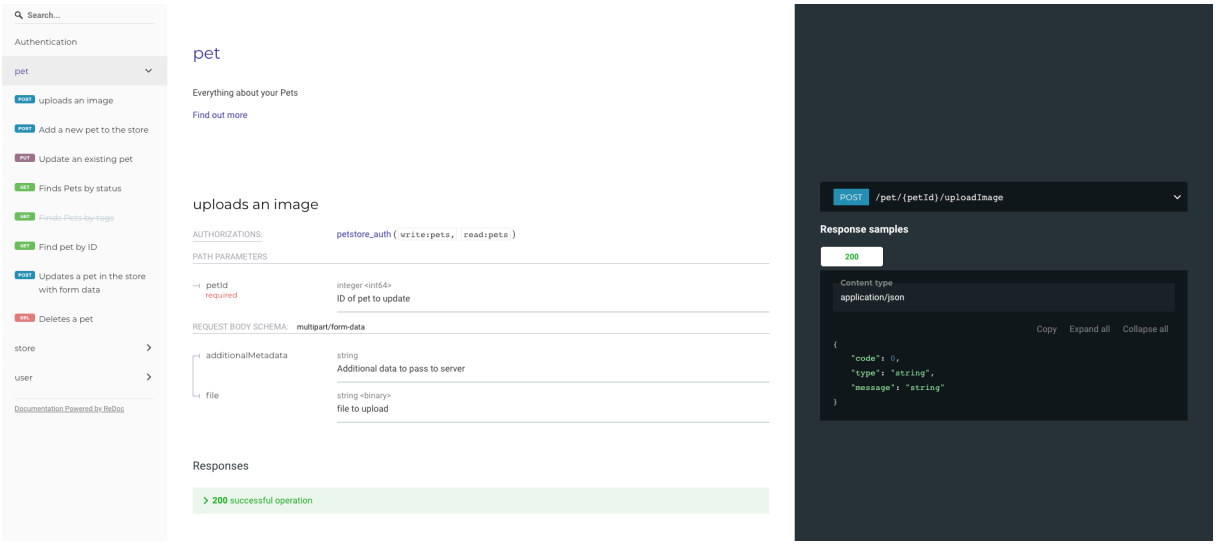
OpenAPI Developer Portals on a Shoestring

Andrew Owen

Preface



This guide enables you to create a fully featured dev portal for your Swagger or OpenAPI 3.0 content without spending a dime.

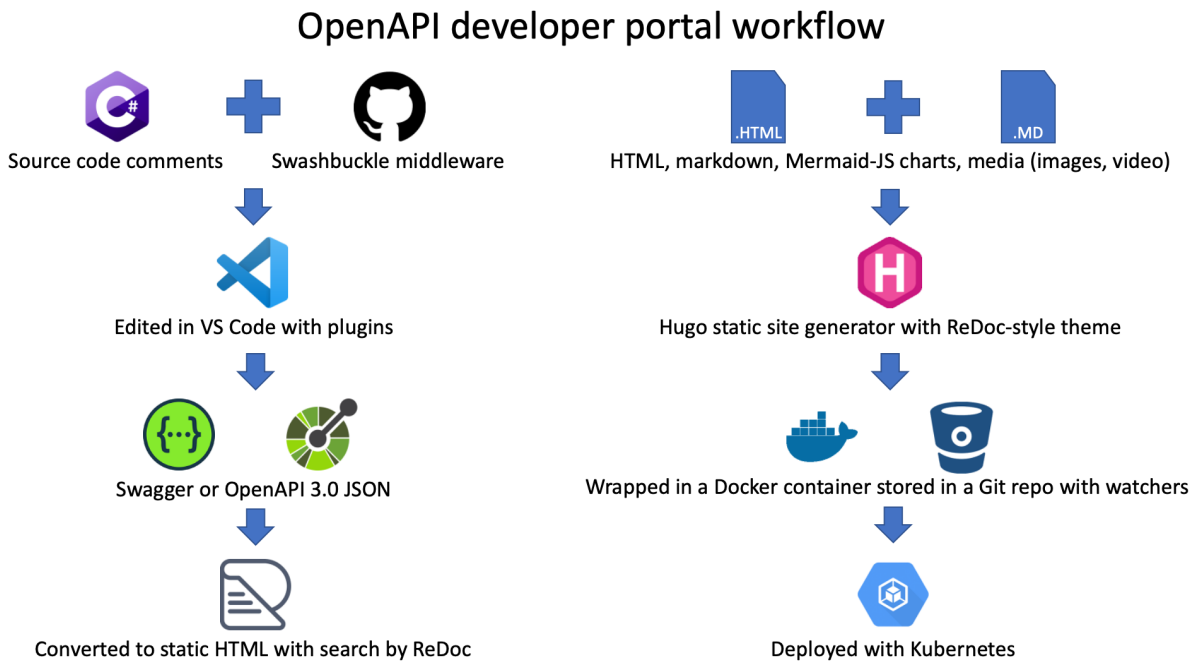
It was produced in *DocBook 5* (`https://docbook.org/`) using *XMLmind XML Editor Personal Edition* (`https://www.xmlmind.com/xmleditor/download.shtml`) and *AsciiDoc fopub* (`https://github.com/asciidoctor/asciidoctor-fopub`).

Part I. Backend

A dev portal is just a website that presents API docs to developers. It can be as simple as a static web page or as complex as you want to make it. In this guide, we'll make use of the following back-end technologies:

- Static Site Generators
- Source control
- Containerization
- Automation

In addition you will inevitably spend a fair amount of time using a browser. For better or worse, *Chrome* is the new defacto standard.



Static Site Generators

Hugo

Hugo is a type of web server known as a static site generator. It is lightweight and fast and can serve dynamic and static content. You can test content locally before deploying it. For OpenAPI content we will use the command line version of ReDoc to convert Swagger JSON files to a static HTML page that will be served by Hugo.

Install CLI tools (macOS)

1. *Homebrew* is a package manager. From the *Terminal*, enter `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
2. *Git* provides source control. From the *Terminal*, enter `git` and follow the instructions to install *Xcode*.
3. *Hugo* is a static site generator. From the *Terminal*, enter `brew install hugo`
4. *NPM* is a package manager for *JavaScript*. From the *Terminal*, enter `brew install nodejs`
5. *ReDoc* renders an OpenAPI file into a static HTML page. From the *Terminal*, enter `npm install -g redoc-cli@0.9.8`

   Ensure you are using version 0.9.8 or later: `redoc-cli --version`
6. *Swagger2PDF* converts an OpenAPI JSON file into a static PDF document. From the *Terminal*, enter `npm install swagger-spec-to-pdf`

Install CLI tools (Windows)

Requirements

- Powershell 3 (or later): `https://www.microsoft.com/en-us/download/details.aspx?id=34595`

> Powershell 5 is already installed in Windows 10.

- .NET framework 4.5 (or later): `https://www.microsoft.com/net/download`

Instructions

1. Download and install *Scoop*: `https://scoop.sh`
2. *Git* provides source control. From *PowerShell*, enter `scoop install git`
3. *Hugo* is a static site generator. From *PowerShell*, enter `scoop install hugo`
4. *NPM* is a package manager for *JavaScript*. From *PowerShell*, enter `scoop install nodejs`
5. *ReDoc* renders an OpenAPI file into a static HTML page. From *PowerShell*, enter `npm install -g redoc-cli@0.9.8`

   Ensure you are using version 0.9.8 or later: `redoc-cli --version`
6. *Swagger2PDF* converts an OpenAPI JSON file into a static PDF document. From the *PowerShell*, enter `npm install swagger-spec-to-pdf`
7. *cURL* is a command line data transfer tool. From *PowerShell*, enter `scoop install curl`

Test a local copy of the dev portal

Navigate to the folder where you have cloned the Git repository.

1. From the command line, enter `hugo server`.
2. Copy the URL from the console output and paste it into the *Chrome* browser.

Add Google Analytics to a static HTML page

Paste the following text after the `<head>` tag:

```
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async src="https://www.googletagmanager.com/gtag/js?
id=<yourID>"></script>
```

Source control

Git

If you are generating comments from code, you will be working directly in the software repository. Typically this will be a Git repository. You should familiarize yourself with the processes for creating branches, creating pull requests, resolving conflicts and merging changes. There are a number of commercial hosting options including Github, GitLab and Bitbucket. Here we'll look at Bitbucket.

Get the URL from Bitbucket

1. In *Chrome*, navigate to the required repository.
2. In the left icon menu, click **Clone**.
3. Copy the **URL**.

Clone the repository and create a local branch

1. Open *PowerShell* / *Terminal* and navigate to where your local repository folder. Example: `~/development/`
2. Enter `git clone` and paste the path you copied from *Chrome*. This creates a local copy of the repository.
3. In *Visual Studio Code*, open the repository's folder.
4. In the lower left corner of the window, click **Branch**.
5. Enter a name in the box (`feature/apidocs- <api class name>`) and select the branch to base it on. Example: `development`.

You can now make your changes.

Publish changes

1. In *Visual Studio Code*, in the left icon menu, click **Source Control**.
2. Enter a short description of why you made the changes in the box.
3. Click the **Commit** (tick) icon. If prompted to configure *Visual Studio Code* to automatically stage changes, you should do so.

4. In the lower left corner of the window, click **Publish Changes**.

Create a pull request

1. In *Chrome* navigate to the Bitbucket dashboard.
2. From the **Repositories** menu, select the repository you are working with.
3. From the left icon menu, click **Create pull request**.
4. Select your local repository from the **Source** list.
5. Confirm the **Destination** repository (typically, **development**) and click **Continue**.
6. Enter a **Description**. This should contain any additional explanation for the change.
7. Attach a copy of the *ReDoc* `HTML` file.
8. Select **Reviewers**. Typically, this is pre-populated. You can start typing a name to find a user. Ensure the product owner is included.
9. Click **Create**.

You will receive email notifications when the status of the pull request changes. Example: When the change is approved.

Resolve conflicts

1. In your local branch: `git pull origin master`
2. Merge incoming code changes while retaining doc changes.
3. `git commit -am "<your commit message>"`
4. `git push`

Merge changes

In *Chrome*, navigate to the pull request and click **Merge**.

Delete branch

1. After the change is successfully merged, from the left icon menu, click **Branches**.
2. Locate the branch you were working in and from the **Actions** menu, select **Delete branch**.
3. Click **Delete** to confirm your action.

Edit UI text in Bitbucket

While not directly related to APIs, you may on occasion have need to modify UI text contained in a file in a Git repository stored in Bitbucket. You must have a Bitbucket account to edit the files.

1. Navigate to the file in Bitbucket.
2. Click **Edit**.
3. Make your changes.
4. Click **Commit**.
5. (Optional) Enter a title in the **Commit message** box.
6. Select the **Create a pull request for this change** check box.
7. Click **Commit**.
8. (Optional) Enter a **Branch name**. Example: `feature/uitext`
9. Click **Create pull request**.
10. Enter a **Description**.
11. **Click Create**.

When your pull request has been reviewed and there has been at least one successful build, providing there are no merge conflicts, you can merge your change.

1. Navigate to the pull requests.
2. Locate your pull request and click its hyperlink.
3. Click Merge.

If the merge was successful, you should now delete your branch.

1. Navigate to the branches.
2. Click your branch's **Actions** button and select **Delete branch**, then click **Delete**.

Issue tracking

Track documentation tasks in Jira

- Standalone documentation tasks should have the issue type **Documentation**.
- Developer tasks that require documentation should have the label **Documentation**.
- Documentation should be part of the *definition of done*.

Containerization

Install Docker

You should deploy your dev portal in a Docker container. You can try it locally before you build your automation toolchain with *Docker Desktop*: `https://www.docker.com/products/docker-desktop`.

After installation, you can run Docker images locally. Example:

```
docker run -d --name rabbitmq -p 15672:15672 -p 5672:5672 bitnami/
rabbitmq:latest
```

Create a Dockerfile for Hugo

The easiest way to deploy the Hugo static site is in a Docker image, as defined by a Dockerfile.

- `FROM` defines the base image (in the example Alpine, a lightweight Linux distribution).
- `COPY` copies files and folders to the Docker image.
- `ARG` specifies arguments for the Docker build command.
- `RUN` executes commands.
- `EXPOSE` informs a user about the ports used.
- `CMD` specifies the component and its arguments to be used by the image.

```
FROM alpine:3.8 as runner
COPY . .
ARG HUGO_VERSION=0.57
RUN apk --no-cache add \
        curl \
        git \
    && curl -SL https://github.com/gohugoio/hugo/releases/download/v
${HUGO_VERSION}/hugo_${HUGO_VERSION}_Linux-64bit.tar.gz \
        -o /tmp/hugo.tar.gz \
    && tar -xzf /tmp/hugo.tar.gz -C /tmp \
    && mv /tmp/hugo /usr/local/bin/ \
    && apk del curl \
```

```
    && rm -rf /tmp/*

EXPOSE 80
CMD hugo --renderToDisk=true --watch=true --port=80 --bind="0.0.0.0" --
baseURL="${VIRTUAL_HOST}" server
```

Start Docker on login (macOS)

Some APIs may require middleware, for example a local *RabbitMQ* instance. It can be convenient to have Docker start the middleware when you log in. For example:

1. Open the *Automator* application.
2. Select the **Application** type and click **Choose**.
3. In the **Actions** menu, select **Utilities** > **Run Shell Script**.
4. In the **Run Shell Script** section, select **/bin/bash** from the **Shell** menu.
5. In the box, enter:

```
cd /usr/local/bin
while (! ./docker stats --no-stream ); do
sleep 10
done
./docker start rabbitmq
```

6. From the **File** menu, click **Save**.
7. Navigate to the `Applications` folder (or your user `Applications` folder).
8. Enter **StartRabbit** in the **Save As** box and click **Save**.
9. Open *System Preferences* and click **Users & Groups**.
10. Select your user and click **Login Items**.
11. Click **Add** (**+**).
12. Navigate to the location where you saved the *StartRabbit* application and select it.
13. Click **Add**.
14. Close *System Preferences*.

The script will run the next time you log in. The path has to be changed because `/usr/local/bin` is not part of the path for startup scripts. The script waits for the Docker daemon to start before starting the rabbitmq container. It does this by querying

Docker until it gets a response. A spinning cog is displayed in the right menu while the script is running.

Kubernetes

If you are deploying more than one Docker instance, you need *Kubernetes*. You can learn more here: `https://kubernetes.io/`

Automation

Jenkins

You should automate your deployments so when your code changes, your API docs are updated. You can learn more about *Jenkins* here: `https://jenkins.io/`

Part II. Tools

IDE

Visual Studio Code

If you're using windows and writing API docs directly in the code, you may be using Visual Studio. If you are not using Windows then then next best thing is Visual Studio Code (`https://code.visualstudio.com/`).

Before you set up your dev environment, ensure you have installed the CLI tools.

Set up a .NET dev environment

1. Download and install the *Visual Studio Code* editor (`https://code.visualstudio.com/`).
2. Download and install the *.NET Core SDK* (`https://dotnet.microsoft.com/download`).
3. Create a folder called `development` where you want to store your local repositories.
4. In *BitBucket Server* (Stash) go to the repository you want to work with and click the clone icon from the left toolbar.
5. Copy the HTTP address.
6. In the *PowerShell* / *Terminal*, navigate to the `development` folder.
7. Enter `git clone` and paste the HTTP address to complete the line.

Build the OpenAPI 2.0 (Swagger) JSON file

This assumes you have a working .NET dev environment and have correctly configured Swashbuckle:

1. In *PowerShell* / *Terminal*, navigate to the `src` folder in your local repository.
2. Enter `dotnet build`.
3. Navigate to the folder where the DLL was built and enter `dotnet run`.
4. In *Chrome*, navigate to the URL shown in the build output. Example: `http://127.0.0.1:5000`.
5. Click the `/swagger/v1/swagger.json` link to download the JSON file.

Plug-ins

Some recommended plug-ins include:

- Better TOML by bungcip
- C/C++ by Microsoft
- C# by Microsoft
- Code Spell Checker by Street Side Software
- GitLens by Eric Amodio
- Markdown Preview Enhanced by Yiyi Wang
- Prettier - Code formatter by Esben Petersen

Preview API doc changes

Requirements

- You can create tasks to automate processes in *Visual Studio Code*.
- Each repository requires its own set of tasks.
- You must add `.vscode/` to the `.gitignore` file to any repository you intend to use tasks with.
- Separate scripts are required for macOS and Windows.

Convert Swagger JSON to ReDoc HTML

Enter: `redoc-cli bundle <filename>.json`.

Running scripts

- After you have configured the scripts for a given repository you can run them from Visual Studio Code.
- Press **command**+**shift**+**B** (macOS) or **ctrl**+**shift**+**B** (Windows) to run the build task.
- The *ReDoc* `HTML` file is opened in *Chrome*.
- If the server takes a long time to start, the ReDoc task will fail. After the server has started, run the **ReDoc** task from the **Terminal** menu.

Example VScode tasks JSON file

This file should be placed inside the `.vscode` folder in the Git repository and *.vscode* added to the `.gitignore` file.

tasks.json

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "build",
            "command": "dotnet",
            "type": "process",
            "args": [
                "build",
                "${workspaceFolder}/src/Dev.Example.Com.Myproject/
Devg.Example.Com.Myproject.csproj"
            ],
            "problemMatcher": "$msCompile"
        },
        {
            "label": "Swagger",
            "type": "shell",
            "command": ".vscode/swagger.sh",
            "windows": {
                "command": ".vscode\\swagger.cmd"
            },
            "presentation": {
                "reveal": "always",
                "panel": "new"
            }
        },
        {
            "label": "ReDoc",
            "type": "shell",
            "command": ".vscode/redoc.sh",
            "windows": {
                "command": ".vscode\\redoc.cmd"
            },
            "presentation": {
                "reveal": "always",
                "panel": "new"
            },
            "problemMatcher": []
```

```
        },
        {
            "label": "Build API Docs",
            "dependsOn": ["Swagger", "ReDoc"],
            "group": {
                "kind": "build",
                "isDefault": true
            },
        }
    ]
}
```

Example scripts (macOS)

redoc.cmd

```
cd ~/
rm redoc.json
rm redoc-static.html

# wait for server to start
echo waiting for service to start
sleep 10

curl -o redoc.json http://localhost:62512/swagger/v1/swagger.json
redoc-cli bundle redoc.json
open -a "Google Chrome" redoc-static.html
```

swagger.cmd

```
export CONSUL=http://consul.dev.example.com
cd src
dotnet build --source http://dev.example.com/myproject/
cd Dev.Example.Com.Myproject
dotnet run --urls=http://localhost:62512
```

Example scripts (Windows)

redoc.cmd

```
cd ~/
```

```
del redoc.json
del redoc-static.html

# wait for server to start
echo waiting for service to start
sleep 10

curl -o redoc.json http://localhost:62512/swagger/v1/swagger.json
redoc-cli bundle recod.json
start chrome.exe redoc-static.html
```

swagger.cmd

```
set CONSUL=http://consul.dev.example.com
cd src
dotnet build --source http://dev.example.com/myproject/
cd Dev.Example.Com.Myproject
dotnet run --urls=http://localhost:62512
```

Part III. Writing

Swashbuckle

Autogenerated Swagger

Swashbuckle is a framework that converts code comments into Swagger doc (`https://github.com/domaindrivendev/Swashbuckle`).

Document API methods with multiple domain models

Some APIs use domain models to perform more than one task using a single API method.

Swagger shows the various domain models as different options in the Request Body Schema. However, it always shows the payload for the first domain model, regardless of the domain model selected.

For these API methods, the schema should be documented in a separate HTML page with a relative link, served by the static site generator.

Documentation for these API methods is declared in a private class as follows:

```
postPath.Post.Summary = "Short name";
postPath.Post.Description = "Long description";
```

The string must be contained on a single line. If you need to insert a carriage return you must use the HTML `<br/>` tag.

The long description should contain the following text:

> The request body schema shown applies only to the first listed command.

This should be followed by a relative link to a markdown page describing the various commands.

Create API docs in VS Code

With Swashbuckle, API documentation is created as comments in the C♯ source code (`.cs` files). These comments are automatically converted to OpenAPI 2.0 (*Swagger*) docs when the application is built. Most content can be written in markdown.

Tag API docs in the code

To make it clear in the code that the comments will be public facing, you can use XML comment format tags around the other comments:

```
/// <!--apiddocs-->
/// ...
/// <!--/apiddocs-->
```

API group descriptions

To enable a summary for a group, the Startups.cs file must include the following:

```
if (File.Exists(xmlPath))
{
    c.IncludeXmlComments(xmlPath, true);
}
```

API methods

Typically, methods are associated with a particular API. However, some methods are inherited and must be edited separately. Documentation can be added using these tags:

- `summary` (plain text): A short description of the method. If no summary is provided, the API name is used.
- `remarks`: A description of the method.
- `param`: A short description of the input parameters.
- `response`: A short description of the response code.

Example:

```
/// <!--apidocs-->
```

```
/// <summary>
/// Find product by SKU.
/// </summary>
/// <remarks>
/// Find a product where the `sku` is known.
/// </remarks>
/// <param name="sku">Stock keeping unit.</param>
/// <response code="400">Bad request.</response>
/// <!--/apidocs-->
[SwaggerOperation(Taqs = new[] { "Products" })]
[HttpGet(Name = GetProductBySkuRoute)]
...
```

Response parameters

Typically, responses for a given API are grouped in a single file. Documentation can be added using summary and example tags. You can find parameters in a project by searching files for `[JsonProperty`. The example tags are always converted to a single string so they should only be used with the following types:

- enum (example: DateTimeOffset)
- GUID (globally unique identifier)
- string

Where you need to provide an example for other types, add a carriage return in the last line of the summary, followed by Example: <your example>. Do not use example tags for arrays. If you use an array example in the summary, you must escape the first square brace (\[,]).

Example:

```
/// <!--apidocs-->
/// <summary>
/// Stock keeping unit.
/// </summary>
/// <example>
/// 013AV_MILK_1L
/// </example>
/// <!--/apidocs-->
[JsonProperty{required = Required.Always}]
public string Sku { get; set; }
```

Required parameters

When parameters are required, you can mark the with `[Required]` but the `.cs` file will need a `using System.ComponentModel.DataAnnotations;` declaration at the beginning of the file.:

If there is already a `[JsonProperty(Required = Required.DisallowNull)]` then do not use `[Required]`.

Example:

```
...
/// <!--/apidocs-->
[Required]
public bool DeductSavingsFromBasket { get; set; }
```

Commenting out comments

If you need to prevent the comments being converted to XML so that you can see what the method name is in the ReDoc output, you can use XML comment syntax:

```
/// <!-- <summary>
/// Stock keeping unit.
/// </summary> -->
```

In XML, certain characters must be escaped or the XML will fail to build from the comments (without warnings). For example, ampersands (`&`) and angle brackets, (`<`, `>`). If you need to use an ampersand with code font style, you must use HTML `<code>` tags. If you use the markdown backtick (`` ` ``) the escaped ampersand will not be converted to a single character.

Markdown

All tagged comments can contain plain text. Some can also contain markdown. Any that can contain markdown can also contain HTML. However, the way markdown and HTML are rendered will vary depending on the tag.

Style	Markdown
heading level 1	`# Heading 1`
heading level 2	`## Heading 2`
heading level 3	`### Heading 3`
bold	`__bold__`
italic	`_italic_`
code font	`` `code` ``

ordered list	`1. item 1`
	`1. item 2`
	`  1. item 2.1`
	`1. item 3`

itemized list	`* item 1`
	`* item 2`
	`  * item 2.1`
	`* item 3`

web link	`[web link](https://www.example.com)`
relative link	`[relative link](../flowcharts/flowchart.md)`
email link	`[email link](mailto:api@example.com?subject=API%20doc%20request&body=I%20am20using20the20APINAME20API20and20req`

table with header	`| Firstname | Lastname | Age |`
	`| --- | --- | --- |`
	`| Jill | Smith | 50 |`
	`| Eve | Jackson | 94 |`

Style	Markdown
line break: created by ending a line with a backslash. These must be tagged with an XML comment to alert developers that the backslash is intentional.	`/// <remarks>Long description.<!--newline-->\` `/// [Request docs](mailto:api@example.com)</remarks>`
program listing	`In <summary> tags, use ``` above and below each code block.` `In <remarks> tags, do not use ```. Ensure all other text runs on from the start and each line ends with <!--newline>\.`  `The last line before the code block will typically be:` `/// __Example__:<!–newline-->\` `///`  `Then add the code, starting on the next comment line:` `{` `"id": "string"`  `}`

Mermaid

Create charts with Mermaid markdown

Mermaid (`https://mermaidjs.github.io/`) is a markdown extension that simplifies the creation of charts. You can add a Mermaid extension to themes used by the Hugo static site generator. You can preview the markdown in VScode using the *Markdown Preview Enhanced* extension.

Before saving the file you must change ```` ```mermaid ```` to `{{<mermaid>}}` and ```` ``` ```` to `{{</mermaid>}}`.

Examples:

````
```mermaid
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```
````

````
```mermaid
graph LR;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```
````

````
```mermaid
sequenceDiagram
participant User
participant Mobile App
participant STS
participant API
User->>Mobile App: 1. Click login link.

loop challenge
Mobile App->Mobile App: 2. Generate code verifier and code challenge.
end
````
````

```
Mobile App->>STS: 3. Authorization code request and code challenge to
 authorize.
STS->>User: 4. Redirect to login/authorization prompt.
User->>STS: 5. Authenticate and consent
STS->>Mobile App: 6. Authorize code.
Mobile App->>STS: 7. Authorization code and code verified to OAuth token.

loop validate
STS->STS: 8. Validate code verifier and challenge.
end

STS->>User: 9. ID token and access token.
User->>API: 10. Request user data with access token.
API->>User: 11. Response.
```

```mermaid
gantt
        dateFormat  YYYY-MM-DD
        title Adding GANTT diagram functionality to mermaid
        section A section
        Completed task            :done,    des1, 2014-01-06,2014-01-08
        Active task               :active,  des2, 2014-01-09, 3d
        Future task               :         des3, after des2, 5d
        Future task2               :         des4, after des3, 5d
        section Critical tasks
        Completed task in the critical line :crit, done, 2014-01-06,24h
        Implement parser and jison          :crit, done, after des1, 2d
        Create tests for parser             :crit, active, 3d
        Future task in critical line        :crit, 5d
        Create tests for renderer           :2d
        Add to mermaid                      :1d
```

Troubleshooting

Create a PDF from OpenAPI JSON

On occasion you may be asked to produce a version of your dev portal as a PDF file. Distilling an interactive website into a PDF is impossible. Most browsers will not even let you print the site in its entirety. However, you can use *Swagger2PDF* to generate a simple PDF from an OpenAPI JSON file:

```
swagger2pdf -s swagger.json
```

Ensure new features and changes are documented

Doc teams typically cover the work of multiple development teams, creating developer and user docs. It is not always possible to attend every scrum meeting and sprint review. To ensure developer and user docs are delivered as close as possible to the sprint in which features are delivered, it is essential that the team receives timely notification.

There are three ways to do this in *Jira*:

- Create a **Documentation sub task** on a ticket.
- Create a **Documentation task** and associate it with a ticket.
- Add the **Documentation** label to a ticket.

You should configure a doc team kanban board to display tickets that meet any of these criteria. You can then watch a ticket to track its progress. Let your developers know how to create doc tickets:

- You can request new documentation for existing features by creating a **Documentation** task.
- The doc team will prioritize documentation requests to best meet customer needs.
- Requests for documentation that are made other than using Jira will be given the lowest priority.

Fix a broken API

Things to try:

- Does the start of the `.cs` file have the following:

```
using Swashbuckle.AspNetCore.SwaggerGen;
```

- Are there incorrectly declared types?

```
Type = "integer" / Type = "boolean"
```

- If the comments are absent from the JSON is the `DocumentationFile` property set in the `.csproj` file (should be set for debug and release). Example:

```
<DocumentationFile>bin\Debug\netcoreapp2.2\Your.ApiGateway.xml</
DocumentationFile>
```

Convert OpenAPI YAML to JSON

ReDoc requires OpenAPI 3.0 / Swagger 2.0 source in JSON format to create a static HTML page. Typically, auto-generated content, such as that produced by Swashbuckle, is already in JSON format. However, hand edited API schema are typically created in YAML.

1. Navigate to `https://editor.swagger.io/`. Alternatively, download the Swagger Editor to use keep the content inside the corporate network (`https://swagger.io/tools/swagger-editor/download/`).
2. Select **File** > **Import File**.
3. Fix any critical schema errors.
4. Select **File** > **Convert and save as JSON**.

Convert Swagger 2.0 to OpenAPI 3.0 to resolve schema errors

ReDoc requires schema in OpenAPI 3.0 format. If you provide a Swagger 2.0 JSON file, it will attempt to do the conversion itself, but schema errors may cause the conversion to fail. Ideally you should fix the schema errors. However, if you require a quick approximation of how the API will appear you can try using the *Mermade* online converter: `https://mermade.org.uk/openapi-converter`.

Force iFrame link to open in parent window

ReDoc produces a static HTML page. The easiest way to add it to a Hugo site while keeping the navigation in the header is to use an iFrame. A better approach would be to ingest the page and dynamically recreate the whole thing. But that would require work by an experienced Hugo developer.

The quick and dirty approach is:

1. Add `<base target="_parent">` in the `<head>` section of the ReDoc `HTML` file.
2. Put the `HTML` pages in Hugo's `static` folder.
3. Create a markdown page in Hugo's `content` folder:

```
---
---
<iframe src="../../../markdownfile/" frameborder="0" allowfullscreen
 style="position:absolute;top:1;left:0;width:100%;height:100%;"></
iframe>
```

Prevent API method links going to the wrong place

In the API docs, clicking a method link in the left pane in ReDoc should take the user to the appropriate method. However, when developers reuse Operation IDs, the link will instead take them to the first instance of that Operation ID. Developers must either use Operation IDs that are unique across the entire API gateway, or use no Operation IDs at all.

Remove .DS_Store files from a Git repository

1. In the shell, navigate to the root of the repository.
2. Enter `find . -name .DS_Store -print0 | xargs -0 git rm -f --ignore-unmatch`.
3. Commit changes.

Install Java 8

Oracle changed the license for JDK 8, effective 16 April 2019. You can still download it from `https://www.oracle.com/technetwork/java/javase/downloads/`

`jdk8-downloads-2133151.html`, but if in doubt, you can use OpenJDK 8 instead (`https://www.azul.com/downloads/zulu/`).

Run Windows apps on macOS

Because API doc tools make heavy reliance on UNIX-land command line tools, it may be preferable to use a Mac. However, traditional technical writing tools may be Windows-only, for example MadCap *Flare*. The easiest way to run Windows programs on a Mac is with *Parallels* (https://www.parallels.com/).

Run a local Rabbit MQ

Some API gateways have RabbitMQ as a build dependency. Because the remote server is not always available, you may need to run a local instance of RabbitMQ. You can use Docker to do this from the command prompt:

```
docker run -d --name rabbitmq -p 15672:15672 -p 5672:5672 bitnami/
rabbitmq:latest
```

1. Navigate to `http://localhost:15672/` and create a user called `rabbituser` with the password `rabbituser` as an administrator.
2. Click the **rabbituser** user and in the **Virtual Host** section, then click **Set permission**.
3. In the source code, locate the `appsettings.Development.json` file and change all instances of `guest` to `rabbituser`.

To restart the service:

```
docker start rabbitmq
```

To remove an old Rabbit MQ:

```
docker system prune -a
```

Appendix A. Style guide

Style sources

If you don't already have a technical communications style guide, you should create one. This should be a short document that lists the main style sources for technical terms, general writing and spelling, any deviations you want to make from those guidelines and a list of any domain-specific terminology. I recommend:

- Microsoft Writing Style Guide (`https://docs.microsoft.com/en-us/style-guide/`) for technical terms.
- AP Stylebook (`https://www.apstylebook.com/`) for general writing guidelines.
- Merriam-Webster (`https://www.merriam-webster.com/`) for spelling.

I have used the *Chicago Manual of Style*, but I find its academic style more suited to publications with a long lead time such as you might find in a waterfall development environment. For technical writing in an agile development environment, I find AP's newspaper style is a better fit.

When writing for an international audience, I use International English (American English). There are many regional variants, but the most popular alternative to International English with a consistent formalized spelling is Hiberno English (*-ise* endings). If you are writing exclusively for an Australian, British, Irish or New Zealander audience, you can use the *Cambridge Dictionary* (`https://dictionary.cambridge.org/`). Other regions will typically accept documentation in International English.

If I could go back in time and prevent the publication of one book, Strunk & White's *The Elements of Style* would be a strong contender. I disagree with Geoffrey K. Pullum that it is "mostly harmless" (`https://www.chronicle.com/article/50-Years-of-Stupid-Grammar/25497`) because its disciples think that they understand how to write well when often they do not.

Common errors

If you're lucky enough to have devs that write docs, don't be too hard on them, but be aware of common errors.

- **after** (not *once*)
- **and so on** (not *etc.*)
- **because** (not *as*)
- **center** (spelling)
- **enable** (not *allow* except when referring to access control)
- **enter** (not *input*)
- **for example** (not *e.g.*)
- **fulfillment** (spelling)
- **IDs** (capitalization)
- **is**/**are** (not *will be*: passive voice)
- **list** (not *dropdown*)
- **organization** (spelling)
- **set up** (action)
- **that is** (not *i.e.*)
- **that**/**which** (contracting/expadning clauses)
- **URL** (capitalization)
- **user's** (apostrophe)
- **using** (not *via*)
- **want** (not *wish*/*would like*)

Boilerplate text

One of the limitations of the current generation of API doc tools is that they have little to no support for automated content reuse. For that reason, you should keep a list of standard definitions and valid payload examples. For example:

Address
: Use your company's HQ.

Barcode
: 019000000002 (*UPC-A format including check digit*)

Brand
: Ownbrand (*too generic to be registered as a trademark*)

Country
USA (*ISO 3166-1 alpha-3 format*)

CreditCardNumber
4111111111111111 (*Use any CVV, NAME and future EXPIRY DATE. Keep purchase value under $500*. `http://testcreditcardnumbers.com/`)

Currency
USD (`https://www.iso.org/iso-4217-currency-codes.html`)

DateTimeOffset
2019-11-01T00:00:00-05:00 (`https://en.wikipedia.org/wiki/ISO_8601` - *use long version to avoid ambiguity*)

DebitCardNumber
6304000000000000 (*bank card number with the defunct Laser identifier*)

Email
username@example.com (*example.com is reserved for examples*)

FirstName
Don't use other terms for *first name*. In some cultures, the family name precedes the given name.

GUID
01234567-890a-bcde-f012-34567890abcd

LastName
Don't use other terms for *last name*. In some cultures, the family name precedes the given name.

Telephone
+1 202 555 0199 (*555 numbers ending 0100-0199 are fictitious*)

URL
`www.example.com` (*example.com is reserved for documentation*)

API doc writing tips

API writing is not the same as traditional technical writing:

- Use a terse, factual writing style. Sentence fragments are desirable. Avoid adjectives and adverbs.

- Provide complete information about each API component.
- Provide working code snippets for each method, function, and resource. You don't need complete examples, but show a common use of that element.
- Provide flow charts (in *Mermaid*) showing the sequence of the most commonly used methods for common use cases.
- Provide sample programs demonstrating common use cases.
- Provide a *Getting Started* guide showing how to develop a program for a common use cases.
- Provide performance and tuning information.
- Provide a contact in case developers have questions or need additional assistance.

Web API terminology

DELETE
: Deletes the specified resource..

GET
: Request data from a specified resource.

HEAD
: Almost identical to GET, but without the response body.

OPTIONS
: Describes the communication options for the target resource.

PATCH
: Applies partial modifications to a resource.

POST
: Send data to a server to create/update a resource.

PUT
: Send data to a server to create/update a resource.

HTTP response codes

Swashbuckle auto-populates response codes. Even if you do not add anything to the code beyond the standard description, if you add a period at the end you'll be able to see in the output document which response codes are auto-populated and which

have been edited. For more information, see `https://en.wikipedia.org/wiki/List_of_HTTP_status_codes`.

1xx: information

100
Continue.

101
Switching protocols.

102
Processing.

103
Early hints.

2xx: success

200
Success.

201
Created.

202
Accepted.

203
Non-authoritative Information.

204
No content.

205
Reset content.

206
Partial content.

207
Multi-status.

208
Already reported.

226
Instance-manipulations used.

3xx: redirection

300
Multiple choices.

301
Moved permanently.

302
Found.

303
See other.

304
Not modified.

305
Use proxy.

306
Switch proxy.

307
Temporary redirect.

308
Permanent redirect.

4xx: client errors

400
Bad request.

401
Unauthorized.

402
Payment required.

403
Forbidden.

404
Not found.

405
Method not allowed.

406
Not acceptable.

407
Proxy authentication required.

408
Request timeout.

409
Conflict.

410
Gone.

411
Length required.

412
Precondition failed.

413
Payload too large.

414
URI too long.

415
Unsupported media type.

416
Range not satisfiable.

417
Expectation failed.

418
I'm a teapot.

421
Misdirected request.

422
Unprocessable entity.

423
Locked.

424
Failed dependency.

425
Too early.

426
Upgrade required.

428
Precondition required.

429
Too many requests.

431
Request header fields too large.

451
Unavailable for legal reasons.

5xx: server errors

500
Internal server error.

501
Not implemented.

502

Bad gateway.

503

Service unavailable.

504

Gateway timeout.

505

HTTP version not supported.

506

Variant also negotiates.

507

Insufficient storage.

508

Loop detected.

510

Not extended.

511

Network authentication required.