José Angel Bustamante

Challenge for Tekton

Instantiation & Instructions Document

Version 1.0

03/28/2024

Working with Docker-Compose

There are some prerequisites for using the included docker-compose.yml files:

- Make sure you have docker installed (on windows install docker desktop)
    - Create and install an https certificate in Windows:

```
dotnet dev-certs https -ep $env:USERPROFILE\.aspnet\https\cert.pfx -p password!
```

- Create and install an https certificate in Unix:

```
dotnet dev-certs https -ep ~/.aspnet/https/cert.pfx -p password!
```

It's possible that the above step gives you an error message: "*`A valid HTTPS certificate is already present`* *error*". In that case you will have to run the following command, and then `Re-Run Step 2`

```
dotnet dev-certs https --clean
```

Trust the certificate

```
dotnet dev-certs https --trust
```

Docker-Compose Commands

Tekton WebAPI includes 3 Docker-Compose Files

- WebAPI + PostgreSQL (default)
- WebAPI + MSSQL
- WebAPI + MYSQL

WebAPI + PostgreSQL (default)

```
docker-compose -f docker-compose.postgresql.yml up -d
docker-compose -f docker-compose.postgresql.yml down
```

WebAPI + MSSQL

```
docker-compose -f docker-compose.mssql.yml up -d
docker-compose -f docker-compose.mssql.yml down
```

WebAPI + MYSQL

```
docker-compose -f docker-compose.mysql.yml up -d
docker-compose -f docker-compose.mysql.yml down
```

Your API should be available at `https://localhost:5100/swagger` and `http://localhost:5010/swagger`

Specifications

Let's first examine the Environment Variables passed into the tekton-webapi container.

- ASPNETCORE_ENVIRONMENT : Custom Environment Name.
- ASPNETCORE_URLS : Enter in the Port list.
- ASPNETCORE_HTTPS_PORT : Custom SSL Port.
- DatabaseSettings__ConnectionString : Valid Connection String.
- HangfireSettings__Storage__ConnectionString : Valid Connection String.
- DatabaseSettings__DBProvider : This will be the database engine.
- HangfireSettings__Storage__StorageProvider : This will be the database engine.
- ElasticSearchUrl : Example http://localhost:9200/
- ExternalDiscountsUrl : Example https://mockapi.io/projects/6603b4cb2393662c31cf74e9/
- DiscountsRequestUrl : This is the trailing uri of the External Discount Service URL (Example: products/)
- CacheExpirationMinutes : Minutes in Cache for the product status. Defaulted to 5.

VOLUMES: Getting the Log Files

In order to get the Log Files on the Host file system, so they can be analyzed, the right volumes must be set:

```
volumes:
      - ~/.aspnet/https:/https:ro
      - ./Logs:/App/Logs
```

- ~/.aspnet/https : This is the path of the Certificate generated in Step 2.
- ./Logs: This is the path of an existing 'Logs' folder on the Host machine. All the generated logs will be stored there.

OBSERVABILITY: Viewing the Log Files

By setting the Environment Variable "ElasticSearchUrl" as outlined in the Specification Section, logging data will be aggregated using the Elastic Stack, allowing for visualization through Kibana when utilizing docker compose.

Each of the docker-compose will have the same exact variables with values suited to the context.

Note that the default Docker Image that will be pulled is `angelbus/tekton-webapi:latest`. This is my public Image Repository.

DEVELOPMENT: Quick Start Guide

So, for a better developer experience, I have added Makefile into the solution. Now that our solution is generated, let's navigate to the root folder of the solution and open up a command terminal.

To build the solution,

```
make build
```

By default, the solution is configured to work with postgresql databases (mainly because of the OS licensing). So, you will have to make sure that the postgresql database instance is up and running on your machine. You can modify the connection string to include your username and password. Connections strings can be found at src/Host/Configurations/database.json and src/Host/Configurations/hangfire.json. Once that's done, let's start up the API server.

```
make start
```

That's it, the application would connect to the defined postgresql database and start creating tables, and seed required data.

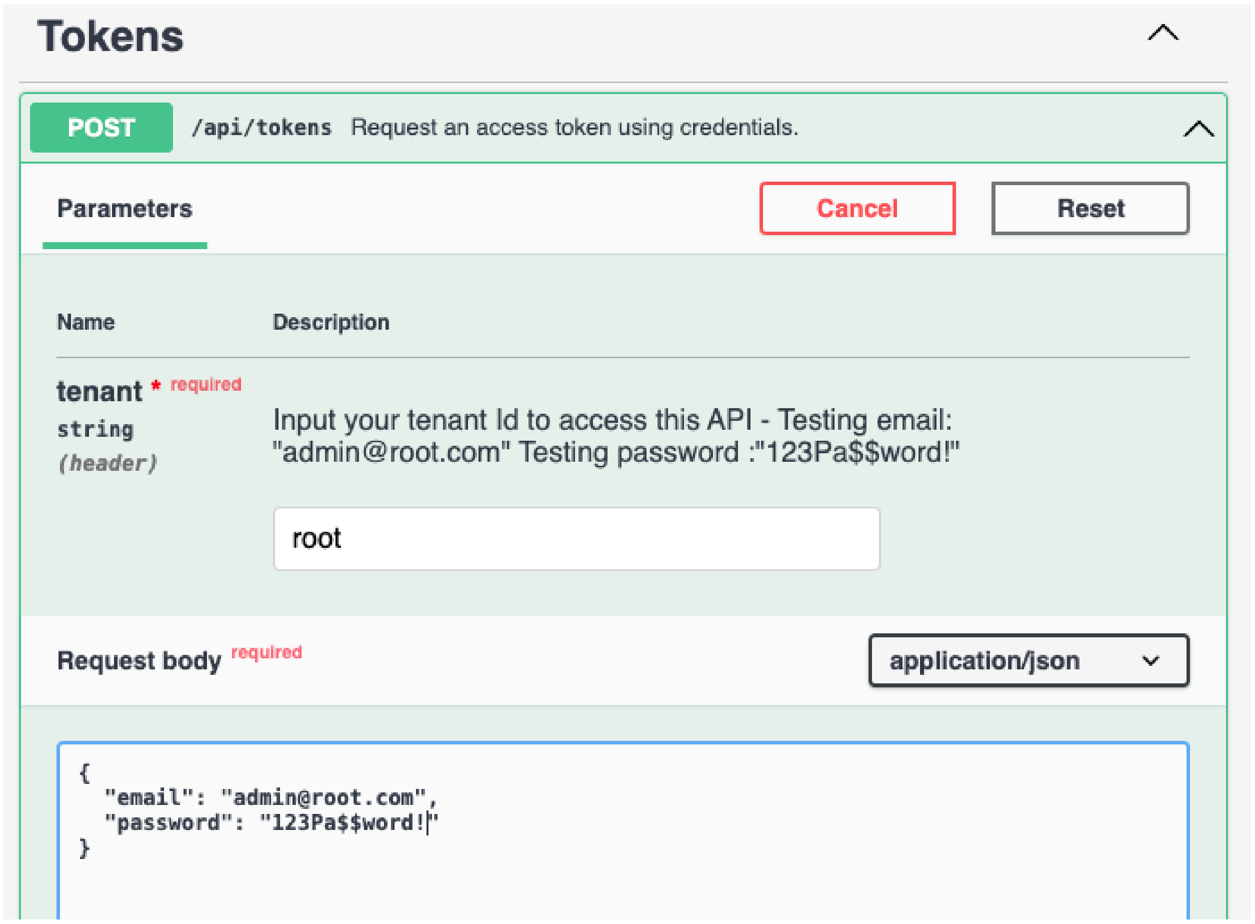
For testing this API, we have 2 options.

- Swagger @ localhost:5001/swagger
- Postman collections are available ./postman

Credentials & Tokens

```
{
    "email":"admin@root.com",
    "password":"123Pa$$word!"
}
```

The default credentials to this API are:



Request an access token using credentials.

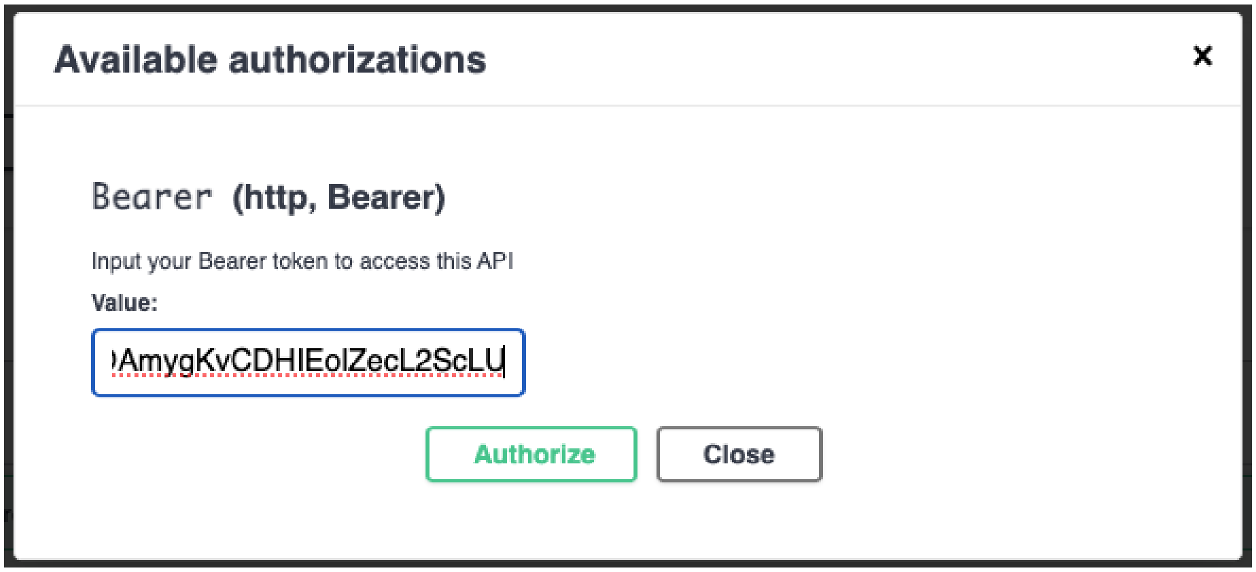
tenant * required
string
(header)

Input your tenant Id to access this API - Testing email: "admin@root.com" Testing password :"123Pa$$word!"

root

```
{
  "email": "admin@root.com",
  "password": "123Pa$$word!"
}
```

The token must be added to “Available authorizations” in Swagger. Via REST API, the token must be put into the request header.



Available authorizations

Bearer (http, Bearer)

Input your Bearer token to access this API

Value:

AmygKvCDHIEolZecL2ScLU

Record of Changes

Table 1 - Record of Changes

Version Number	Date	Author/Owner	Description of Change
1.0	*03/28/2024*	*Angel Bustamante*	*Initial Version*