**Payconiq Technical Challenge:**

**Spring Boot on Local connecting to MySQL Docker container**

**1.Use MySQL Image published by Docker Hub (https://hub.docker.com/\_/mysql/) Command to run the mysql container**

docker run --detach --env MYSQL\_ROOT\_PASSWORD=root --env MYSQL\_USER=admin --env MYSQL\_PASSWORD=admin --env MYSQL\_DATABASE=stockRepository --name mysql --publish 3306:3306 mysql:5.7

2.**In the Spring Boot Application, application.properties setup for connecting with docker**

Since I was using docker toolbox(Windows 10 Home) , i was not able to bind sql hosted in docker with local host, I was only able to bind only with docker toolbox ipAddress

Please change 'spring.datasource.url' based on below case

Case 1: If you are using docker desktop

spring.datasource.url = jdbc:mysql://localhost:3306/stockRepository?createDatabaseIfNotExist=true

Case 2: If we are using docker toolbox

spring.datasource.url = jdbc:mysql://"ip Adress of docker machine":3306/stockRepositorycreateDatabaseIfNotExist=true

command to find docker Ip Adress: "docker-machine ip"

**3. Test Case in Application:**

All test case are using database connection. Please make sure to start Mysql server before building the application.

**4. OpenApi Implementation:**

OpenApi has been implemented in this project.

OpenApi endpoint List:

**Api Document Endpoint**: <http://localhost:8080/stock-api-docs>

**Api Swagger Endpoin**t : <http://localhost:8080/stock-api-swagger.html>

Controller Info:

1. **Get All Stocks with Pagination(**[**http://localhost:8080/api/stocks**](http://localhost:8080/api/stocks)**)**

* This endpoint fetches all the stocks in database.
* Above endpoint is configured with pagination.
* It takes Pageable object as optional input.
* Pagination size is configurable with Pageable object.
* Default size for pagination is 20.
* Test data for custom implementation

{

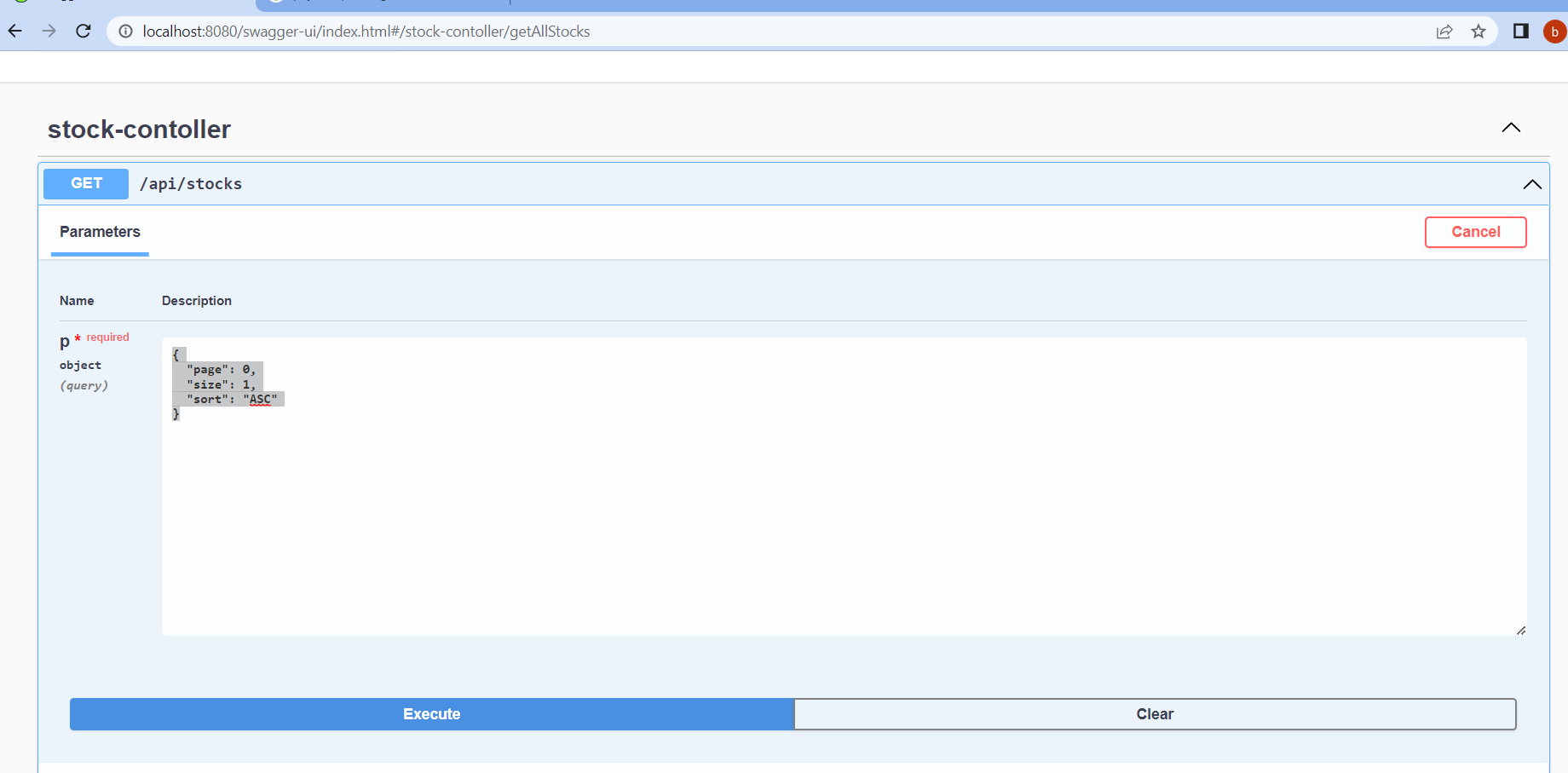
"page": 0,

"size": 1,

"sort": "ASC"

}

* Pass above value as request body.



1. **Get Stock based on id (**[**http://localhost:8080/api/stocks/{id}**](http://localhost:8080/api/stocks/%7bid%7d)**)**

Above endpoint takes id of stocks as @PathVariable.

Case 1 : if given id is present in database, will return stock.

**{**

**"id": 1001,**

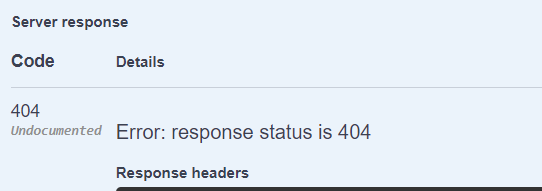
**"name": "Lic",**

**"currentPrice": 234,**

**"lastUpdate": "2022-05-10T02:41:19.000+00:00"**

**}**

Case 2: if given id is not present in database, will return null along with HttpStatus code :404



1. **Create Stock (**[**http://localhost:8080/api/stocks**](http://localhost:8080/api/stocks)**) Post Method**

Above endpoint creates the stock,

Have to provide StockDto object for creation,

{

"id": 0,

"name": "Test",

"currentPrice": 1235,

"lastUpdate": "2022-05-10T08:32:54.506Z"

}

**id and lastUpdated params are optional.**

**name is (@NotNull)**

**if currentPrice is not given will be updated to integer default value**

1. **Update the available stock (**[**http://localhost:8080/api/stocks/{id}/{price}**](http://localhost:8080/api/stocks/%7bid%7d/%7bprice%7d)**) Patch Method**

**A**bove endpoint Updates the stock present in database.

**Case 1** : if given id is present in database, it will update the values and return the id of the stock updated message.

Request:

http://localhost:8080/api/stocks/1001/2324

Response:

**id 1012 has been updated**

**Case 2** : if given id is not present in database, it return id not found message.

Request:

http://localhost:8080/api/stocks/1/2324

Response:

**id 1014 value not found in database**

5, Delete the Stock ([http://localhost:8080/api/stocks/{id}](http://localhost:8080/api/stocks/%7bid%7d)) DELETE

Case 1: if given id is present in database, it will delete the stock and return 204 status code.

Case 2: if given id is not present in database. It will return 404 status code.