**Currency Exchanger**

**Docker Image:**

docker file and docker-compose.yml was added to build an image and run it on docker to run the application run from the app directory that contains the csproj file the following command

to create database Image

docker run -e 'ACCEPT\_EULA=Y' --name mssql -e 'SA\_PASSWORD=A-a\_7276P@a$$wRD!' -p 14330:1433 -d mcr.microsoft.com/mssql/server:2019-latest

and then connect to tcp:localhost,14330 with user name: sa and password: A-a\_7276P@a$$wRD! and execute the DB script in in sql manager

Then create app image

docker build -t currencyexchangerservice .

and finally run the Image

docker run -it --rm -p 3000:80 --name currencyexchangercontainer currencyexchangerservice

in case of running the service on docker after those commands you can open the app on [http://localhost:3000](http://localhost:3000/)

in case of running the service from IDE you can open the app from [http://localhost:54661](http://localhost:54661/swagger)

to view from swagger or us postman

**NOTE:** it will take some time to build in case of using docker

**Logging:**

SeriLog Added to the project and logging will be written to console and file in Logs folder in case running the project from IDE the folder will be auto generated in the application directory in case of running the app through docker the file can be seen from the docker terminal

**Database:**

**if the app will run from IDE uncomment IDE connection string**

**if the app will run from Docker uncomment Docker connection string**

database where provided in 2 ways :

1- through migration and for auto create the database and tables just uncomment the auto migration region in program.cs

2- scripts for creating the database and the table in SetUpDB.sql it can be run manually to create the database in case of changing the database name, the new name will be added to 3 places (appsettings, testserviceprovider in test project, in the sql script)

**XUnit Test:**

the unit test was created for testing the caching flow and main functions required in the currency exchanger APIs like adding trade and user passing the limit

**PostMan:**

postman collection and environment added in the project to help testing the APIs and the scenarios behind them like fetching the API from the Integrated api and then make sure it was saved in the cache, the collection contains the required requests and test cases for them, the collection and the environment will just need to be added to postman and run the collection to check the test cases are passed environment has 2 parameters (API keys for the integration, and base) in case of using own API key or change the URI port the variable in the Environment must be uodated