Patterns and Practices of Effective API Development in ASP.NET Core

David Berry

@DavidCBerry73

https://github.com/DavidCBerry13/FoodTruckNationApi


Danger:
Exampleware

“Think of this talk as a discussion rather than a hard and fast set of rules”

Goals

- An API that is easier for users to consume
- A codebase that is easy the maintain
- A pragmatic implementation that meets our business goals

What We Will Discuss

Why Create a Web API

Accessible

- Callable from any platform
- No code to be distributed

Composable

- APIs can be used by multiple client applications
- Easy to mash up data from multiple data sources

Securable

- HTTP traffic can traverse firewalls
- Well known patterns for controlling access

When is a Web API the Wrong Choice

Large Datasets

- Cost to serialize/deserialize data
- Problematic in joining large datasets

Single Consumer Data

- Is it worthwhile to add another layer?
- Are the latency costs worth it?

REST Practices

Resource URLs

`GET /api/foodtrucks`

Gets a list of all of the food trucks

`GET /api/foodtrucks/10`

Gets the food truck with the id number of 10

`GET /api/foodtrucks?tag=tacos`

Gets a collection of food trucks that have the tag “tacos”

`GET /api/foodtrucks/10/Reviews`

Gets a collection of the reviews for food truck with the id number of 10

Resource URL Practices

Favor plural nouns for resource names

Avoid URLs more than two layers deep

Use query parameters to your advantage

HTTP Verbs

	GET	POST	PUT	DELETE
`/api/foodtrucks`	Gets all food trucks	Create a food truck	Error	Error
`/api/foodtrucks/10`	Get food truck 10	Error	Update food truck 10	Update food truck 10

HTTP Status Codes

200 OK	201 CREATED	304 Not Modified
400 Bad Request	401 Unauthorized	403 Forbidden
404 Not Found	409 Conflict	500 Internal Server Error

HTTP Status Codes

Response Code	Name	Notes
200	OK	Most frequently used status code
201	Created	Used when a resource is crated with a PUT request. The URI of the new resource should be included in a Location header field
304	Not Modified	Used for GET requests when the resource has not changed and caching is enabled
400	Bad Request	The request contained invalid data. The client can fix the errors in the request and try again
401	Unauthorized	Used when the user has not authenticated
403	Forbidden	Used when the user has authenticated but does not have permission to perform the requested action
404	Not Found	The specified resource could not be found
409	Conflict	The resource you are trying to add or update is in conflict with another resource. Resolve the conflict and try again
500	Internal Server Error	Something went wrong on the server. Usually a coding error in the API itself

HTTP Status Code 404

`/api/foodtrucks/123`

- Makes sense to return HTTP 404 when food truck 123 does not exist
- Everyone understands and expects this behavior

`/api/foodtrucks or /api/foodtrucks?tags=Tacos`

- I personally prefer to return a status of HTTP 200 and an empty collection in these cases
- Think it is a little easier to develop against
- Avoids confusion of no results found vs incorrect URL

HATEOS

```
{
    "foodTruckId": 1,
    "name": "American Burger",
    "description": "Classic American Burgers, Hot Dogs and Fries",
    "website": "http://foodtrucknation/BurgerTruck",
    "tags": [
        "Burgers",
        "Hot Dogs“
     ],
    "meta": {
        "self": "http://localhost:8000/api/FoodTrucks/1",
        "reviews": http://localhost:8000/api/FoodTrucks/1/Reviews",
        "schedules": "http://localhost:8000/api/FoodTrucks/1/Schedules"
    }
}
```

API Design Rules

Design APIs from the consumer point of view

Think about how many individual API calls it takes to complete a business operation

Try to keep each API focused on one logical business domain

Be consistent. Within your API. With established practices

Be informative when something goes wrong

Code Organization

Some Good Coding Principles

Keep separate concerns separate

Aim for a balanced distribution of responsibilities

Don’t repeat yourself

Code should easily reveal its intentions about how it works

Code Organization Techniques

Use View Models

Return IActionResult objects

Delegate logic out of controllers and into other classes

Make use of filters for cross cutting concerns

Dependency injection is your friend

Enterprise Ready Code

Validate Everything

Logging matters

Handle exceptions and return good error messages

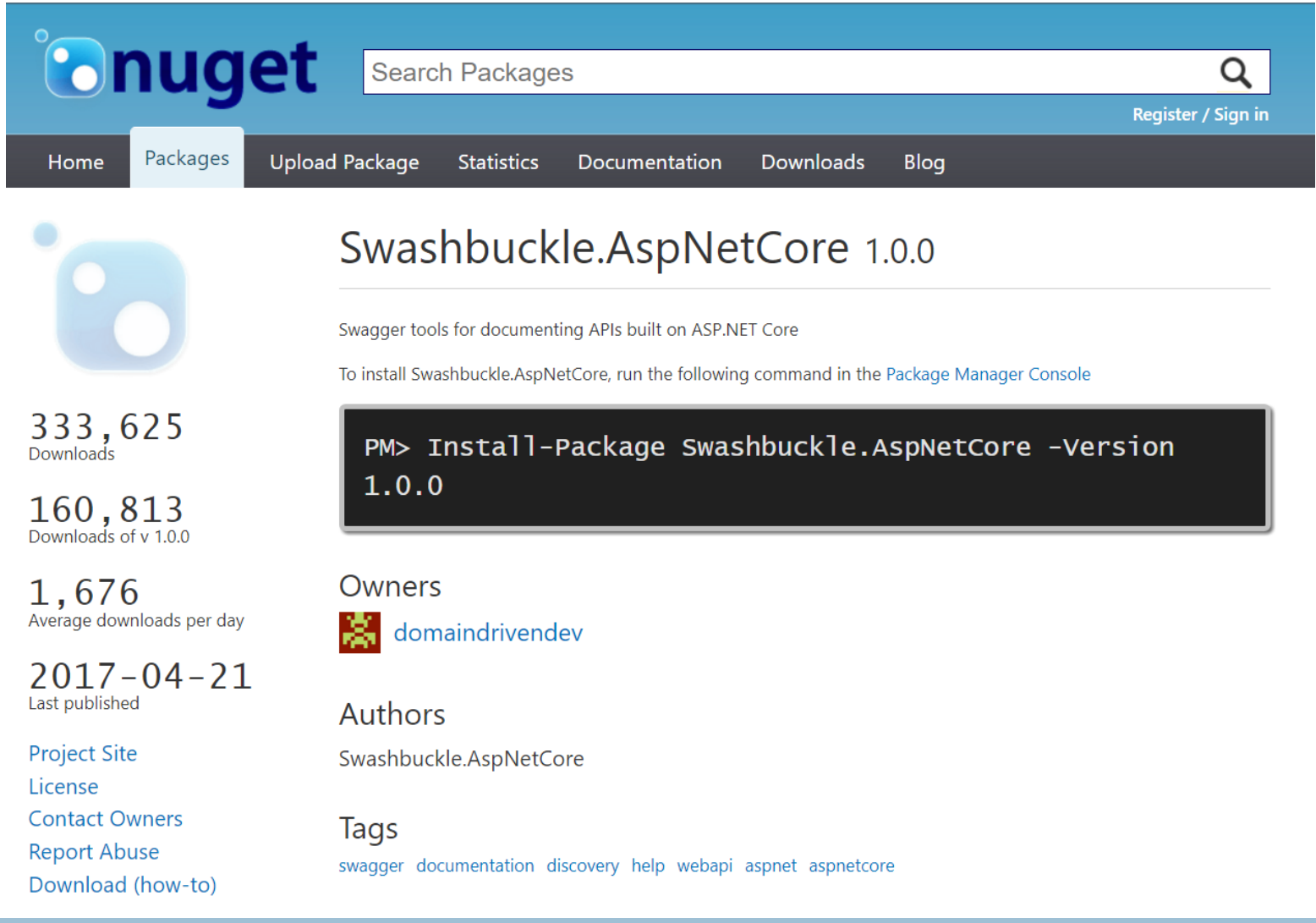
Versioning

Versioning Configuration

```
public void ConfigureServices(IServiceCollection services)
{
    …
    services.AddApiVersioning(cfg =>
    {
        cfg.DefaultApiVersion = new
            Microsoft.AspNetCore.Mvc.ApiVersion(1, 0);
        cfg.AssumeDefaultVersionWhenUnspecified = true;
        cfg.ReportApiVersions = true;
        cfg.ApiVersionReader = new HeaderApiVersionReader("api-version");
    });
    …
}
```

Documenting Your API

Swashbuckle



Swashbuckle Configuration

```
/// <summary>
/// Helper method to setup Swagger – Called from ConfigureServices()
/// </summary>
private void ConfigureServicesSwagger(IServiceCollection services)
{
    var pathToXmlComments = Configuration["Swagger:FileName"];
    services.AddSwaggerGen(options =>
    {
        options.SwaggerDoc("v1",
        new Info
        {
            Title = "Food Truck API",
            Description = “Demonstration api built around tracking food trucks",
            TermsOfService = "For demonstration only"
        });

        var filePath =
            Path.Combine(PlatformServices.Default.Application.ApplicationBasePath,
                pathToXmlComments);
        options.IncludeXmlComments(filePath);
        options.DescribeAllEnumsAsStrings();
    });
}
```

Swashbuckle Configuration

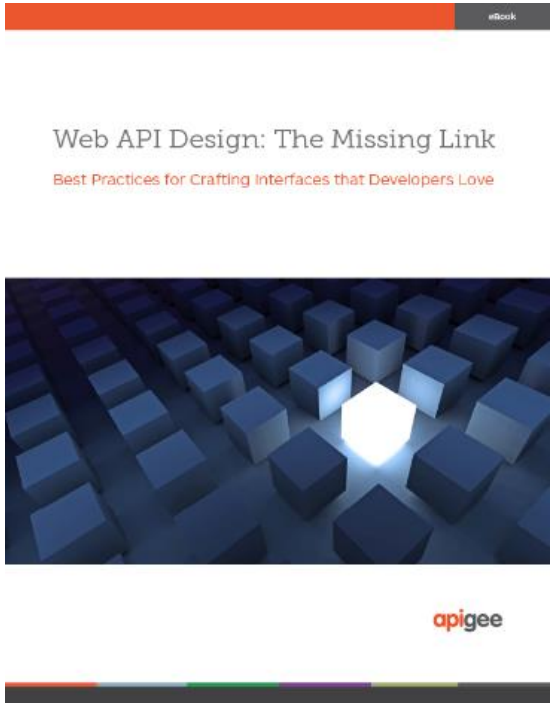
```
// In Configure (Startup.cs)

    app.UseSwagger(c =>
    {
        c.PreSerializeFilters.Add((swagger, httpReq) =>
            swagger.Host = httpReq.Host.Value);
    });

    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Docs");
    });
```

Resources

REST Design Resources



https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

https://apigee.com/about/blog/developer/web-api-design-missing-link

ASP.NET Core APIs

Implementing and Securing an API with ASP.NET Core

By Shawn Wildermuth

https://www.pluralsight.com/courses/aspdotnetcore-implementing-securing-api