University of Houston

COSC 6376 Cloud Computing

Weidong Shi

Final Project

Stock History Web App

Ashutosh Kumar
1854393

1 Introduction

For the Final Project for the Cloud Computing class, I have made a Web App to look at some historical data for US stocks. The frontend for this app has been made using Vue.js, which is a JavaScript framework. The backend has been completed using Amazon Web Services (AWS). To access historical data for US stocks, I use Polygon.io API.

There are two main functions that are required for this web app. The first is to add the stock to the database as well as to retrieve the information regarding the same from the 3rd party source. And the second is to remove some stock. These two functions are implemented in both, frontend and backend.

In the frontend, these functions do the addition and the removal of the stocks from the webpage, which the user can view and with which the user can interact. In the backend, there is an additional function, which retrieve all the details from the database and sends it to the frontend, whenever requested. All the implementation details are given in the following sections.

2 Frontend – Vue.js

Since this project is for a Cloud Computing class, the frontend for this web app is kept simple. I used Vue.js framework to create the frontend.

As mentioned above, we have two functions in the frontend – `AddStock` & `RemoveStock`. These two functions do exactly what their names suggest – they add and remove the stocks, respectively. Let us look at these two methods in some detail.

2.1 `AddStock` Function

This function reads the user input – the name of the stock and the date – and it sends this information to AWS through the AWS API Gateway. AWS Lambda functions do the required processing and send the data for the requested stock and for the requested date back, which is then displayed to the user.

2.2 `RemoveStock` Function

This function detects when the user clicks the 'cross' button and then sends a `DELETE` request through the AWS API Gateway to the backend, where the stock is deleted by AWS Lambda. This stock is also deleted from the frontend, so the user does not see it anymore.

2.3 Some Images of the frontend

The following image 1 shows what the frontend looks like before any stocks are added.

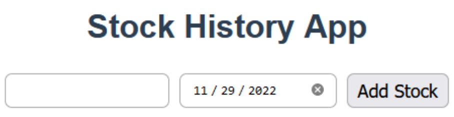
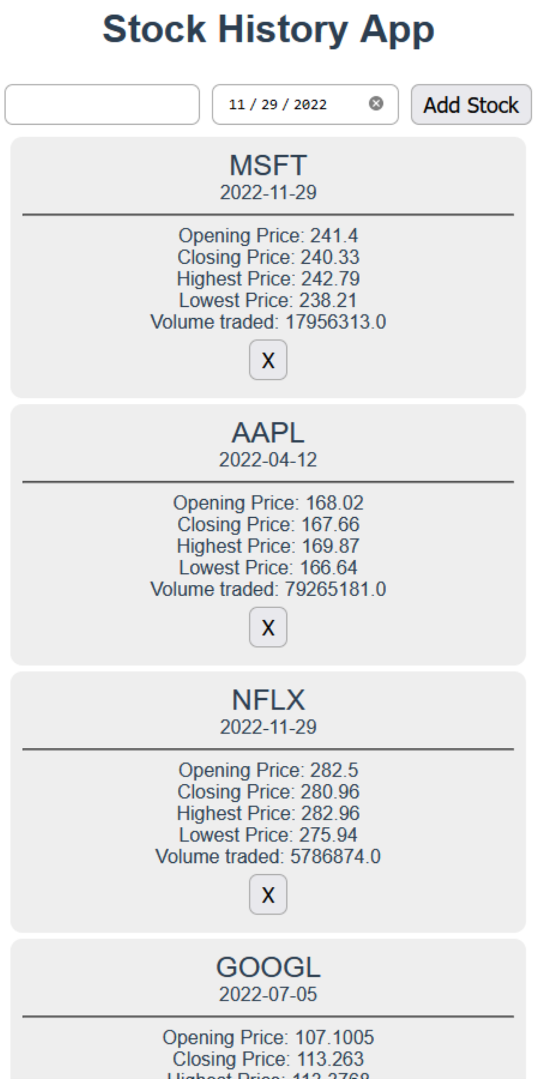


Figure 1: Frontend before any stocks are added.

The following image 2 shows what the web app looks like after some stocks are added to it.


Stock History App

11 / 29 / 2022 Add Stock

MSFT
2022-11-29
Opening Price: 241.4
Closing Price: 240.33
Highest Price: 242.79
Lowest Price: 238.21
Volume traded: 17956313.0
X

AAPL
2022-04-12
Opening Price: 168.02
Closing Price: 167.66
Highest Price: 169.87
Lowest Price: 166.64
Volume traded: 79265181.0
X

NFLX
2022-11-29
Opening Price: 282.5
Closing Price: 280.96
Highest Price: 282.96
Lowest Price: 275.94
Volume traded: 5786874.0
X

GOOGL
2022-07-05
Opening Price: 107.1005
Closing Price: 113.263

Figure 2: Frontend before any stocks are added.

3 Backend – AWS

The backend is the main part of this project. The backend has been implemented in AWS. As mentioned above, there are three functions implemented here – each in AWS Lambda using Python language. One retrieves the stock data from the 3rd party API and stores it in AWS DynamoDB. The other removes a stock. And the last one retrieves all the stored information regarding the stocks from the database, whenever the frontend requests so.

3.1 Implementing Functionalities – AWS Lambda Functions

To implement all the functionalities, I created three Lambda Functions.

3.1.1 `AddStockFunction`

This function gets the stock ID, stock name as well as the request date from the frontend through AWS API Gateway. The frontend uses the `POST` method to send this information through the API Gateway. Then, Polygon.io's API is triggered with the stock name and the date, which sends the raw data. This data is stored in the database using the `put_item` method of the library `boto3`. This can be seen in the image 3.

3.1.2 `RemoveStockFunction`

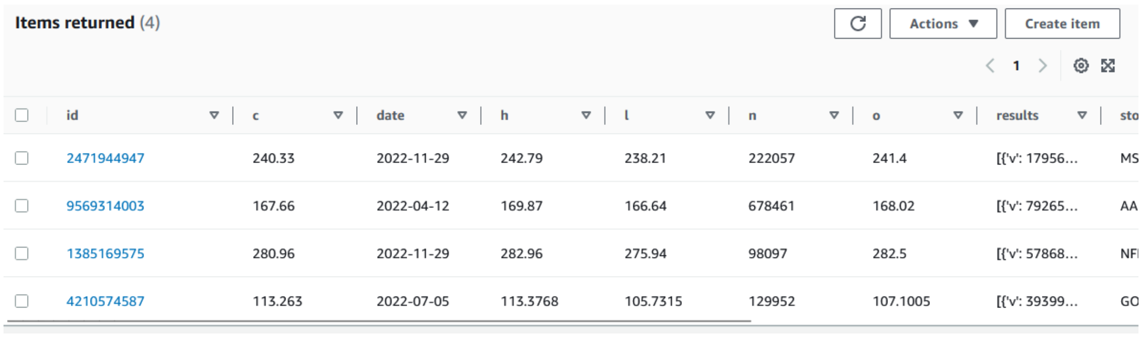
This function gets the ID of the stock from the frontend through the API Gateway. Using the method `delete_item`, the row is deleted from the DynamoDB table. The frontend uses the `DELETE` method to request the deletion through the API Gateway.

3.1.3 `GetStockFunction`

This function uses the `scan` method of the `boto3` library to get all the rows from the database, which are then 'returned'. The frontend uses the API Gateway to request this information using the `GET` method.

3.2 Database – AWS DynamoDB

I use AWS DynamoDB to store the data. I created a table called `stocks-project`, which uses a numerical ID as the 'partition key'. The Lambda Function `AddStockFunction` gets the information from the 3rd party API and adds it to this table. This table, along with a few entries, can be seen below 3.

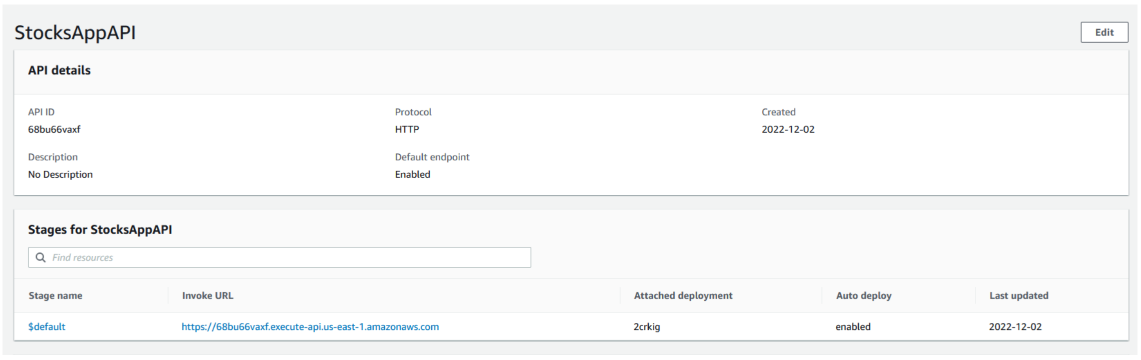


id	c	date	h	l	n	o	results	sto
2471944947	240.33	2022-11-29	242.79	238.21	222057	241.4	[{'v': 17956...	MS
9569314003	167.66	2022-04-12	169.87	166.64	678461	168.02	[{'v': 79265...	AA
1385169575	280.96	2022-11-29	282.96	275.94	98097	282.5	[{'v': 57868...	NFl
4210574587	113.263	2022-07-05	113.3768	105.7315	129952	107.1005	[{'v': 39399...	GO

Figure 3: AWS DynamoDB Table

3.3 Accessing the Backend – AWS API Gateway

The frontend needs to be able to send and receive data to and from the backend. This is done through AWS API Gateway. I created a `HTTP` API. This can be seen in the image below 4.



API details

API ID 68bu66vaxf	Protocol HTTP	Created 2022-12-02
Description No Description	Default endpoint Enabled	

Stages for StocksAppAPI

Stage name	Invoke URL	Attached deployment	Auto deploy	Last updated
$default	https://68bu66vaxf.execute-api.us-east-1.amazonaws.com	2crkig	enabled	2022-12-02

Figure 4: `HTTP` API

In this API, I created three routes – `POST`, `DELETE`, and `GET`. When the frontend uses the `POST` method, the Lambda function `AddStockFunction` gets triggered. When the `DELETE` method is used, `RemoveStockFunction` is triggered. And when the `GET` method is used, `GetStockFunction` is triggered. These three methods can be seen in the image below 5.

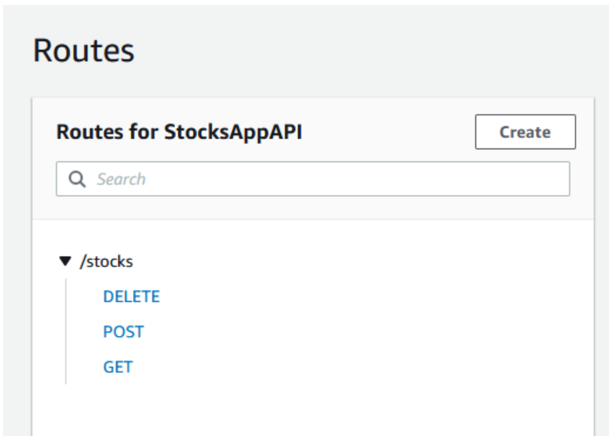
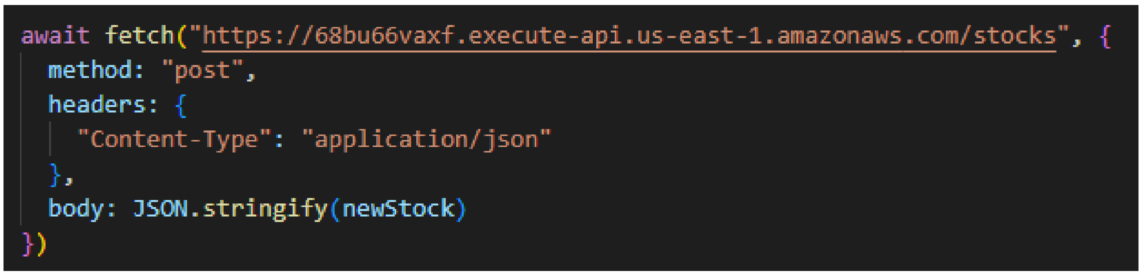


Figure 5: API Routes

The way the frontend adds a stock can be seen in the image below 6.



```
await fetch("https://68bu66vaxf.execute-api.us-east-1.amazonaws.com/stocks", {
  method: "post",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(newStock)
})
```

Figure 6: Code used in the frontend to add a stock.