

Exercise 6

Documenting your service with Swagger / OpenAPI

Prior Knowledge

Basic understanding HTTP verbs, REST architecture
Service from Exercise 5

Objectives

Understanding Swagger and how to use tsoa to enable it in Typescript

Software Requirements

(see separate document for installation of these)

- Exercise 5 + requirements
- Swagger UI

Overview

The OpenAPI specification, which is still often known as Swagger - is an effective JSON/YAML model for describing RESTful services.

One of the main reasons we chose tsoa to build our service is that it takes an “OpenAPI-first” approach. This makes it very easy to modify our service to generate OpenAPI.

Steps

1. Start with your **purchase-starter** directory from the previous exercise.

(It should be passing all the tests.)
2. Before we do any coding, let's look at the way the project is setup.

code tsoa.json

```
1  {
2    {
3      "entryFile": "src/app.ts",
4      "noImplicitAdditionalProperties": "throw-on-extras",
5      "controllerPathGlobs": ["src/**/*.Controller.ts"],
6      "spec": {
7        "outputDirectory": "src/generated",
8        "specVersion": 3
9      },
10     "routes": {
11       "routesDir": "src/generated"
12     }
13   }
14 }
```

Notice that this time there is an extra stanza in the file compared to the previous exercise. Lines 6-9 say to use OpenAPI spec version 3:

<https://swagger.io/specification/>

and to output the swagger into the same directory as before (src/generated).

3. We can immediately generate the OpenAPI:

```
tsoa spec
```

4. This will create:

```
src/generated/swagger.json
```

Take a look. It will look a bit like:

```
home > oxsoa > purchase-starter > src > generated > {} swagger.json > ...
1  {}
2  "components": {
3    "examples": {},
4    "headers": {},
5    "parameters": {},
6    "requestBodies": {},
7    "responses": {},
8    "schemas": {
9      "PurchaseOrder": {
10        "properties": {
11          "id": {
12            "type": "string"
13          },
14          "poNumber": {
15            "type": "string"
16          },
17          "lineItem": {
18            "type": "string"
19          },
20          "quantity": {
21            "type": "number",
22            "format": "double"
23          },
24          "date": {
25            "type": "string",
26            "format": "date-time",
27            "default": "2021-03-28T07:04:00.086Z"
28          },
29          "customerNumber": {
30            "type": "string"
31          },
32        }
33      }
34    }
35  }
```

5. This is a valid Swagger/OpenAPI spec. However, we can improve it a bit. You might question why I chose to generate it into the src folder? We can get the service itself to serve this up, and even to serve a nice UI to visualise it, so it's handy to consider it part of the src tree.

However, in a later exercise we are going to discover how we can use it in an API manager. In this model it would be better to export it somewhere else.

6. Let's enable the "Swagger UI" to serve a nice webpage.

tsoa uses another package to do this - let's install it:

```
yarn add swagger-ui-express
yarn add --dev @types/swagger-ui-express
```

7. Let's add this to app.ts:

In the imports add:

```
import * as swaggerUi from "swagger-ui-express";
const swaggerDocument = import('./generated/swagger.json');
```

8. Then add (next to other app.use() code lines):

```
app.use("/api-docs", swaggerUi.serve,
  async (_req: express.Request, res: express.Response) => {
    return res.send(
      swaggerUi.generateHTML(await swaggerDocument)
    );
  });
```

9. What does this do? Basically when someone calls <http://localhost:8000/api-docs> this will generateHTML against the swagger doc that was generated by tsoa.
10. Before we run that, let's edit the scripts in package.json to make sure that we always build the specs.

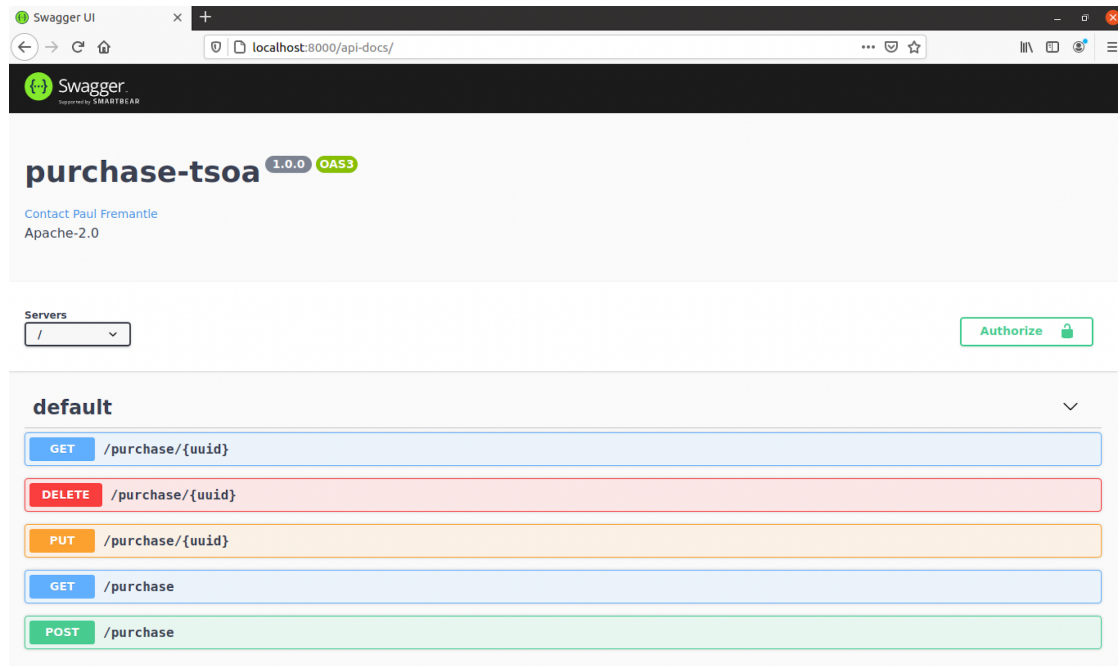
Change where it says "tsoa routes" to "tsoa spec-and-routes"

```
Debug
"scripts": {
  "start": "tsoa spec-and-routes && ts-node src/app.ts",
  "dev": "nodemon --watch . --ignore 'src/generated/routes.ts' --ext ts --exec \"tsoa spec-and-routes && ts\",
  "test": "newman run purchase-tests.postman_collection.json",
  "test-dev": "nodemon --watch . --ignore 'src/generated/routes.ts' --ext ts --exec \"sleep 5 && newman run\",
  "lint": "yarn run eslint src/",
  "build": "tsc",
  "watch": "tsc --watch"
},
```

11. Start your server: e.g.

```
yarn dev
```

12. Now browse to <http://localhost:8000/api-docs>



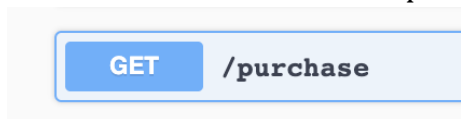
You should see a nice UI.

Where did the name **purchase-tsoa** and version **1.0.0** come from?
These are automatically picked up from the project name and version in package.json!

13. Try changing them. Notice that our nodemon watch doesn't watch for package.json so you will have to restart your server.

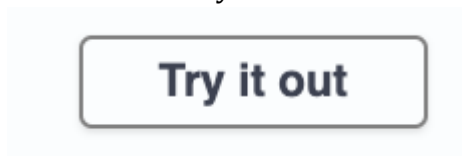
14. Let's try the service out from the Swagger UI.

15. Go to the button GET with no parameters:

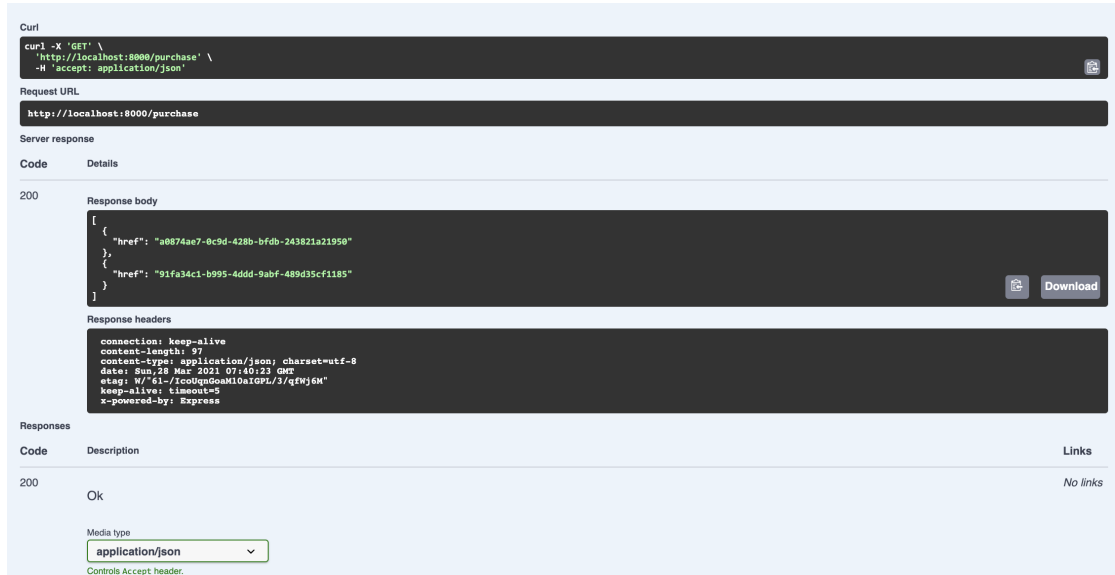


Click on it.

16. Now click on Try it Out:



17. You should see something like:



Curl

```
curl -X 'GET' \
  'http://localhost:8000/purchase' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8000/purchase

Server response

Code	Details
200	<p>Response body</p> <pre>{ "href": "a0874ae7-0c9d-428b-bfdb-243821a21950", }, { "href": "91fa34c1-b995-4ddd-9abf-489d35cf1185" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-length: 97 content-type: application/json; charset=utf-8 date: Sun, 18 Mar 2021 07:10:23 GMT etag: W/"61-/icoUqoGaoMIOaIGPL/3/qfWj6M" keep-alive: timeout=5 x-powered-by: Express</pre>

Responses

Code	Description	Links
200	Ok	No links

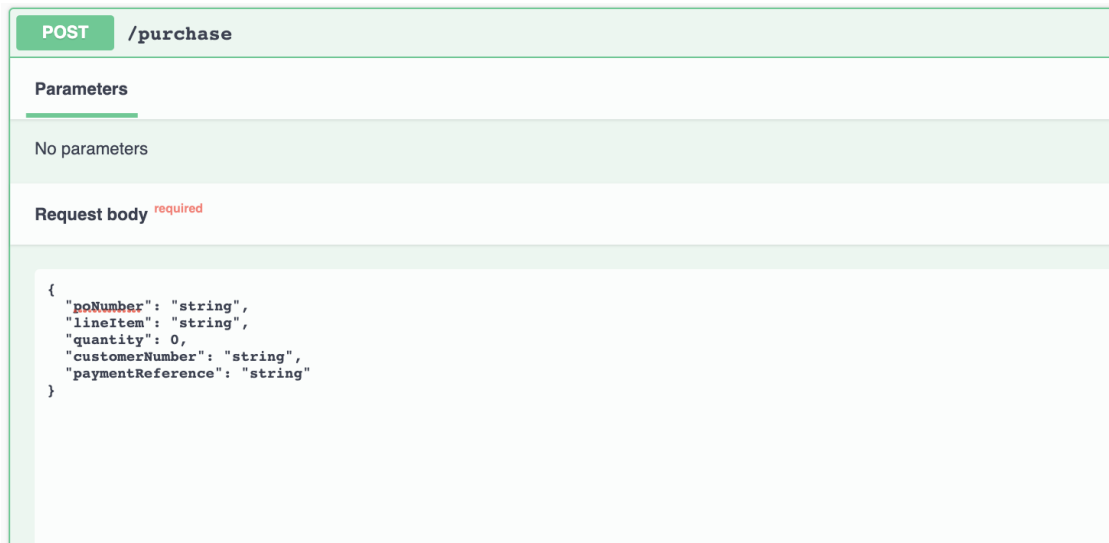
Media type

application/json

Controls Accept header.

18. Use this capability to GET and DELETE a specific uuid.

19. Now go to create a new order with POST:



POST /purchase

Parameters

No parameters

Request body **required**

```
{
  "poNumber": "string",
  "lineItem": "string",
  "quantity": 0,
  "customerNumber": "string",
  "paymentReference": "string"
}
```

20. This is pretty cool! The swagger generation has looked at the POCreationParams type (the subset of our purchase model), and the UI has now generated an acceptable JSON doc for you to edit.

Create a couple of orders this way.

21. This documentation is ok, but it is lacking a lot. Let's go to the PurchaseController and document the @GET method.

Just above @Get("/{uuid}")

```
@Get("/{uuid}")  
public async getPurchase(@Path() uuid: string
```

type /** and then hit Enter.

vscode should automatically complete:

```
/**  
 *  
 * @param uuid  
 * @returns  
 */
```

This is based on a system called JSDoc (<https://jsdoc.app/>)

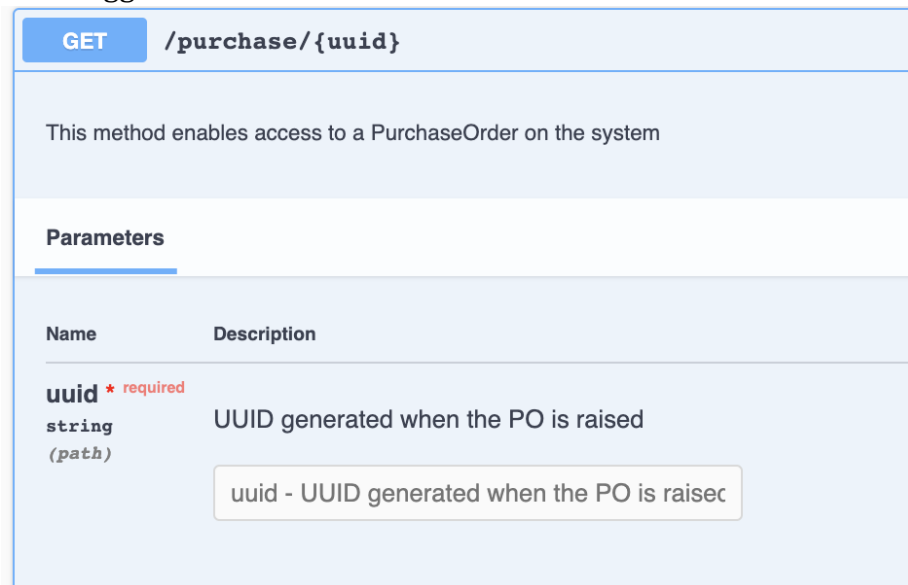
Fill it in for example like this:

```
/**  
 * This method enables access to a PurchaseOrder on the system  
 * @param uuid UUID generated when the PO is raised  
 * @returns PurchaseOrder or Error  
 */  
@Get("/{uuid}")  
public async getPurchase(@Path() uuid: string  
): Promise<PurchaseOrder | ErrorReport> {
```

22. Note that the documentation for POST will probably not correctly describe the "successful" return code. We can fix that with a decoration of that method:

```
@SuccessResponse("201", "Created")  
    // Custom success response  
@Post()
```

23. Assuming you have yarn dev and nodemon watching, now go and reload the Swagger UI:



Pretty sweet huh?

24. We can also decorate our model with jsdoc:

```

4  ✓ /**
5    * PurchaseOrder objects represent orders for purchases
6    */
7    @Entity()
8  ✓ export class PurchaseOrder {
9
10     @PrimaryGeneratedColumn("uuid")
11     id: string;
12
13  ✓    /**
14     * The poNumber is your purchase order number which we will track
15     */
16  ✓    @Column({
17       length: 256
18     })
19     poNumber: string;
20
21  ✓    /**
22     * The line item must match our catalogue
23     */
24     @Column("text")
25     lineItem: string;
26

```

25. This comes through in the schemas in the UI (see below the VERBS):

Schemas

PurchaseOrder ▾ {

description:

PurchaseOrder objects represent orders for purchases

id*

string

poNumber*

string

The poNumber is your purchase order number which we will track

lineItem*

string

The line item must match our catalogue

quantity*

number(\$double)

date

string(\$date-time)

default: 2021-03-28T07:56:54.378Z

customerNumber*

string

paymentReference*

string

isDeleted

boolean

default: false

26. Note that this is where our cool “Pick” approach would need tidying up a bit :-) The schemas that are generated aren’t that readable:

```
POCreationParams {
  description:      From T, pick a set of properties whose keys are in the union K
  poNumber*        string
                   The poNumber is your purchase order number which we will track
  lineItem*        string
                   The line item must match our catalogue
  quantity*        number($double)
  customerNumber*  string
  paymentReference* string
}
```

27. There is more you can do with tsoa to describe:

<https://tsoa-community.github.io/docs/descriptions.html>

and to provide examples:

<https://tsoa-community.github.io/docs/examples.html>

28. That’s the main lab concluded. There are some extensions if you want.

Extension:

1. Take a look at redoc and redoc-cli

<https://github.com/Redocly/redoc>

<https://github.com/Redocly/redoc/blob/master/cli/README.md>

Use redoc cli to create a standalone HTML that documents your API

2. Sign up with Swagger Hub's free trial:

<https://swagger.io/tools/swaggerhub/>

Use the tool to design an API from scratch. You can use your own ideas, but if you want one from me, how about creating an API for a simple todo list tracker.