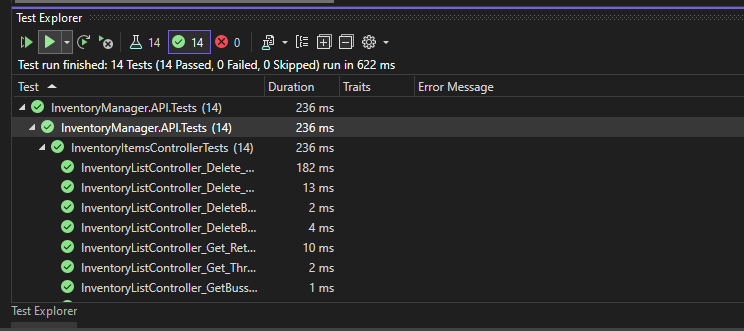
**Readme**

**Instructions to run the Application**

1. For the backend project we only press run or Ctrl + F5 in Visual Studio for launch the IIS Express with the asp.net core.
2. Also you must have an instance of RabbitMQ server on <http://localhost:15672/> using the user “guest” and password “guest”. For install this, you must install first this link :<https://www.erlang.org/downloads> and after RabbitMQ: <https://www.rabbitmq.com/download.html>, after that you must run the following command inside the installation path: rabbitmq-plugins enable rabbitmq\_management
3. 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 Frontend Project, is an angular project, we can run from VS Code and in the terminal pointing to the project path and we must execute **npm start** or **ng serve --proxy-config proxy.conf.json.**

Before executing it we must restore the npm packages, for this we will execute the command "npm install" in the path of the installation folder "...InventoryManager.Frontend/" and then do it again in the folder "...InventoryManager.Frontend \Inventory.Frontend"

**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. In ConfigureServices Method in Startup.cs we can change the IInventoryAppService Injection from InventoryCQRSAppService to InventoryAppService for use instead of CQRS pattern a typical Repository pattern

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.

I also added RabbitMQ as a message broker for handling events/messages using the MassTransit library, this allows us to send events asynchronously

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 test I used **NUnit** for Controller Tests **with full coverage** and **PostMan** for creating a collection of requests.



For the SPA project I used Angular with Bootstrap and for the test i used Jasmine.

The Angular SPA project also uses Clean Code best practices to organize the project and separate responsibilities between application components, also applying SOLID principles.

**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, the unit tests for the rest of the layers were missing.

**Docker**:

After trying to use Docker in the end I did not have time to achieve it due to a problem with RabbitMQ docker-compose integration. Is posible execute the container of the project without the RabbitMq dependency following the next steps:

-Comment the MassTransi region for desactivate RabbitMq

- Using the FakeEventManager instead of CustomEventManager in the builder.Services.AddScoped for dependency injection.

**RabbitMQ consumer message**

I didn't have time but I could have implemented a microservice that would consume the ItemRemoved and ItemExpired events. This would be possible by implementing within the microservice, for example for the ItemRemoved event, something like this:

