

You can organize your requests into collections, making it easier to manage and group related requests together. Collections allow you to organize requests hierarchically, add descriptions, and even share them with teammates.



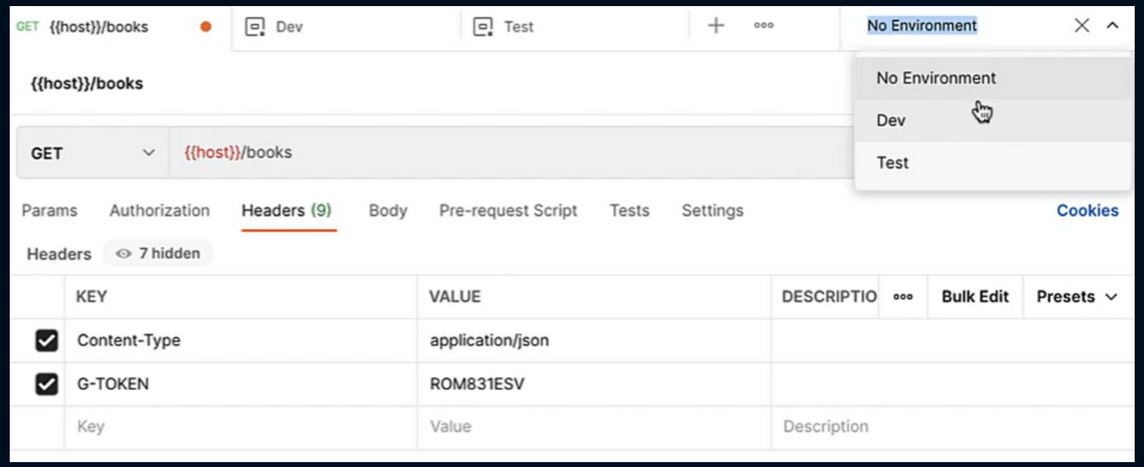
Create collection



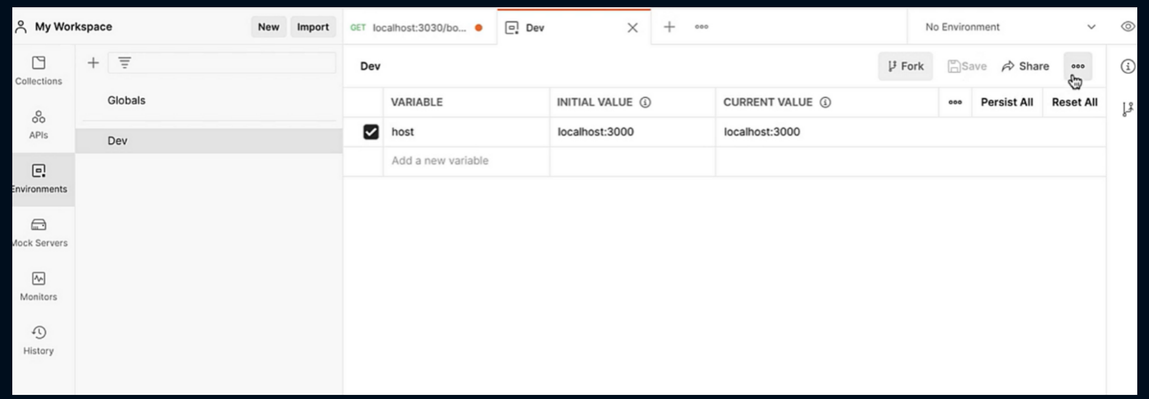
Add request to collections

Postman environment

Postman allows you to define environments and variables, making it easier to work with different environments (e.g., development, staging, production) or to store and reuse dynamic values across requests. This feature is particularly useful for managing API endpoints, authentication tokens, or other variables that may change based on the environment.



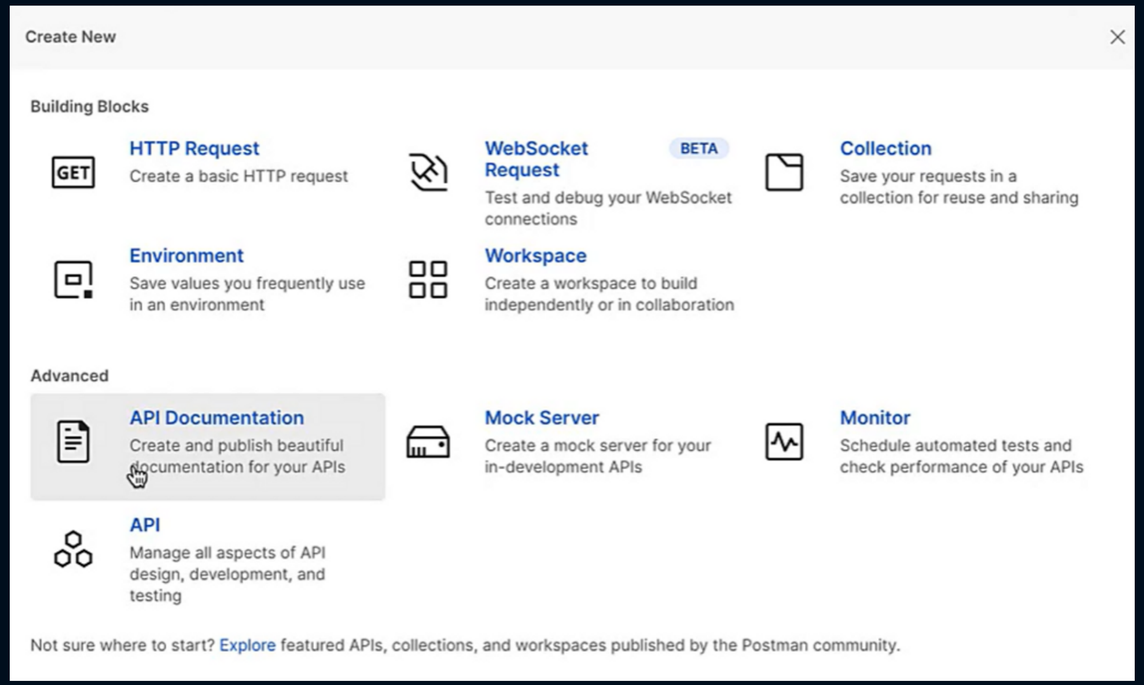
Selecting between configured environments



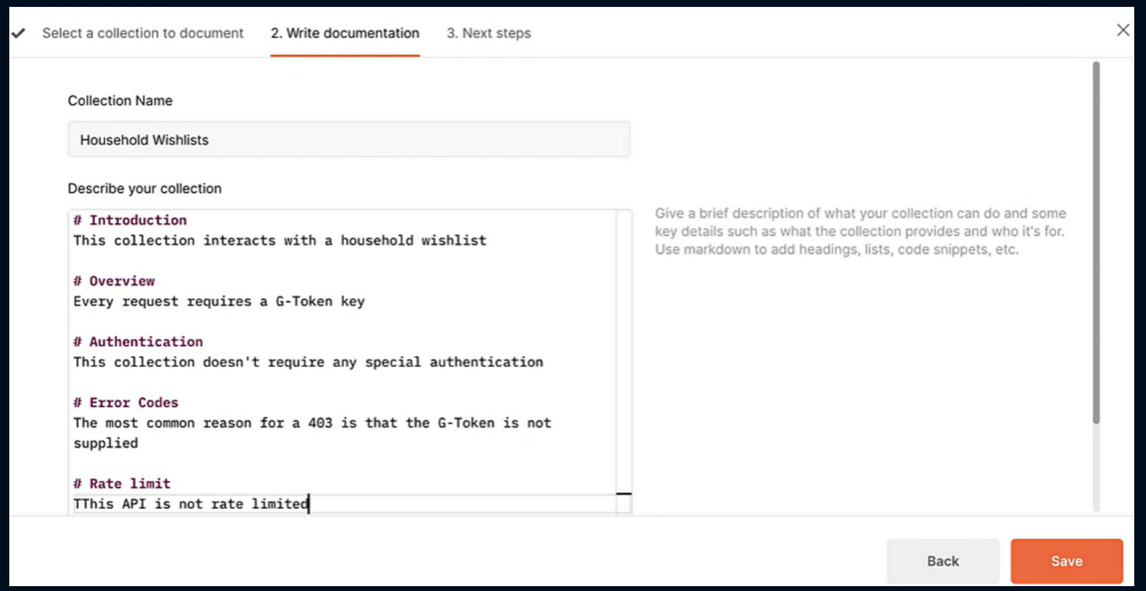
Adding variables to environments

Documentation

Postman allows you to generate API documentation based on your collections. You can add descriptions, examples, and annotations to your requests and export the documentation in various formats, such as HTML or Markdown.



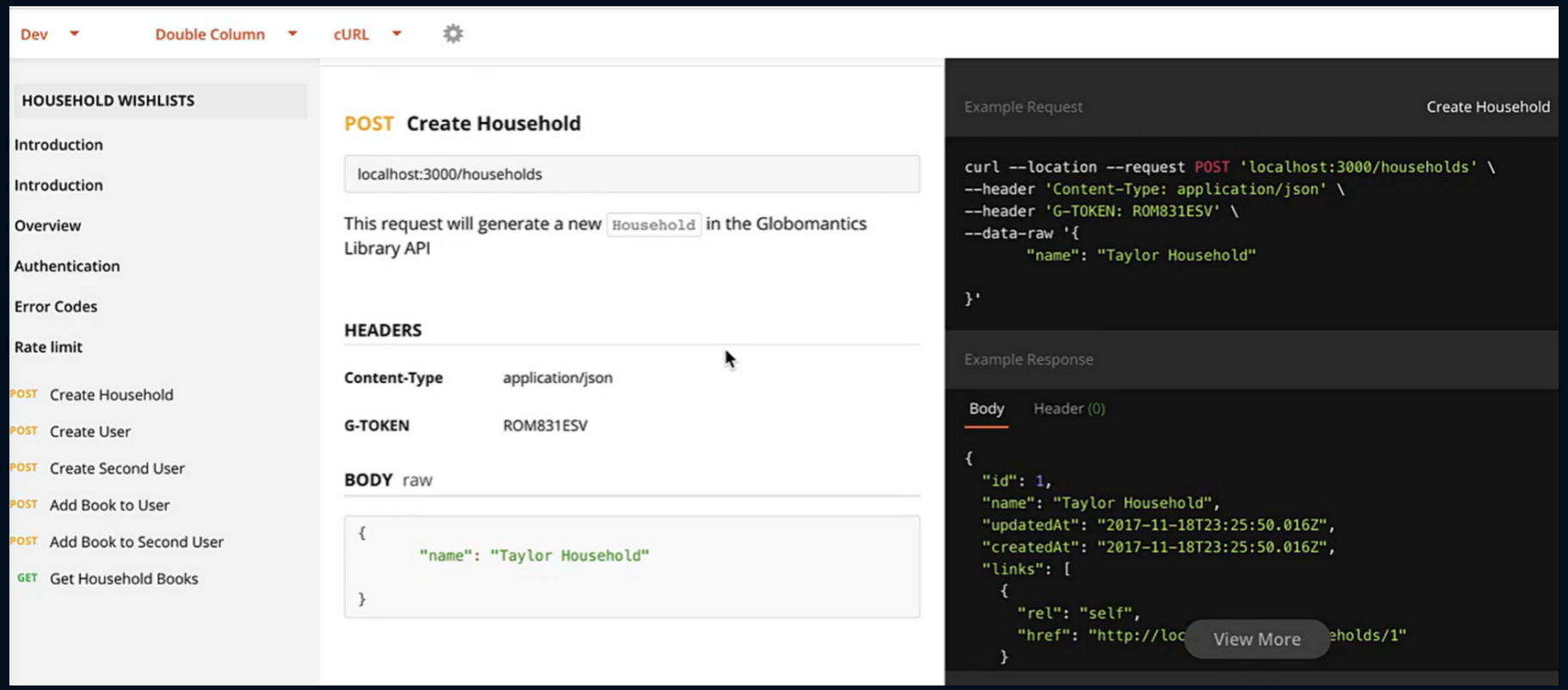
Create Documentation



Add Description to the API documentation

Documentation

Postman provides a feature called ”Publishing API Documentation” that allows you to generate and share comprehensive API documentation based on your collections



Publishing API documentation

Open API Specification Overview

OpenAPI specification describes the capabilities of your API, and how to interact with it. It's standardized, and in JSON or YAML format. Is an industry-standard specification for describing and documenting RESTful APIs. It provides a machine-readable format that allows developers to define the structure, endpoints, request/response formats, and other details of an API.

API Documentation

The OAS serves as a comprehensive documentation tool for APIs. It provides a standardized way to describe the available endpoints, operations, request/response formats, headers, etc.

Language Agnostic

The OAS is language and platform agnostic, meaning it can be used with APIs developed in various programming languages and frameworks.

Code Generation

The OAS specification can be leveraged to generate client SDKs, server stubs, and other code artifacts.

Testing and Mocking

With the OAS, you can generate mock servers that mimic the behavior of the API endpoints defined in the specification.

Visualization and Collaboration

The OAS can be visualized using various tools, making it easier to understand and navigate the API structure.

Tooling Ecosystem

Has a thriving ecosystem of tools and libraries that support its adoption. Automatic validation, documentation generation, API testing, interactive API exploration, and more.

Swagger and Swashbuckle

OpenAPI specification and Swagger specification are the same thing. Swagger is a set of open source built around that OpenAPI specification.

Swashbuckle helps with working with OpenAPI in ASP.Net Core.

Swashbuckle.AspNetCore

- Generates an OpenAPI specification from your API
- Wraps Swagger-UI and provides an embedded version of it
- https://github.com/domaindrivendev/Swashbuckle.AspNetCore

NSwag is an alternative for Swashbuckle

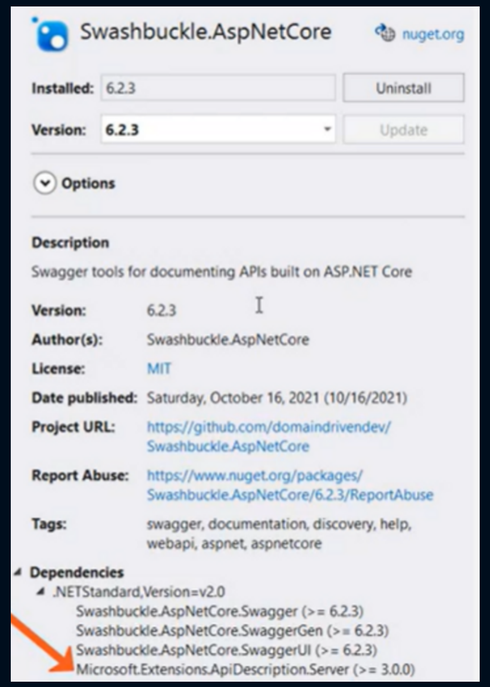
- https://github.com/RicoSuter/NSwag

Swashbuckle features

Small Open API Spec yml file

Setting up Swagger

Swashbuckle uses three main tools provided by Swagger:
Editor - helps with creating the specification
UI - renders a documentation UI from the Specification
Codegen - Consist of a set of tools that help with generating client classes , test, from the specification



Swashbuckle.AspNetCore Nuget package

```
// Learn more about configuring Swagger/OpenAPI
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
```

Swagger basic setup

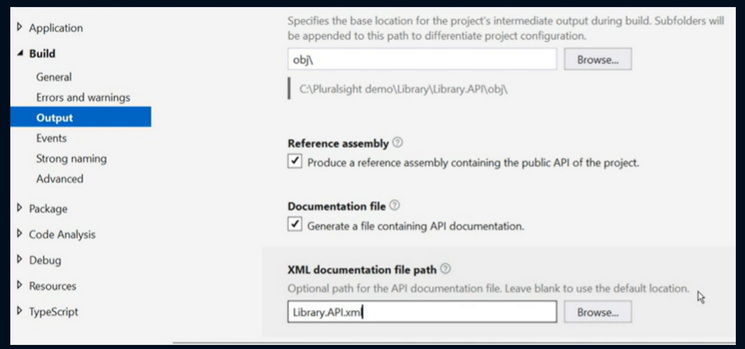
Add XML comments

XML comments are a way to document code elements such as classes, methods, properties, and parameters within your C# codebase. Swashbuckle is a popular library used in ASP.NET Web API projects to generate interactive API documentation using the OpenAPI Specification

By leveraging XML comments in your code and configuring Swashbuckle to include them, you can enhance the generated API documentation with descriptions, examples, and additional information about your API endpoints.

```
/// <summary>
/// Get an author by their id
/// </summary>
/// <param name="authorId">The id of the author you want to get</param>
/// <returns> An author with id, firstname and lastname fields</returns>
[HttpGet("{authorId}")]
0 references
public async Task<ActionResult<Author>> GetAuthor(
    Guid authorId)
```

Add XML comments



Config XML comments file

```
.Services.AddSwaggerGen(setupAction =>

setupAction.SwaggerDoc("LibraryOpenAPISpecification", new()
{
    Title = "Library API",
    Version = "1"
});

var xmlCommentsFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
var xmlCommentsFullPath = Path.Combine(
    AppContext.BaseDirectory, xmlCommentsFile);

setupAction.IncludeXmlComments(xmlCommentsFullPath);
```

Setting up XML docs in Swagger

Describing the response type and format

When using Swashbuckle, an OpenAPI/Swagger documentation tool for ASP.NET Web API projects, you can describe the response type and format of your API endpoints. This allows consumers of your API documentation to understand the structure and format of the responses they can expect from each endpoint.

```
// Add services to the container.
builder.Services.AddControllers(configure =>
{
    configure.ReturnHttpNotAcceptable = true;

    configure.Filters.Add(
       new ProducesResponseTypeAttribute(
           StatusCodes.Status400BadRequest));
    configure.Filters.Add(
        new ProducesResponseTypeAttribute(
            StatusCodes.Status406NotAcceptable));
    configure.Filters.Add(
        new ProducesResponseTypeAttribute(
            StatusCodes.Status500InternalServerError));

/// <summary>
/// Get a book by id for a specific author
/// </summary>
/// <param name="authorId">The id of the book author</param>
/// <param name="bookId">The id of the book</param>
/// <returns>An ActionResult of type Book</returns>
/// <response code="200">Returns the requested book</response>
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(B
[HttpGet("{bookId}")]
0 references
public async Task<ActionResult<Book>> GetBook(
    Guid authorId,
    Guid bookId)
```

Indicating response types

```
<PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
  <GenerateDocumentationFile>True</GenerateDocumentationFile>
  <DocumentationFile>Library.API.xml</DocumentationFile>
  <IncludeOpenAPIAnalyzers>true</IncludeOpenAPIAnalyzers>
</PropertyGroup>
```

Error List

Entire Solution | 0 Errors | 7 Warnings | 2 Messages | Build + IntelliSense

Code	Description
API1000	Action method returns undeclared status code '404'
API1000	Action method returns undeclared status code '404'
API1000	Action method returns undeclared status code '404'
API1000	Action method returns undeclared status code '422'
API1000	Action method returns undeclared status code '404'
API1000	Action method returns undeclared status code '404'

Enabling API Conventions

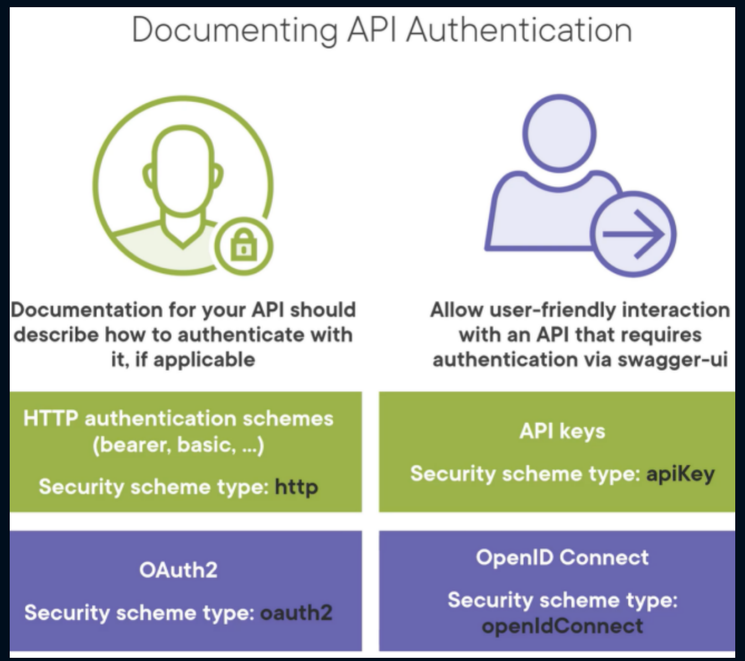
```
[HttpPost()]
[Consumes("application/json", "application/xml")]
0 references
public async Task<ActionResult<Book>> CreateBook(
    Guid authorId,
    BookForCreation bookForCreation)

}).AddNewtonsoftJson(setupAction =>
{
    setupAction.SerializerSettings.ContractResolver =
        new CamelCasePropertyNamesContractResolver();
}).AddXmlDataContractSerializerFormatters();
```

Indicating Formats

Security

You can incorporate security definitions and requirements into your API documentation. This helps communicate the authentication and authorization mechanisms necessary for consuming your API.



Documenting Authentication

```
// configure basic authentication
builder.Services.AddAuthentication("Basic")
    .AddScheme<AuthenticationSchemeOptions,
        BasicAuthenticationHandler>("Basic", null);
// Add services to the container.
builder.Services.AddControllers(configure =>
{
    configure.ReturnHttpNotAcceptable = true;

    configure.Filters.Add(
       new ProducesResponseTypeAttribute(
           StatusCodes.Status400BadRequest));
    configure.Filters.Add(
        new ProducesResponseTypeAttribute(
            StatusCodes.Status406NotAcceptable));
    configure.Filters.Add(
        new ProducesResponseTypeAttribute(
            StatusCodes.Status500InternalServerError));
    configure.Filters.Add(new AuthorizeFilter());

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();
```

Setting up Authentication and Authorization

```
setupAction.AddSecurityDefinition("CityInfoApiBearerAuth", new OpenApiSecurityScheme()
{
    Type = SecuritySchemeType.Http,
    Scheme = "Bearer",
    Description = "Input a valid token to access this API"
});

setupAction.AddSecurityRequirement(new OpenApiSecurityRequirement
{
    {
        new OpenApiSecurityScheme
        {
            Reference = new OpenApiReference {
                Type = ReferenceType.SecurityScheme,
                Id = "CityInfoApiBearerAuth" }
        }, new List<string>() }
});
```

Setting up Authorization