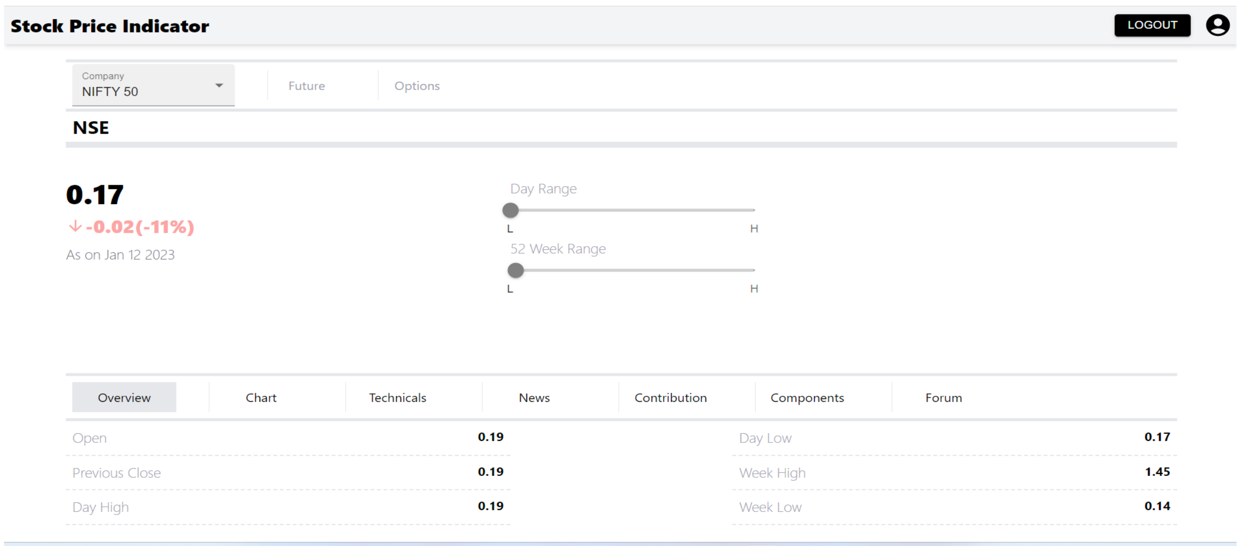
FLIPR Hackathon
REPORT



Stock Price Indicator

LOGOUT

Company
NIFTY 50

Future | Options

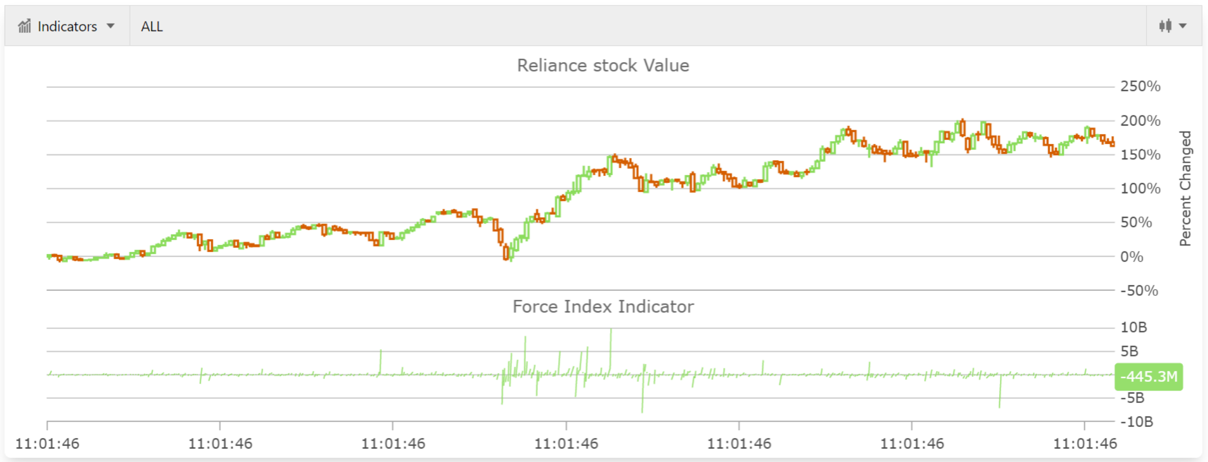
NSE

0.17
↓ -0.02(-11%)
As on Jan 12 2023

Day Range
L H
52 Week Range
L H

Overview | Chart | Technicals | News | Contribution | Components | Forum

Open	0.19	Day Low	0.17
Previous Close	0.19	Week High	1.45
Day High	0.19	Week Low	0.14



Introduction

As Part of the Flipr hackathon we have built a stock price indicator for the companies given in the problem statement. We used **ReactJs** for the front end, **Nodejs** for our backend, and **MongoDB** as our database to build our web app. We deployed this web app in an **EC2** instance on **AWS** the link to our web app is shared here web app link**.** A brief description of components of the front end and backend routes with database schema is written in this documentation.

Team Name	Team Members
Vishnu Gadi Team	Noel Vincent Polakallu
	Vishnu Shravan Karyampudi
	Pranav Balijapelly
	Varun Akavarupu

Front-End Documentation

Overview

The front end of our application is built using **ReactJs** for structuring and defining the skeleton of our website and **Tailwind CSS** for styling our application. We have divided our front end into different components, each component's description is documented below.

Sign-Up Component

This component contains the signup function's styling, structure, and logic. This component takes username, and password as input and calls the backend to validate and create the user. Once a signup is done the user needs to log in to our website.

Login Component

This component contains the styling, structure, and logic for the login function. This component calls the backend with the username and password entered by the user. The backend validates it and sends a JWT which will be stored in the local storage.

Nifty & BSE Component

This component is a replica of the image that was shared. This component queries the database for the day low, day high, 52-week low, and 52-week high values, etc. These values will be used to display the data on the web page. The structure, styling, and functionality of the page where the stock values of a particular day are displayed is written here.

Company Details Component

This component contains the styling, structure, and logic for the segment where a graph regarding the stock values of a particular company is displayed. This company queries the backend to get the stock values and uses react ignite chert library to display the values as a graph.

Back-End Documentation

Overview

Our Backend API is JSON-based JWT API. All the requests are to the endpoint of the local IP address. And every request must contain a JWT token for authentication/authorization.

The Backend of our application is built using Nodejs and MongoDB is used as a database. We chose MongoDB as our database cause it is Optimal for this application i.e MongoDB is optimized for storing time series data and general-purpose data.

Authentication

Authentication is done using JWT. The following endpoints are provided related to authentication.

POST http: //<ipaddress>/register

Request

A JSON object containing username and password

{username: <username>, password: <password>}

Responses

Status Code	Reason	message
400	<username> is already present.	{ message: User with this username already exists }
200	New users are created.	{ "username": "test", "_id": "ID of the user created" }

POST http://<ipaddress>/login

Request

Request body should contain JSON object containing username and password

{username: <username>, password: <password>}

Responses

Status Code	Reason	Message
401	Invalid user or password.	{ "message": "Authentication failed. Invalid user or password." }
200	Login successful	{ "username": user’s name, "token": JWT token value }

PUT http://<ipaddress>/logout

Request

Header must contain “Authorization”: Bearer <JWT token>”

Responses

Status	Reason	Message
401	If token is not present/ invalid token	{ “message”:User not logged in }
200	Valid token	{

		"msg": "You have been Logged Out" }

Resources

The API is RESTful and arranged around resources. All requests must be made with the integration of the JWT token sent when logged in.

Typically, this API serves 2 major resources. Those are

GET HTTP: //<ipaddress>/metadata/{companyName}

Request

A parameter company name is sent in the URL for retrieving its metadata.

Ex:- http://<ipaddress>/metadata/Reliance

Info

Getting the metadata of the company which includes the open, high, closed volumes, etc.. related info from the database of that particular company.

Response

Status Code	Reason	Response Message
200	If a company name is a valid request is served with a success message and data.	{ “Open”: 233.12; “Close ”: 222.543; “dayHigh”: 275.53; “dayLow”: 198.8378; “52weekLow”: 155.23; “52weekHigh”: 600.823 }

400	Bad Request	"Invalid data provided"
500	Internal server error	"Unable to process the request"

GET historicaldata/{CompanyName}

Getting the metadata of the company which includes the open, high, close volumes, etc.. related info from the database of that particular company.

GET http: //<ipaddress>/historicaldata/{companyName}

Request

A parameter company name is sent in the URL for retrieving its historical data.

Ex:- http://<ipaddress>/histroicaldata/Reliance

Info

Getting the historical data of the company i.e the past yearly data is sent as a response to valid requests.

Response

Status Code	Reason	Response Message
200	If a company name is a valid request is served with a success message and data.	[{ "timestamp": 2018-06-15 , " Open": 233.12, "Close ": 222.543,....}, { "timestamp": 2018-06-16 , " Open": 233.12, "Close ": 222.543,....}, { "timestamp": 2018-06-17 , " Open": 233.12, "Close ": 222.543,.....} ....]
400	Bad Request	"Invalid data provided"
500	Internal server error	"Unable to process the request"

Database Schema Documentation

Overview

Backend uses 3 major database schemas for UserSchema, StockCollectionSchema, and CompanySchema.

CompanySchema

```
{
        company_id: { type: Number, index: true },
        companyname: { type: String, required: true }
}
```

StockSchema

```
{ {
        timestamp: { type: Date, index: true },
        open: { type: Number, required: true },
        high:  { type: Number, required: true },
        low: type: Number, required: true },
        close:  { type: Number, required: true },
        adjclose: { type: Number, required: true },
        volume: { type: Number, required: true },
        companyid: {type: Number, required: true},
        metadata: { companyid: Number   },
        nse: {type: Boolean},
        bse: {type: Boolean}
},
{
        timeseries: {timeField: 'timestamp',metaField: 'metadata', granularity: 'hours'}
 } }
```

UserSchema

```
{
       username: { type: String, index: true },
       hash_password: {type: String}
}
```

LINKS

GITHUB FRONTEND LINK

GITHUB BACKEND LINK

WEBSITE LINK