富士通 Java ブログ

MicroProfile OpenAPI を使用したセキュア API のドキュメント化

2023 年 1 月 6 日 初版

数村 憲治

Beyond the Twelve-Factor App でも言及されているように、API の設計はモダンなアプリケーション開発において重要な位置づけになっています。また、API のふるまいだけではなく、その API が前提とするセキュリティ要件も設計段階で明確にしておくことが、セキュアな API 構築につながると考えられています。

本稿では、MicroProfile Interoperable JWT RBAC（以降、MP JWT と記載）を使用したセキュア API を例に、MicroProfile OpenAPI（以降、MP OpenAPI と記載）でドキュメント化する方法について解説します。ただし、本稿では、MP OpenAPI に関する基本事項については説明しませんので、必要に応じて、過去の記事を参照ください。

- MicroProfile OpenAPI による API 管理（Part1）：既存の Java コードをドキュメント化する
- MicroProfile OpenAPI による API 管理（Part2）：MicroProfile OpenAPI annotation を利用しドキュメント化する

ここで紹介するプログラムの完全なソースコードは、以下で参照可能です。

- https://github.com/fujitsu/app_blog/tree/master/openapi-jwt-202212（GitHub のページへ）

MP JWT を使用した MicroProfile アプリケーション

本章では、MP JWT を使用した MicroProfile アプリケーションを例に、MP OpenAPI のドキュメント化方法を簡単に紹介します。

MP JWT は、マイクロサービスのエンドポイントに対して、JSON Web Tokens（JWT）を使用した、ロールベースアクセス制御（Role Based Access Control）を提供します。MP JWT については、過去の記事もご参照ください。

- MicroProfile JWT によるスケーラブルでセキュアなマイクロサービス構築 – 入門編（Part 1）–
- MicroProfile JWT によるスケーラブルでセキュアなマイクロサービス構築 – 入門編（Part 2）–

以下のプログラムは、Jakarta RESTful Web Services のアプリケーションで、ユーザ ID を指定した URL（例："http://example.com/user/100"）をリクエストすると、該当するユーザ名を返却する API を定義しています。このプログラムでは、（1）の@RolesAllowed に記載しているロールを持つユーザがアクセスでき、ロールは JSON Web Token 内で指定しアプリケーションに渡します。API の概要については（2）で、API に指定するパラメタについては（3）で、API のレスポンスについては（4）/（5）で、 MP OpenAPI を使用しドキュメント化されています。

```
@Path (”/user”)
public class User {
  @GET
  @Path (”/{id}”)
  @Produces (”text/plain”)
  @RolesAllowed (”管理者”)・・・(1)
  @Operation (summary = ”JWT を使用したセキュアな呼出し”)・・・(2)
  @Parameter (name = ”id”, description = ”ユーザ ID”)・・・(3)
  @APIResponse (responseCode = ”200”,
                description = ”指定した ID に該当するユーザの名前を表示します。”,
                content = @Content (mediaType = ”text/plain”))・・・(4)
  @APIResponse (responseCode = ”404”, description = ”指定した ID に該当するユーザが存在しません。”)・・・(5)
```



```
public Response user (@PathParam ("id")  int id)  {
  String name = getName (id) ;
  if  (name == null)
    return Response.status (Response.Status.NOT_FOUND) .entity ("該当ユーザは存在しません¥n") .build () ;
  else
    return Response.ok (name) .build () ;
}
```

上記プログラムを実行するために、MicroProfile 5.0 互換実装の一つである、Launcher を用意します。 Launcher は以下よりダウンロードし利用します。

- https://github.com/fujitsu/launcher/releases/download/4.0/launcher-4.0.jar

Launcher の使い方は、ダウンロードした launcher-4.0.jar を任意の場所に置いて、java コマンドの -jar オプションに指定するだけです。（インストール作業は不要です。）詳細な Launcher の使用方法は、以下のドキュメントを参照してください。

- https://github.com/fujitsu/launcher/blob/master/docs/Usage.adoc

```
$ java  -Dmp.jwt.verify.publickey.algorithm=ES256  -Dmp.jwt.verify.publickey={Base64 エンコードした JWK}  -jar
launcher-4.0.jar --deploy secureapi-1.0.war
```

このプログラムのビルド・実行の詳細については、GitHub を参照ください。

上記プログラムのような MP OpenAPI でドキュメント化された内容を可読化するためにはいくつか方法があります。一つは、yaml 形式で表示する方法で、該当 MicroProfile アプリケーションに対して、MP OpenAPI のエンドポイント（この場合であれば、"http://localhost:8080/openapi"）を指定します。もう一つは、Swagger UI を使用する方法で、該当 MicroProfile アプリケーションに対して、Swagger UI のエンドポイント（この場合であれば、"http://localhost:8080/openapi-ui"）を指定します。いずれも、図 1 や図 2 に示すように、API の呼出し方法やレスポンスに関して、必要な情報が出力されているようにみえます。

図 1：MP OpenAPI エンドポイント

図 2：Swagger UI

しかし、実際に、この API にアクセスするためには、適切な JSON Web Token を Authorization ヘッダに付与して呼び出す必要がありますが、そのことについての記述がありません。先ほどの、Swagger UI の「Try it out」ボタンから、API のテスト呼出しが行えますが、単純に、ユーザ ID を指定するだけでは、レスポンスコード 401 が返却されます。図 3 の例では、ユーザ ID に、「10」を指定し実行した結果ですが、ドキュメントに記述のない、「401 Error: Unauthorized」というレスポンスが返却されています。

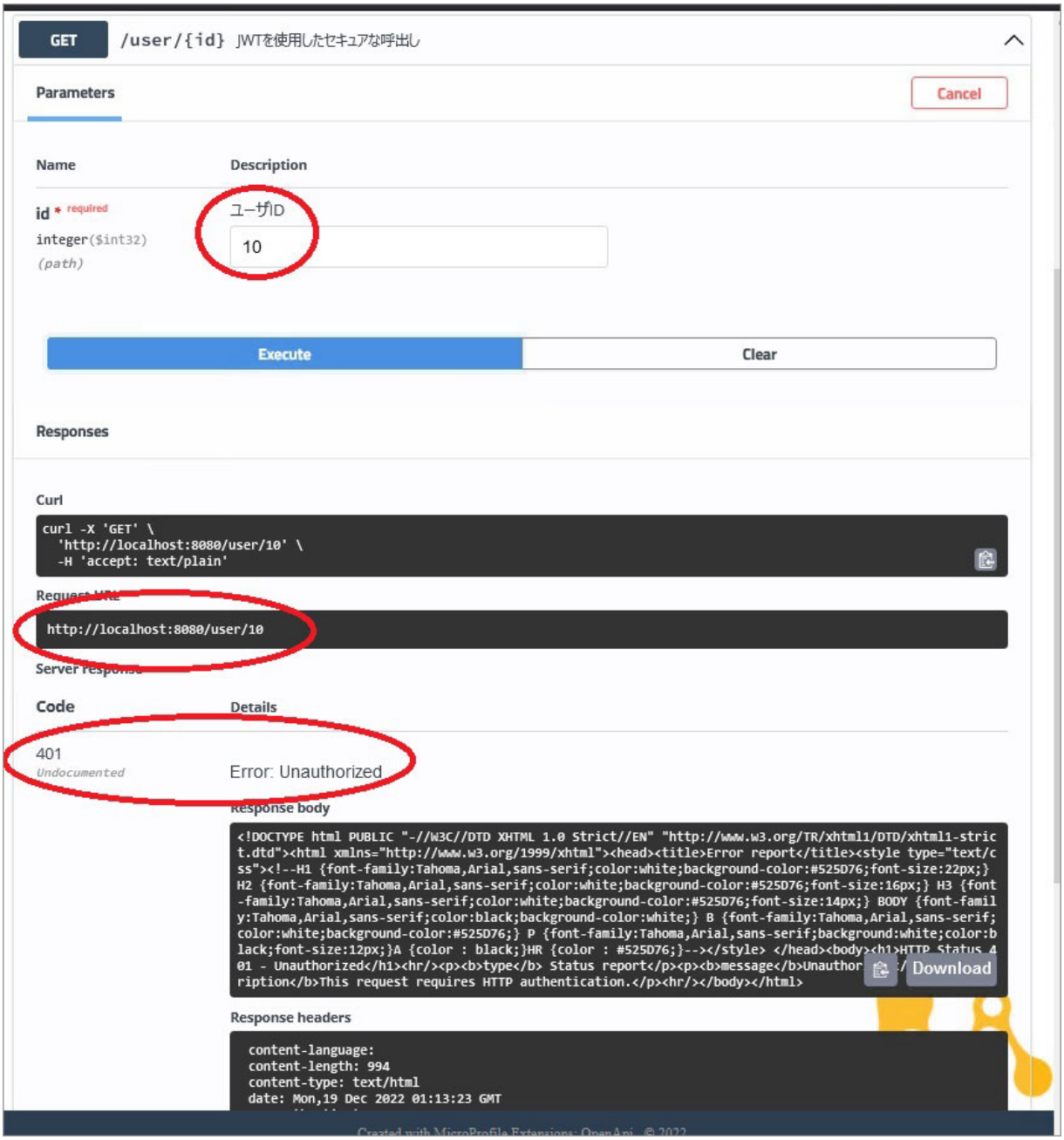

図 3：ユーザ ID に「10」を指定して、API を実行した結果

次章では、このような、API が前提とするセキュリティ要件を、MP OpenAPI でドキュメント化する方法を紹介します。

セキュリティスキーマの使用

OpenAPI では、セキュリティ関連情報を記述する、 セキュリティスキーマ が定義されています。MP OpenAPI では、このスキーマを @SecurityScheme アノテーションで記述できます。前章のプログラムを、@SecurityScheme を使って書き直したものが以下になります。

```
@Path (”/secureuser”)
@SecurityScheme (securitySchemeName = ”JWT-Auth”,  ・・・(1)
                 type = SecuritySchemeType.HTTP, ・・・(2)
                 scheme = ”bearer”, ・・・(3)
                 bearerFormat = ”JWT”, ・・・(4)
                 description = ”JWT を使用したスキーマ。¥n 適切な Json Web Token を Authorization ヘッダに指定してください。”)
public class SecureUser {
  @GET
```



```
@Path ("/{id}")
@Produces ("text/plain")
@RolesAllowed ("管理者")
@Operation (summary = "JWT を使用したセキュアな呼出し")
@Parameter (name = "id", description = "ユーザ ID")
@APIResponse (responseCode = "200",
              description = "指定した ID に該当するユーザの名前を表示します。",
              content = @Content (mediaType = "text/plain"))
@APIResponse (responseCode = "404", description = "指定した ID に該当するユーザが存在しません。")
@APIResponse (responseCode = "401", description = "不適切なトークンによる呼出し。")・・・(5)
@SecurityRequirement (name = "JWT-Auth")・・・(6)
public Response user (@PathParam ("id")  int id)  {
```

@SecurityScheme には、どのようなセキュリティ手段を使うのかを記述します。 （1）で、このスキーマの名前を指定します。この名前は、後の、@SecurityRequirement のパラメタに指定します。 今回は、JWT を使うため、上記（2）、(3)、(4）記載のように、 type に SecuritySchemeType.HTTP、scheme に"bearer"、bearerFormat に"JWT"を指定します。 SecuritySchemeType.HTTP を指定すると、Authorization ヘッダが使われるようになります。 したがって、この、(2)、(3)、(4）の記述により、

```
Authorization: Bearer
```

という HTTP ヘッダが使われることになります。

@SecurityScheme はスキーマを定義するものですが、 定義したスキーマを使うには、 （6）に記載のように、エンドポイントに対して、@SecurityRequirement を使用します。 name パラメタには、対応する@SecurityScheme の securitySchemeName で指定した値を指定します。 また、(5）のように、適切な JWT を API に指定しなかった場合のレスポンスに関する説明も記述しておきます。

@SecurityScheme と@SecurityRequirement を使用するように変更した上記アプリケーションを Swagger UI で見ると図 4 のようになります。

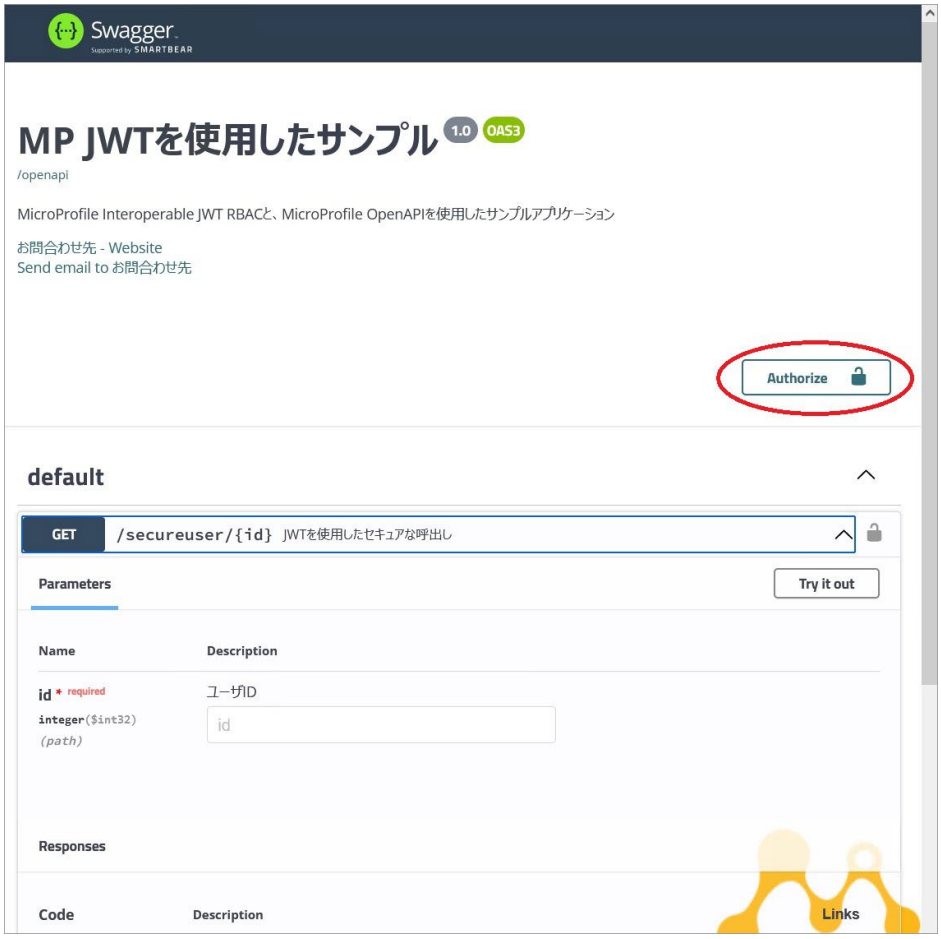

図 4：セキュリティスキーマを使用

画面右側に、「Authorize」というボタンが新たに出現するようになりました。 このボタンを押すと、図 5 のオーソライズ情報入力画面が出現します。

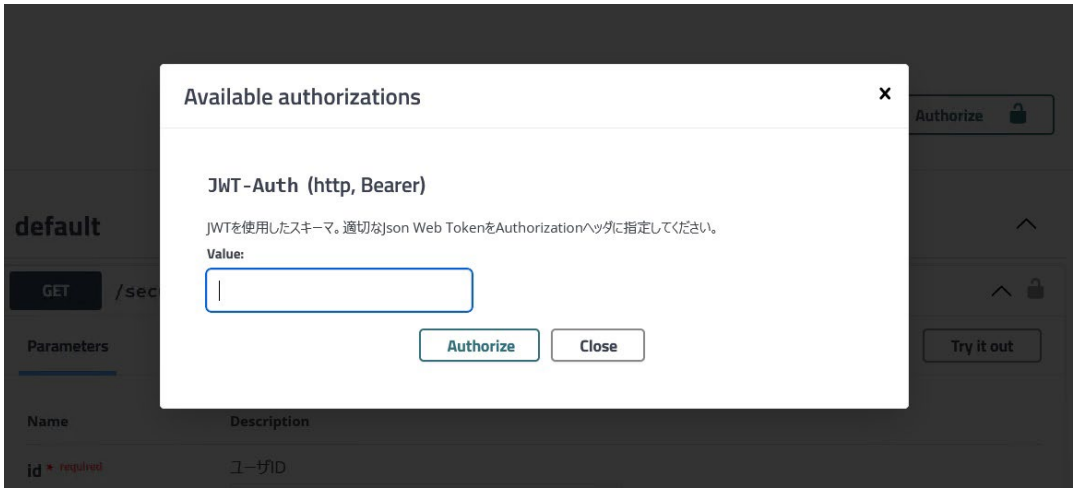

図 5：MP OpenAPI エンドポイント

このダイアログ画面には、@SecurityScheme に記載したセキュリティ要件が表示され、 API 利用者がセキュリティに必要な情報を入力することができます。 今回の例では、図 6 のように適切な JWT を入力します。

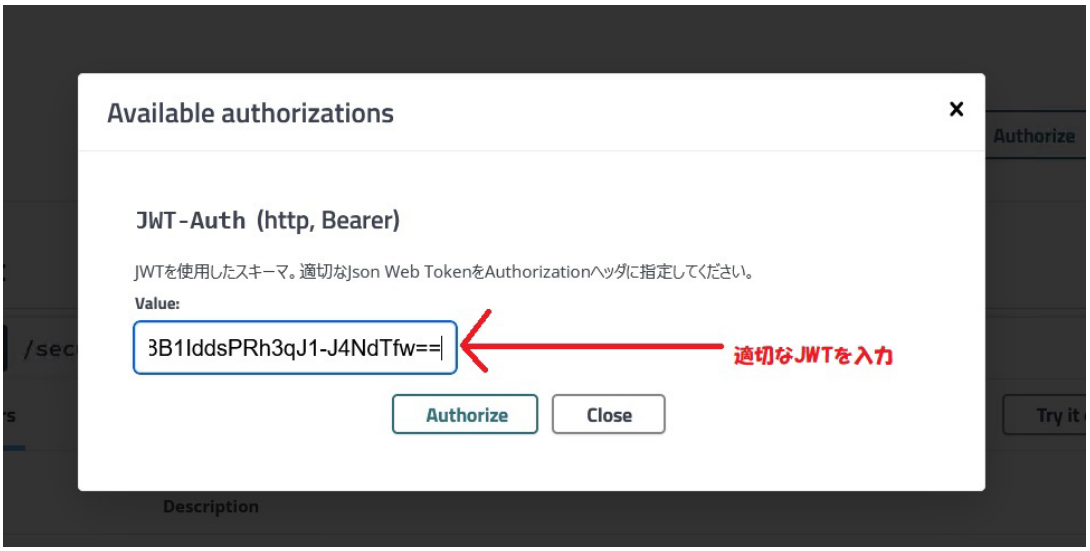

図 6：JWT を入力

オーソライズ情報入力画面を閉じると、図 4「セキュリティスキーマを使用」画面の右側に出ていた鍵のアイコンの絵が、 図 7 のように、開いていた状態から閉じた状態に変わります。

図 7：鍵マーク

鍵が閉じた状態で、ユーザ ID を指定し、リクエストを送信します（Execute ボタンを押下）。 「Curl」の欄には、Authorization ヘッダが、指定した JWT とともに付与されていて、 HTTP ステータスコードが 200 で、正常なレスポンスが返りました。

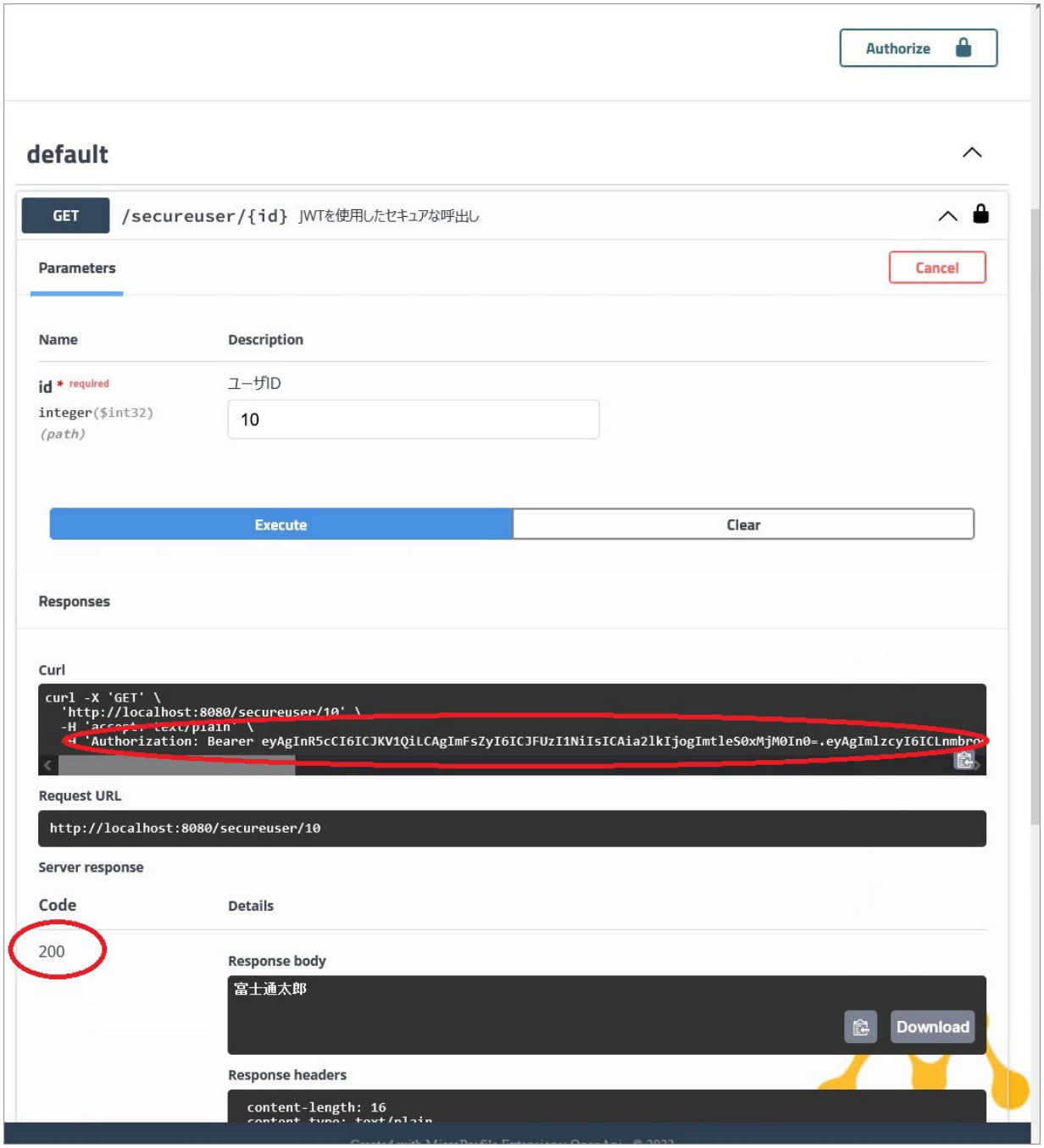

図 8：正常な結果

まとめ

本稿では、MicroProfile OpenAPI を使った、Jakarta RESTful Web Services のアプリケーションのドキュメント化方法、 特にセキュリティスキーマ、また、Swagger UI での検証方法について紹介しました。 セキュアな API の設計・ドキュメント化の際には、本稿で紹介した方法を参考にしてください。