Practical Project Evaluation 1 (Close Scope)

These evaluation goals validating the technical knowledge on theorical and practical aspects. First of all, it consists to understand the provided business and technical requirements for a sample application and, in order to build it, the developer must apply the main technologies and concepts commons in ***Inatel*** project´s environment.

1.Instalation:

The following softwares are mandatory to be installed:

Mandatory Software	Link to download
JDK 11	https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html
Maven	https://maven.apache.org/download.cgi
Docker Engine	https://docs.docker.com/docker-for-windows/install/

The following softwares are optional and can be replaced by others for the purpose:

Optional software	Link do download
Eclipse IDE for Enterprise Java and Web Developers	https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2022-03/R/eclipse-jee-2022-03-R-win32-x86_64.zip
Postman	https://www.getpostman.com/downloads/

2.General information

Internet can be used with no restriction.

The project with all the source code produced must be delivered at the end of the test (further details in *Deliverables* section).

3.Configuration

The application to be developed will use two services that run on docker containers.

You will need two images to run the services below which also will be consumed by application to be developed (more information will come later).

- MySQL: https://hub.docker.com/_/mysql

- stock-manager: https://hub.docker.com/r/lucasvilela/stock-manager

The first service is a MySQL database service. To start it, run the following command line at Windows Power Shell.

```
docker container run -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=bootdb -p 3306:3306 -d mysql:8
```

The second service is a REST application that will be consumed by application to be developed (more information will come later). To start it, run the following command line at Windows Power Shell.

```
docker container run -p 8080:8080 -d lucasvilela/stock-manager
```

Any docker image can be stopped and re-started at any point in time. Any data stored on it is wiped on this process.

To stop a specific container, first execute [`docker ps`] to get container's `id`. Then, execute [`docker stop {id}`] to stop the specific container.

4.Quotation management application

The following sections describes the needed information to create the application from scratch.

Each section contains a new requirement. It is recommended to implement the application in the sequence the sections are presented.

The application should be developed using Maven. Use the most suitable low-level architecture for the given problem.

4.1.Store and read stock quotes

The application to be developed, called **quotation-management**, is a REST based application whose purpose is to **store stock quotes from stock market**.

For example, a user wants to register that stock **PETR4** quotation at **01/12/2018** was **$35.00**.

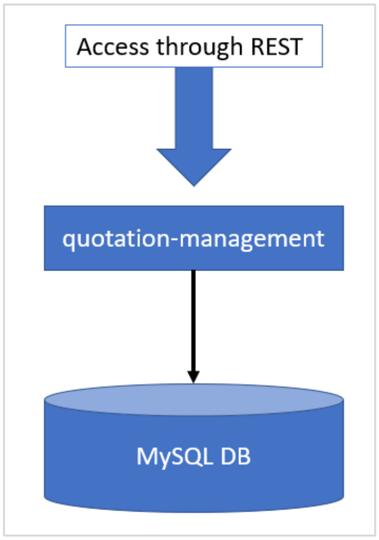
A user can register as many quotes as he wants for the same stock. Also, a user can register quotes from different stocks.

Finally, a user can read stock quotes registered.

To store and read stock quotes, application should expose REST APIs through port **8081**.

The stock quotes should be stored at MySQL DB running on docker container, described in previous section.

The following picture illustrates the application?



Quotation-management application should be developed using Spring Boot.

The **payload for creating or reading a quote** should follow the depicted example.

```
{
        "id": "c01cede4-cd45-11eb-b8bc-0242ac130003",
        "stockCode": "PETR3",
        "quotes":
        {
           "2019-01-01": "10",
           "2019-01-02": "11",
           "2019-01-03": "14"
        }
}
```

Note: The `id` attribute generation must be performed by UUID

In total, the following operations should be exposed at quotation service:

- Create a Stock Quote
- Read a Stock Quotes by `stockCode`
- Read all Stock Quotes

4.2.Validation a stock before processing

A new service is introduced to the solution in which the application being developed is part of.

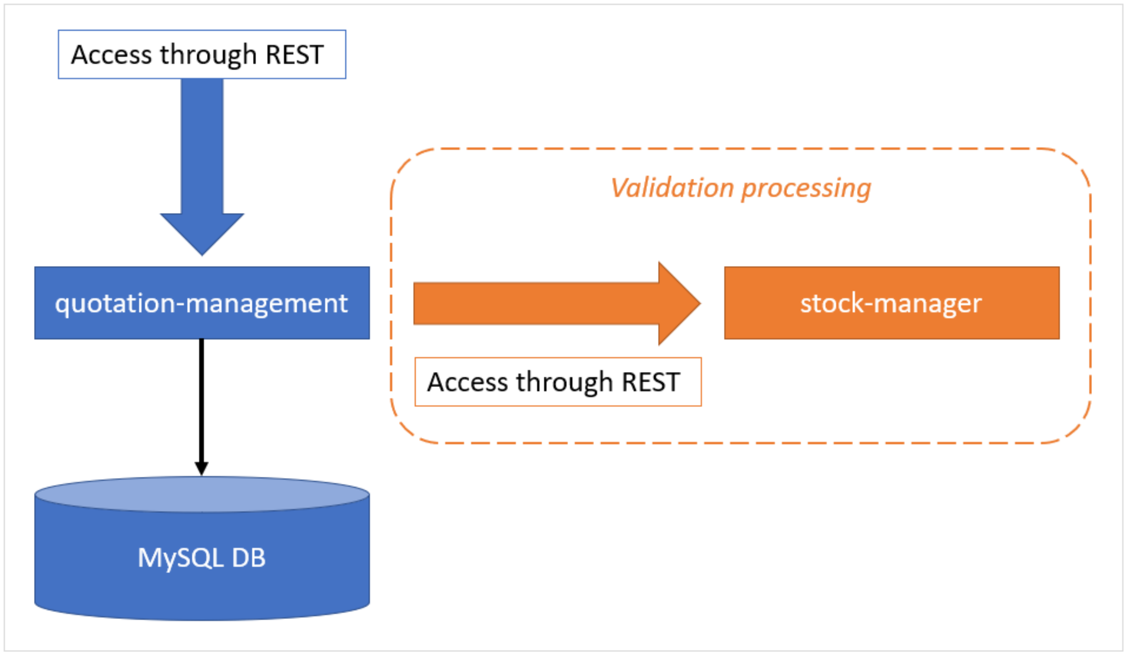
The service is called **stock-manager**. It is responsible for maintaining a list of stocks that can have quotes stored at quotation-management.

In other words, a user is allowed to create stock quotes on quotation-management only if the stock is registered at stock-manager.

Hence, quotation-management should be modified to verify, before creating a stock quote, if stock received is registered at stock-manager.

In case it is not, an error should be returned at create quotation REST API.

The following picture illustrates the application:



Stock-manager service runs on a docker container on port 8080, as described in previous section.

As mentioned in the picture, stock-manager service exposes REST APIs to fulfill its needs.

Any outside user can register stocks on stock-manager.

Quotation-management just reads registered stocks to validate the stock quote create operation.

❖ GET `http://localhost:8080/stock`

Get the list of stocks registered. By default, when server is started, stocks PETR3 and VALE5 are already registered.

- POST http://localhost:8080/stock

  Register new stock. A same expected JSON is:

```
{
	"code": "petr7",
	"description": "test petr"
}
```

4.3.Caching registered stock quotes

To avoid accessing stock-manger service at each create stock quote operation, quotation-management application should cache locally (in memory) the registered stocks.

When validating a stock, whenever the cache is empty, it should populate it from stock- manager service.

If cache is not empty, it should validate with cached data.

The cache should always be cleaned when a new stock is registered at stock manager.

To achieve this, two new functionalities should be introduced.

Quotation-management should register itself at stock-manager during startup. To register itself, the following stock-manager service must be invoked.

- POST http://localhost:8080/notification

The body of the service should follow the below example. It should contain the host and the port of the application being registered (in our case, quotation-management).

```
{
	"host": "localhost",
	"port": 8081
}
```

Whenever stock-manager receives a request to register a new stock, it tries to invoke a notification service at all registered applications.

Hence, quotation-management should expose a new REST service to accept notifications from stock-manager.

Stock-manager expects the following URL.

- DELETE http://[host]:[port]/stockcache

  Stock-manager replaces [host] and [port] by the proper data from registered applications.

Quotation-management, upon reception of this notification, should clean the cache.

4.4.Testing

Manual tests can be done using Postman.

All developed classes and their methods must have unit tests. Except the *getters* and *setters* methods since they are not considered business methods. Preferably use **JUnit** to write the tests.

Create some functional tests for the developed requirements (remember to explore positive and negative test scenarios).

The creation of integration tests is optional, but nice to have.

4.5.Containerization

Create a docker image for quotation-management, so that it can run in a container.

5.Deliverables

The final solution project has to be delivered as a GitHub repository link.

Such project must have the following items:

- ❖ The project must be structured as Maven project and so will be able to be compiled, tested, built and packaged by command line (`mvnw`);
- ❖ The unit tests covering all business methods and classes;
- ❖ The online API documentation based on Swagger and following the OpenAPI 3 Specification. It will be accepted the automation documentation generation tools like Springfox. Be aware in make all configuration. If you prefer, you can deliver the API documentation as a static PDF file, by informing the file´s path;
- ❖ The developer is authorized and encouraged to propose algorithms, utility classes or "out-of-the-box" architectures as long as they represent real contributions to the final solution.

6.Evaluation Criteria

The following items will be considered in evaluations process:

1) Code organizations: software layers, variable naming, methods naming, naming standardization.
2) Assertively: How presented solutions is aligned to the requirements? Everything is working well?
3) Automation tests: Integration, functional and unit tests coverage.
4) Documentation: API REST documentation; Java docs (main classes and methods)

The final result will be the average point for each above item.

# Good luck #