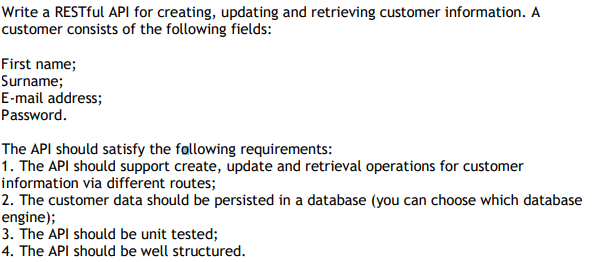
Task 1



Task 1 was built based on .Net Core 3.1 Framework. It was created the solution GuilhermeRocha, with 3 projects:

* GuilhermeRocha.API, ASP.NET Core Web App MVC, responsible for handling the API calls
* GuilhermeRocha.Infrastructure, ASP.NET Core Class Library, responsible for database access
* GuilhermeRocha.Test, ASP.NET Core xUnit Test Project

GuilhermeRocha.API is separated by Controller and BusinessLogic folders. Controller receives the actions and validates arguments (if there are any). After that, it invokes the corresponding Business Object.

BusinessLogic folder on the other hand, it’s responsible for accessing/operating through the application’s context entities and returning a result to the Controller.

GuilhermeRocha.Infrastructure it’s where the actual database access is. It uses Entity Framework Core, as well as Code first migrations, saved on Migrations folder. It connects to a local database for simplicity in the tasks required.

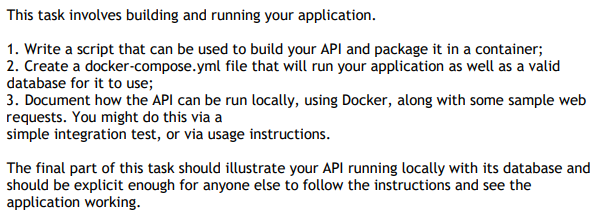
This project itself is independent. API depends on it for querying the database.

GuilhermeRocha.Test was the most challenging for me, since I haven’t done actual unit tests for quite a while now. I’ve decided to test the Actions themselves. I could also opt for the BO route, but I thought this was the more appropriate route, since this way BO and Controller layers were tested.

For simplicity, Swagger was algo installed in order to make it easy for the API to be tested.

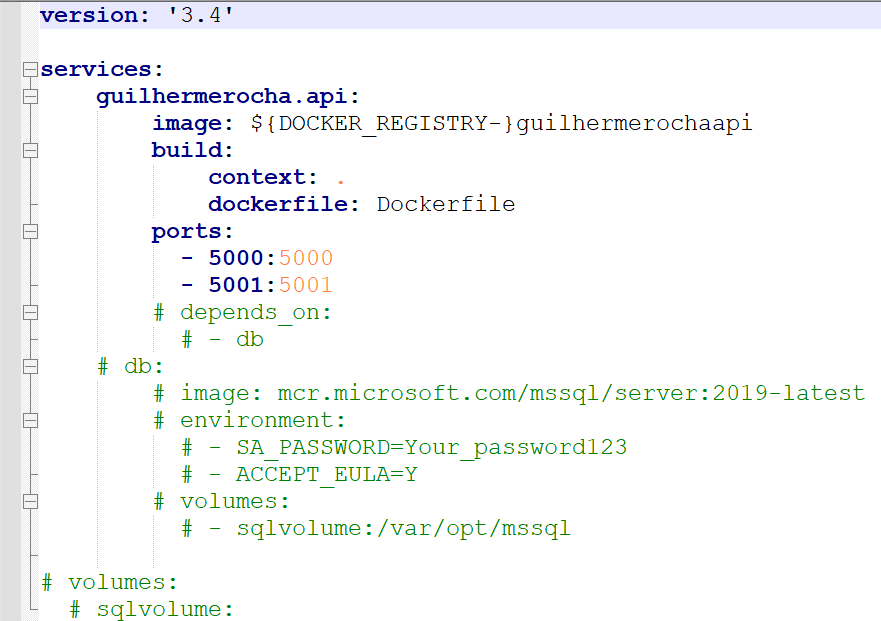
As you can also see, the solution was object oriented, in this case for the User object.

PriorityOrderer is a class that I got from the internet only used for running test by order.

Task 2

By default, when creating a .NET project, it’s possible to use “Enable Docker Support”, and that was what I did to have the basic Dockerfile composition.

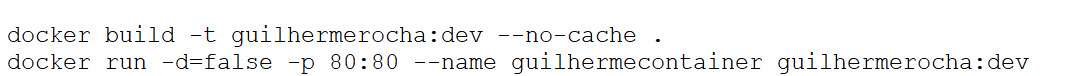
I’ve added the 3 projects and copied them to the docker container.

Docker-compose file was the most tricky part that I’ve done.

As you can see, there is quite a bit of code commented here. This happened because I did not managed to figure out the proper way to make the GuilhermeRocha.API container depend of the DB container. In my Dev projects, there was always a DevOps team responsible for deploying solutions, so I never had a full time responsibility of deployment. (As mentioned in the interview)

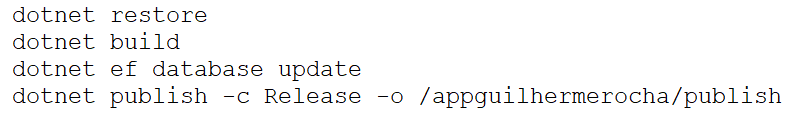
This is a must have to enable that when deploying the containers, the Infrastructure project is able to access the database.

With that being said, it is not possible for the solution to be run on Docker with good data from the DB.

Regardless of that, these would be my go-to commands to run after docker-compose is configured:

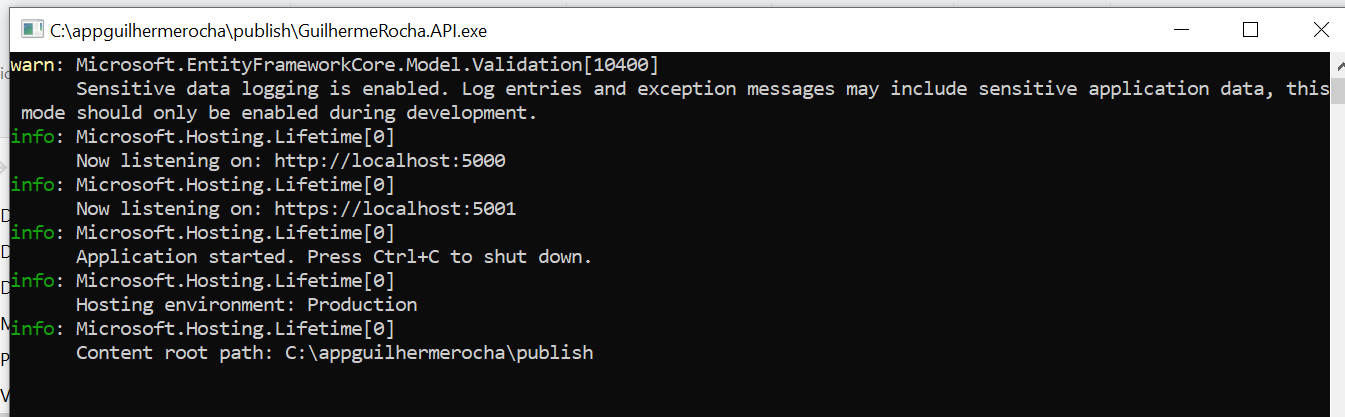
Docker build, with tag and –no-cache, meaning that the build of the container (image) will always start from scratch, with the desired tag.

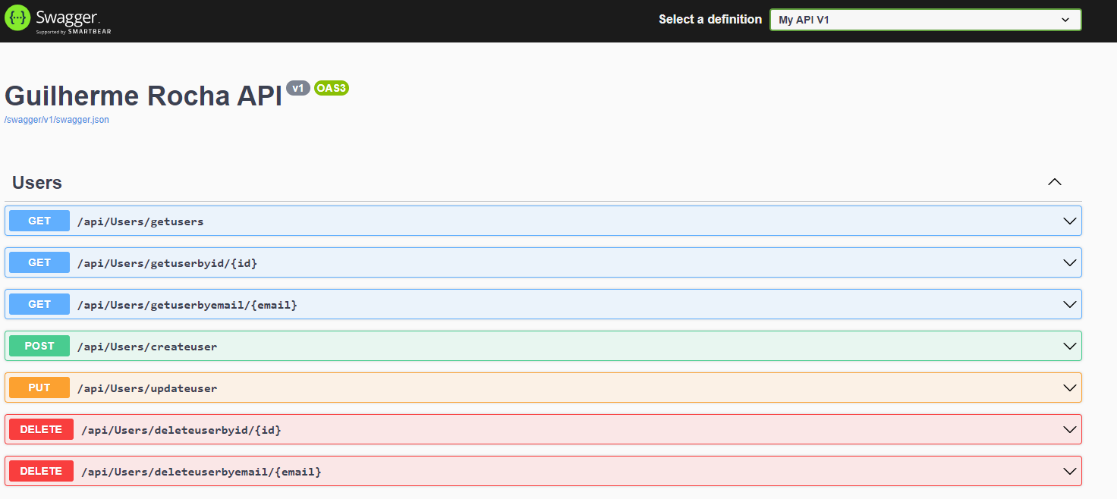
Docker run, with port and name of the previous image created.

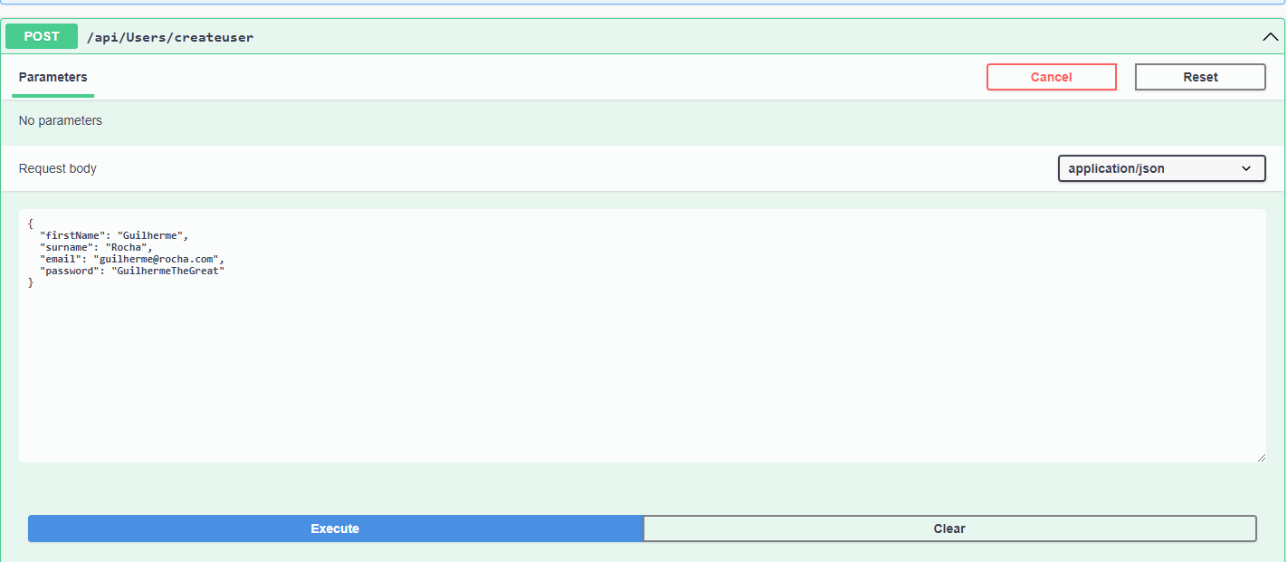
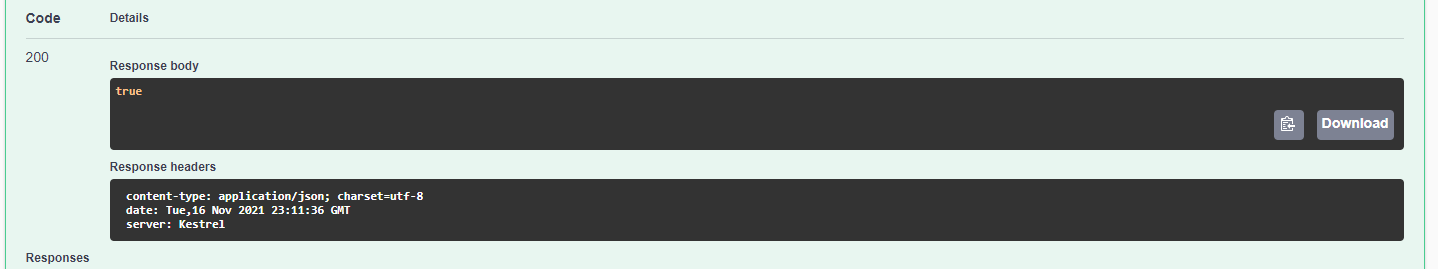
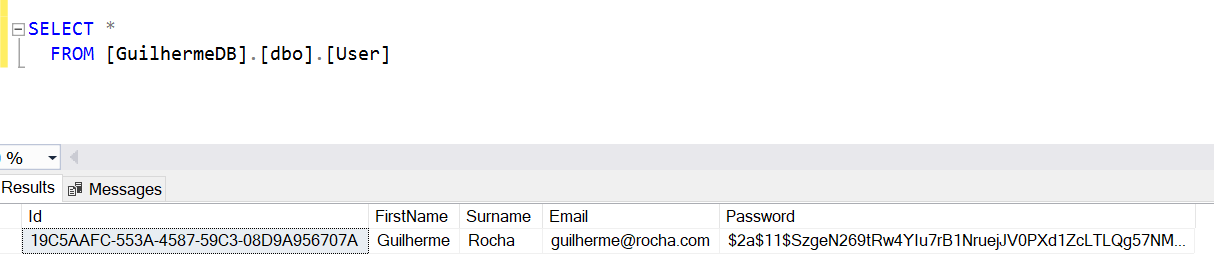
One simpler way to visualize the application running, is using .NET CLI, dotnet.

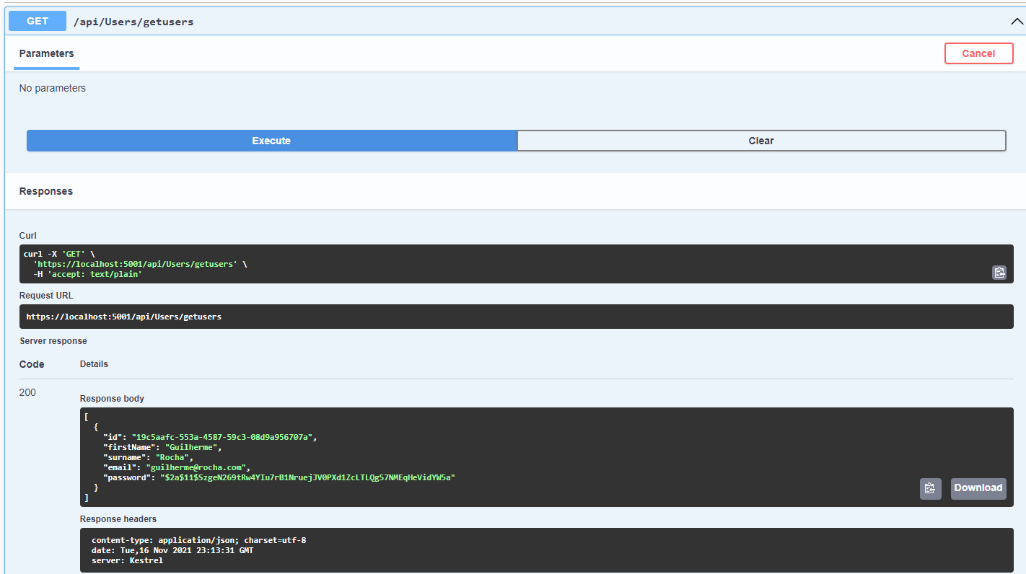
“restore” is used to restore the dependent libraries. “build” for building the solution. “ef database update” to ensure that the database is updated. (BD needs to be created firstly of running migrations).

“publish” created the published application folder from where the application can be run in any machine. (with framework installed)

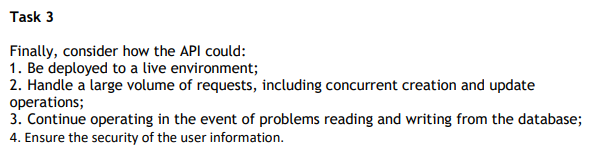
After that is run, a folder will be create on C:/ and it can be run. (GuilhermeRocha.API.exe)



Here we can test some of the CRUD operations created for the user:

With proof of a working application.

Task 3



1 – To deploy this application to a live environment, most of the work is already done. It’s only a matter of having a VM with dotnet and Docker Tools (for example) and run the containers there.

One important issue in my opinion is the http vs https. There will be necessary to create a certificate and add it to the dockerfile (“COPY XXXX.pfx ./XXXX.pfx).

2 – For me, there are 2 different problems in the same scenario: getting information from the database, or updating/creating information.

Let’s imagine that the /Users/getusers action retrieves 1 milion users. Depending of the situation, but lets say that this request is used for a front end application, this action could be modified with some parameters like “itemsperpage” and “pagenumber”, and retrieve only a page with 100 users.

For updating/creating, one of the most easy tasks to do, is to separate the query in smaller ones. Instead of a INSERT INTO… (1 milion users), there could be 1000 insert’s with 1000 users each.

For concurrent creation, Microsoft offers some threading solutions to waiting withing processes like SemaphoreSlim.

3 – I think the best approach for continuous operation regarding database status would be some sort of microservice/queuing implementation. The API could send a message to other microservice that does the actual CRUD operations in the database. That way requests could be queued until the DB is up again.

4 – Regarding security, and using the example of this actual GuilhermeRocha.API. Right now, any user can access any action, regarding of being or not his own.

There could be added a Login/Logout action. Login receives the user email + password, and returns a token with a validation date (60 min for example). All the other requests will require the Bearer token authorization (Authorize attribute). There could be a Token table in the database, that saves userID, token itself and expiration date. This way, when a user authenticated tries to update a user that’s not him, the token will not match user’s email and he will not be authorized to continue. This way authorization and authentication are ensured. There could also be a service that run every 60 min and deletes from DB expired tokens to ensure the DB size keeps at a minimum.

I’ve spent about 10-12 hours developing and answering these tasks. Half of the time was around Task 2, more precisely in the database container.

I hope that I’ve answered everything in the best way possible.

If any more tasks are needed feel free to contact me.

Guilherme Rocha