**Readme**

**Instructions to run the Application**

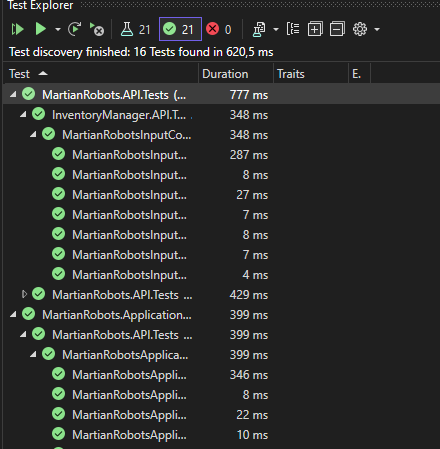
**With Docker:**

You can use Docker running the API project from VS with Docker

**With IIS:**

1. For the API project we only press run or Ctrl + F5 in Visual Studio for launch the IIS Express with the asp.net core.
2. For try de API Services you must use basic authentication with user “Admin” and password “Admin”, you also can use the PostMan collection inside de API project for try this.

For the tests we can run from Test Menu/Run all tests like the screenshot.



2) For the Console Project, is a basic console project that consume the API for solve the problem.

The input can be from console read params or from a Json file in the root of the console project. By default is configured for Json file, but you can change the dependency for the ConsoleInputService if you want.

When you run the console app you must have running the API project before.

If you use Docker or IIS for example, the port is different, you have to change the apiPort constant in the APIService class to the required port.

This project makes use of the 3 defined controllers, the one that is in charge of the CRUD operations of the inputs, the one of the outputs and the one that solves the problem. For each user input/json it is saved in a database in the MongoDB cloud and the same for the exercise outputs.

**Documentation**

I implemented my solution using Asp.Net Core for the backend using a DDD paradigm. I used **DDD** for the design of the backend project, because it is more maintainable and reusable, which allows us to create a project with good practices that is more scalable if we need to continue adding functionalities, separating each part of the project by layers to separate responsibilities (Presentation Layer, Application Layer, Domain Layer and Infrastructure Layer) and dependencies. I also implemented the **CQRS pattern** for Command and Query segregation using the **MediatR** nugget package for the MartianRobotsController(Solve the problem) and MartianRobotsInputController(inputs), and MartianRobotsOutputController(outputs) is implemented with a Repository class.

In addition I implemented validation for input method parameters **(Fluent validation),** traces with **ILogger** in controller’s methods, **Basic security** using the APIBasicAuthenticationHandler class and **Swagger** package for API documentation.

Also, the project is adapted according to the SOLID recommendations and works with the Repository pattern if we need to add more data model repositories in a very simple way without duplicating code.

For the tests i have used **NUnit** for Controller Tests **with full coverage** and **PostMan** for creating a collection of requests.

For solve the problems i have implemented a Command pattern allowing us to define new commands in addition to those required by the problem. Basically a command allows us to execute an action and be able to undo it.

**Brief description of assumptions or not implemented requirements**

**Unit Tests:**

For this example, due to lack of time, I only implemented the unit tests for the API controller and Application layer, the unit tests for the rest of the layers were missing.