Web APIの開発を効率化する方法(ドキュメント編)

@ndruger

本日の内容

Web APIを効率的に開発するためにしているプロジェクトでしているいくつかの工夫、特にドキュメントに関連した部分を紹介する。

APIドキュメントの形式

下記の理由からOpenAPIを利用する。

- Blueprintはツールのメンテナンスがされていない。
- マイナーな形式はツールがない。

リントでドキュメントのエラーをチェックする

- openapi-validatorというツールで、OpenAPIのYAMLをチェックできる。
- CircleCIのジョブでチェックするようにしている。
- デフォルトでibm-cloud-rules.mdのルールが設定されており、設定ファイルで上書きをして利用している。

設定ファイルの例

```
const ibmCloudValidationRules = require('@ibm-cloud/openapi-ruleset')
const { operationIdCasingConvention } = require('@ibm-cloud/openapi-ruleset/src/functions')
const { schemas, operations } = require('@ibm-cloud/openapi-ruleset-utilities/src/collections')

// You can customize with reference https://github.com/IBM/openapi-validator/tree/main/packages/ruleset
module.exports = {
  extends: ibmCloudValidationRules,
  rules: {
    'ibm-enum-casing-convention': 'off', // '400-06' is valid
    'ibm-avoid-inline-schemas': 'off',
    'ibm-avoid-repeating-path-parameters': 'off', // for elixir unit test util doesn't support this
    'ibm-schema-description': 'off',
    'ibm-error-response-schemas': 'off',
    'ibm-string-attributes': 'off', // Response field's minLength/maxLength/pattern is optional
    'ibm-success-response-example': 'off',
    'ibm-property-description': 'off',
    'ibm-requestbody-name': 'off', // For additionalProperties
    'ibm-parameter-description': 'off',
    'ibm-operationid-naming-convention': 'off',
    'ibm-collection-array-property': 'off',
    'ibm-no-array-responses': 'off',
    'ibm-prefer-token-pagination': 'off',
    'ibm-pagination-style': 'off',
    'ibm-operationid-casing-convention': {
      description: 'Property names must follow camel case',
      message: '{{error}}',
      resolved: true,
      given: operations,
      severity: 'warn',
      then: {
        function: operationIdCasingConvention,
        functionOptions: {
          type: 'camel',
        },
      },
    },
  },
}
```

APIドキュメントから一部の実装を生成する(1)

- バリデーションの生成に関しては行わない。
  - パラメーターバリデーションの実装を生成する方法をよく聞くが、好みと合わないので私はあまりしない。
  - コードで使う型定義は広く使いたいのに、生成ツールでOpenAPIから型を作るとコード全体では使い辛い。クラスやモジュールとして実装したくて別のメソッドも持つ場合など。

APIドキュメントから一部の実装を生成する(2)

- 複数のユーザーのロールがある場合、APIドキュメントのAPIごとの`security`フィールドにAPIを呼び出せるロールを記載して、TypeScriptのスクリプトでOpenAPIのYAMLを読み込んで権限判定に利用するコードを生成する。

```
'/console/admin_user':
  get:
    operationId: consoleAdminUserIndex
    security:
      - GlobalAdminApiTokenAuth: []
      - LocalAdminApiTokenAuth: []
    ...
```

生成されるコードとコントローラーでの利用

生成するコードの例。APIと呼び出せるロールのマップを定義する。

```
defmodule Gear.OpenApi do
  defmodule AdminApiRoleMap do
    @api_to_role_ids_map %{
      "consoleAdminUserIndex" => [:global, :local],
      ...
    }
    ...
  end
  ...
end
```

コントローラーのコードがPlugを読み込みPlugの中で上記のコードを利用する。
モジュール名からOpenAPIの `operationId` が生成できるように命名している。

```
defmodule Gear.Controller.Console.AdminUser.Index do
  ...
  plug Gear.Plug.FetchAdminUser, :fetch, api_id: Application.module_to_api_id(__MODULE__)
  ...
end
```

APIドキュメントと実装の整合性をチェックする

- APIドキュメントと実装は乖離しやすい
  - 例) APIに記載されてないエラーステータスコードやエラーレスポンスが返される。
  - 例) APIにないレスポンスフィールドが返される。
  - 例) APIとフィールドのフィールドの型がstringとnumberで異なる。
- ユニットテスト時に、リクエストとレスポンスの内容がAPIドキュメントと一致しているかを自動的にチェックして、合わない場合はテストを失敗させる。

チェック内容

OpenAPIのJSON Schemaを利用したバリデーション + αのチェックしている。

- レスポンスのエラー(ステータスとボディ内のサブコード)が一致するか？
- 正常レスポンスのステータスコードが一致するか？
- クエリ・リクエストボディ・レスポンスボディのフィールドがドキュメント上も存在するか、型や値の制限が一致するか？

ユニットテストのコード

下記のようなAPI呼び出しの中で、前のページのようなチェックがすべて自動的に行われる。

```
# 成功のテストケースの場合
res = get_for_success(@api_schema, @url)

# エラーのテストケースの場合
res = get_for_error(@api_schema, @url)
```