.Net Core WebApi Swagger

1 Swagger Code erzeugen

Die API-Information, die im Backend durch Swagger aufbereitet wird, kann man jetzt dazu benutzen, um Code für verschiedene Programmiersprachen zu erzeugen, z.B. Javascript, Typescript, Java, ….
Dazu gibt es viele verschiedene Tools die als Desktop-App oder aber auch von der Konsole aus zu bedienen sind.
Als Desktop-App kann man z.B. `NSwagStudio` benutzen.
Für die Konsolenanwendung soll jetzt `openapi_generator_cli` besprochen werden, die mir am einfachsten zu verwenden erscheint. Das Projekt liegt auf github: https://openapi-generator.tech/docs/installation/
Das kann man entweder global oder mit npm installieren. Letztendlich braucht man aber nur eine Datei, die man auch manuell in das Projekt kopieren könnte.

1.1 openapi-generator-cli

Das Tool ist so aufgebaut, dass man aus einem OpeApi-Schema server- oder clientseitigen Code erzeugen kann. Mit sogenannten Generatoren gibt man an, was die Zielplattform ist. In unserem Fall ist das Typescript für jQuery und daher wird die Option **`-g typescript-jquery`** angegeben.
Hinweis: zum Erzeugen des Client-Codes **muss das Backend gestartet sein**!

1.2 Installieren

1.2.1 jar-Datei

Grundsätzlich genügt die Datei openapi-generator-cli-5.1.1.jar. Um damit die Javascript-Zugriffsroutinen zu erzeugen, muss man folgenden Befehl eingeben:

```
java -jar openapi-generator-cli-5.1.1.jar generate -i
http://localhost:5000/swagger/v1/swagger.json -g typescript-jquery -o
.\src\swagger
```

Besser ist aber die folgende Variante mit npm.

1.2.2 npm

Zuerst das Paket mit

```
npm install @openapitools/openapi-generator-cli --dev
```

bzw.

```
yarn add @openapitools/openapi-generator-cli --dev
```

installieren.
Dann einmal **`openapi-generator-cli`** eingeben. Dadurch wird die neueste Version heruntergeladen und die Datei **`openapitools.json`** erzeugt (siehe weiter unten).

```
C:\_PR\CSharp\PR4\300-399\350_Tutorings\FrontendTypescriptSwagger>openapi-generator-cli
Download 5.4.0 ...
Downloaded 5.4.0
Did set selected version to 5.4.0
Usage: openapi-generator-cli <command> [<args>]
```

1.3 Erzeugen

1.3.1 Konsole

Zum Erzeugen folgenden Befehl eingeben:

```
openapi-generator-cli generate -i
http://localhost:5000/swagger/v1/swagger.json -g typescript-jquery -o
.\src\swagger
```

1.3.2 openapitools.json

Die Optionen kann man auch im erwähnten openapitools.json festlegen. Alle Optionen siehe https://github.com/OpenAPITools/openapi-generator-cli#configuration. Für obige Einstellungen entspricht das folgendem Inhalt:

```
{
  "$schema": "./node_modules/@openapitools/openapi-generator-cli/config.schema.json",
  "spaces": 2,
  "generator-cli": {
    "version": "5.4.0",
    "generators": {
      "jQuery": {
        "generatorName": "typescript-jquery",
        "output": "./src/swagger",
        "inputSpec": "http://localhost:5000/swagger/v1/swagger.json"
      }
    }
  }
}
```

Ausführen dann mit

`openapi-generator-cli generate`

bzw. mit expliziter Angabe des Generators

`openapi-generator-cli generate --generator-key jQuery`

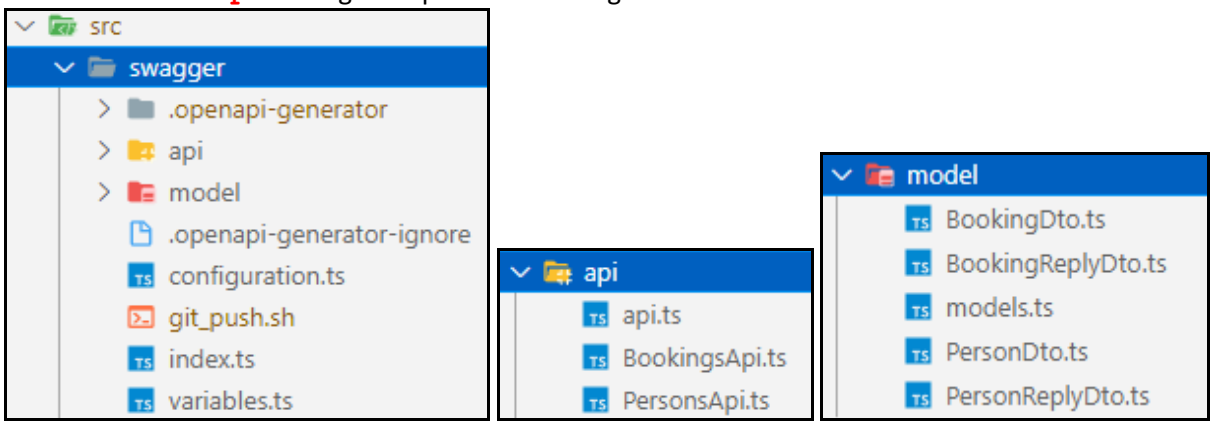
1.3.3 npm-Task

Wie gewohnt bietet es sich an, das als npm-Task zu definieren:

```
"scripts": {
  "tsc": "tsc",
  "swaggergen": "openapi-generator-cli generate --generator-key jQuery",
  "watch:tsc": "tsc --watch"
```

1.4 Erzeugter Code

Im angegebenen Verzeichnis (in unserem Fall src\swagger) wird dann pro Controller eine eigene Klasse im Unterverzeichnis **`api`** erzeugt und pro DTO eine eigene Klasse im Verzeichnis **`model`**.



Zusätzlich werden noch Referenzdateien in diesen Verzeichnissen angelegt, die alle Klassen gesammelt exportieren, nämlich api.ts und models.ts:

```
swagger > api > api.ts > ...
1   export * from './BookingsApi';
2   import { BookingsApi } from './BookingsApi';
3   export * from './PersonsApi';
4   import { PersonsApi } from './PersonsApi';
5   export const APIS = [BookingsApi, PersonsApi];
```

swagger > model > models.ts

```
export * from './BookingDto';
export * from './BookingReplyDto';
export * from './PersonDto';
export * from './PersonReplyDto';
```

Als gesamte Referenzdatei steht index.ts zur Verfügung:

src > swagger > index.ts

```
export * from './api/api';
export * from './model/models';
export * from './variables';
export * from './configuration';
```

1.5 Verwenden

Die Imports erfolgen am einfachsten von dem oben erwähnten index.ts, denn dann muss man sich über die interne Dateistruktur des generierten Codes keine Gedanken machen:

```
import {
  BookingDto, BookingReplyDto, PersonReplyDto,
  BookingsApi, PersonsApi
} from './swagger/index';
```

Im Constructor muss man dann die Basis-Url des Backends angeben:

```
const baseUrl = 'http://localhost:5000';
const personsApi = new PersonsApi(baseUrl);
```

Mit dieser Api-Klasse ruft man dann die verschiedenen Backend-Actions auf:

```
personsApi.personsGet().then(x => x.body)
  .then(data => {
    console.table(data);
    persons = data;
```

Achtung: man muss aus dem Reply zuerst mit `.then(x=>x.body)` die Daten extrahieren!

Aufgrund der import-Anweisungen muss beim bundlen nur dist/index.js angegeben werden:

```
"bundle": "browserify dist/index.js -o dist/app.js",
"watch:bundle": "watchify dist/index.js -o dist/app.js",
```