

**International Institute of Information
Technology, Bangalore
(IIIT Bangalore)**

**Software Production Engineering
Project Report**

FinTrack - Shareable Investment Watchlists!

Under the Guidance of Prof. B. Thangaraju

Teaching Assistant: Neha Kothari



Jasvin James Manjaly
MT2021059

Thakur Devendrasingh
MT2021146

TABLE OF CONTENTS

Table of Contents

1. Abstract	4
2. Introduction	5
Overview	5
Features	6
3. System Configuration	7
Host System Configuration	7
Project Technology Stack	7
DevOps Tools	7
4. Software Development Life Cycle	8
Installations	8
Testing	11
Source Control Management	15
Containerization with Docker	16
Docker Compose	18
Ansible	20
Continuous Integration: Jenkins	24
Ansible Playbook	29
PaaS Deployment with Netlify & Heroku & Github Actions	32
Monitoring - ELK Stack	38
5. Experimental Setup	41
Functional Requirements	41
Non-Functional Requirements	41

Architectural Diagram	42
Code Walkthrough	43
6. Result and Discussion	59
7. Scope for Future Work	65
8. Conclusion	66
9. References	67

1. Abstract



“Share your Investment Watchlists now!”

The youth of our country are getting more and more interested in the securities market. For a newbie, it would be more helpful if they can get an overview of the correct investments by looking at the watchlists of experienced players in the ring.

There is a need for a platform where users can share their investments easily with their friends or relatives.

Typically any investment or investment tracker platform only lets you create watchlists for your own account and not let them be shareable.

Fintrack is the exact solution for this.

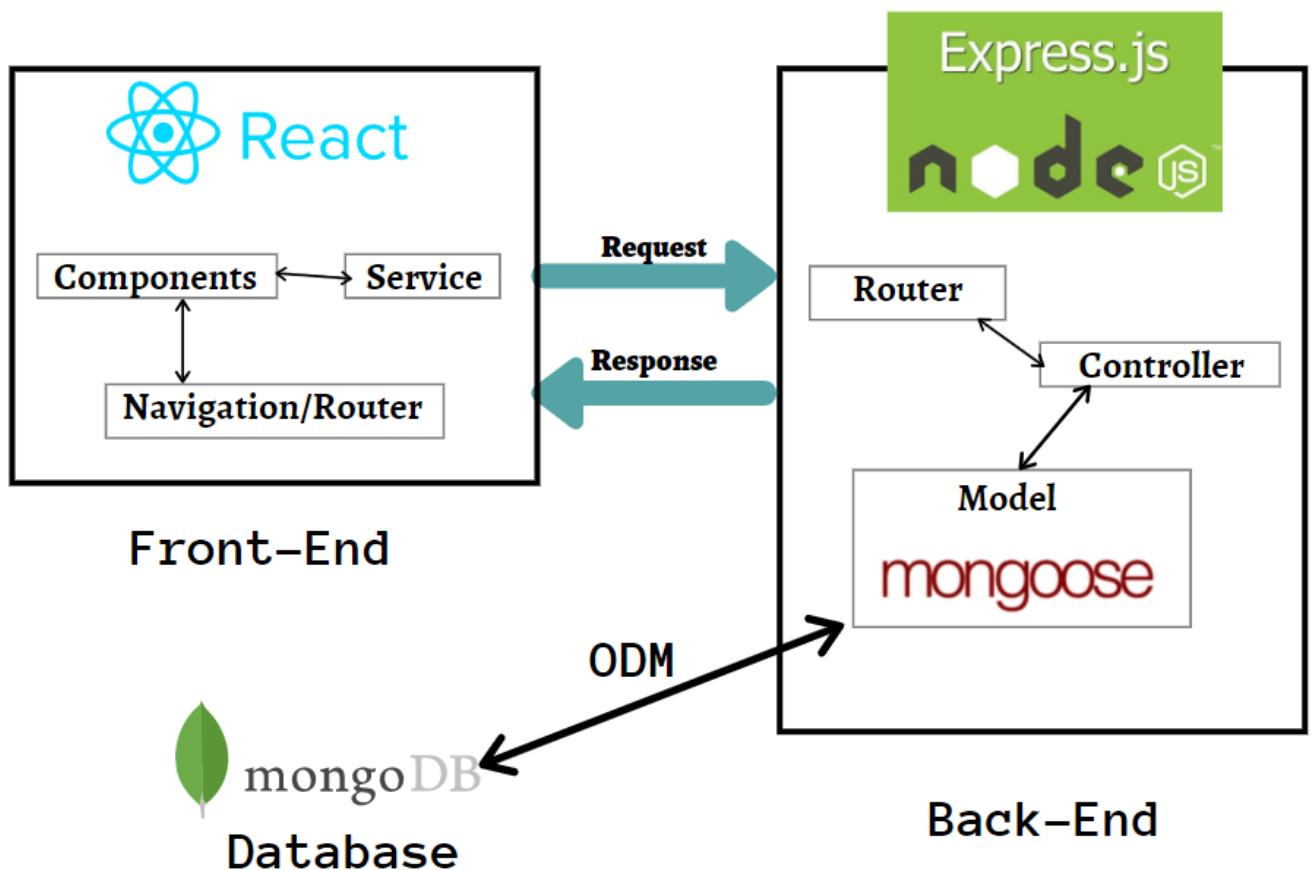
Here users can create their watchlists for stocks and mutual funds (MFs) as per their investments and share it with anyone they would want to - with ease!

2. Introduction

2.1. Overview

FinTrack is a website that allows users to create their watchlists for stocks and mutual funds (MFs) and share it anytime, anywhere and with anyone they would want to.

TechStack used → MERN (MongoDB, ExpressJS, ReactJS, NodeJS)



2.2. Features

a. Login/Signup

Account creation and login for users.

b. Home Page

Shows all the watchlists of the user.

c. Search Instruments

Search for stocks and mutual funds (MFs) on a search bar and add them to watchlists. A user can create multiple watchlists so there will be an option displayed to the user to determine which watchlist to add the current security to.

d. Create Watchlist

Users can create any number of watchlists they want with no duplicate entries. A watchlist can be a stock or MF watchlist but not both.

e. Share Watchlist

Each watchlist will have a share button that will send a publicly shareable link to anyone that visits that URL.

f. View Watchlist (Single Watchlist View)

This will show the specified watchlist's instruments. The view displayed when a user opens the shareable watchlist URL.

g. Delete Watchlist

Users can delete their watchlists anytime they want.

h. Delete Instrument

Delete a stock/MF from a Watchlist.

i. View live stock/mutual fund price → Each stock/MF has a URL to the TickerTape page for the corresponding stock/MF which will show the dynamic rate of change of stock/MF prices.

3. System Configuration

3.1. Host System Configuration

Operating System: 20.04.4 LTS (Focal Fossa)

CPU and RAM: 8 core processor with 16 GB RAM

Kernel Version: Linux version 5.13.0-40-generic

3.2. Project Technology Stack

Frontend: React (HTML, JSX, JavaScript, Bootstrap)

Backend: Express (NodeJS, JavaScript)

Database: MongoDB (No-SQL)

Cloud: MongoDB Atlas, Netlify, Heroku

Build Tool: npm

3.3. DevOps Tools

Source Control Management: Git/Github

Continuous Integration: Jenkins/Github Actions

Containerization: Docker/Docker Hub

Container Orchestration: Docker Compose

Testing: Supertest (Backend) + React Testing Library (Frontend)

Continuous Deployment: Ansible and/or Heroku & Netlify

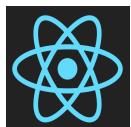
Logger: Morgan

Monitoring: ELK Stack (Elastic Search, Logstash, Kibana)

Secrets Management: Ansible Vault

4. Software Development Life Cycle

4.1. Installations



React

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies.

Update local before installing:

Keep the local packages and softwares updated.

```
sudo apt-get update
```

Install NodeJS and NPM:

Inorder to run React, Node environment shall be installed before starting,

```
sudo apt-get install nodejs  
sudo apt-get install npm
```

```
jasvin@jasvin-Bravo-15:~$ node -v  
v16.13.2  
jasvin@jasvin-Bravo-15:~$ npm -v  
8.1.2
```

React App

```
npx create-react-app fintrack
```

```
jasvin@jasvin-Bravo-15:~/Work/MTech/SPE/FinTrack/frontend$ npm start  
> fintrack@0.1.0 start  
> react-scripts start
```



Express

Express.js, or simply Express, is a back end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Express App

Initializing node project with the default config:

```
npm init -y
```

-y automates the config with node default values i.e without going through the interactive process.

Running the Node Application locally:

```
node server.js
```

Running the Node Application locally with nodemon:

[nodemon](#) DT

2.0.7 • Public • Published 4 months ago

[Readme](#) [Explore](#) BETA [10 Dependencies](#) [3,023 Dependents](#) [219 Versions](#)



nodemon

Install
npm i nodemon

Fund this package

Weekly Downloads
3,449,607

Version
2.0.7

License
MIT

Unpacked Size
107 kB

Total Files
43

Nodemon automatically restarts the node application whenever a file change in the directory is detected.

```
npm install nodemon  
nodemon server.js
```

The server is setup to run on localhost:3001

```
jasvin@jasvin-Bravo-15:~/Work/MTech/SPE/FinTrack/backend$ npm run dev  
  
> fintrack@1.0.0 dev  
> NODE_ENV=development nodemon index.js  
  
[nodemon] 2.0.15  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): .*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node index.js`  
Server running on port 3001  
Successfully connected to MongoDB
```

4.2. Testing



4.2.1. Backend Testing (Supertest)

Supertest provides a high-level abstraction for testing HTTP, while still allowing you to drop down to the [lower-level API](#) provided by superagent.

Here is a snippet of how to write tests with Supertest, in this case, we are testing the `/users` API. Similarly we have tested the `/watchlists` and `/watchlistInstruments` APIs.

```
const mongoose = require('mongoose')
const supertest = require('supertest')
const app = require('../app')
const User = require('../models/user')
const api = supertest(app)
const helper = require('./tests_helper')

beforeEach(async () => {
  // Increasing timeout otherwise sometimes a timeout error can wreck the whole testing phase
  jest.setTimeout(30000)

  await User.deleteMany({})
  await User.insertMany(helper.initialUsers)
})

describe('adding a new user', () => {
  test('valid user is added successfully', async () => {
    const user = {
      'username': 'murray',
      'name': 'James Murray',
      'password': '123456'
    }

    await api
      .post('/api/users')
      .send(user)
      .expect(201)
      .expect('Content-Type', /application\/json/)

    const users = await helper.usersInDb()
    expect(users).toHaveLength(helper.initialUsers.length + 1)

    const usernames = users.map(user => user.username)
    expect(usernames).toContain('murray')
  })
})
```

Note that these files must have the extension `.test.js` in order for Jest to pick up these files as testing files. After this you can run the tests by running the command `npm run test`

```

jasvin@jasvin-Bravo-15:~/Work/MTech/SPE/FinTrack/backend$ npm run test

> fintrack@1.0.0 test
> NODE_ENV=local-test jest --verbose --runInBand

PASS  tests/watchlist_instruments_api.test.js (14.453 s)
  viewing a watchlist's instruments
    ✓ watchlistInstrument object has id property instead of _id (2986 ms)
    ✓ total instruments contained are 1 (836 ms)
    ✓ cannot view watchlists without bearer token (807 ms)
  adding a watchlist instrument
    ✓ new watchlist instrument is created successfully (1050 ms)
    ✓ watchlist instrument with missing instrumentId gets 400 response (796 ms)
    ✓ stock instrument cannot be added to mf watchlist (810 ms)
    ✓ mf instrument cannot be added to stock watchlist (804 ms)
    ✓ cannot add same instrument twice in the same watchlist (787 ms)
  deleting a watchlist instrument
    ✓ existing watchlist instrument is deleted successfully (880 ms)
    ✓ non-existing watchlist returns 404 (928 ms)
  deleting all instruments of a watchlist
    ✓ deletion happens successfully (867 ms)
    ✓ non-existing watchlist returns 404 (837 ms)

PASS  tests/watchlists_api.test.js (9.024 s)
  viewing a user's watchlists
    ✓ watchlist object has id property instead of _id (1707 ms)
    ✓ total watchlists contained are 4 (623 ms)
    ✓ cannot view watchlists without bearer token (593 ms)
  viewing a single watchlist
    ✓ existing watchlist can be viewed successfully (645 ms)
    ✓ existing watchlist cannot be viewed without bearer token (592 ms)
    ✓ non-existing watchlist returns 404 (686 ms)
  adding a watchlist
    ✓ new watchlist is created successfully (682 ms)
    ✓ watchlist object with missing name gets 400 response (646 ms)
    ✓ watchlist object with missing isMF parameter gets 400 response (630 ms)
  deleting a watchlist
    ✓ existing watchlist is deleted successfully (734 ms)
    ✓ non-existing watchlist returns 404 (677 ms)

```

For backend testing we have created a testing database on MongoDB Atlas which has its own URI separate from the production database. This ensures that testing doesn't interfere with the production database at any point in time and we can do operations like deleting entire entries from collections in the testing database in order to maintain consistency between tests.



4.2.2. Frontend Testing (React Testing Library)

The React Testing Library is a very light-weight solution for testing React components. It provides light utility functions on top of react-dom and react-dom/test-utils, in a way that encourages better testing practices. Its primary guiding principle is:

“The more your tests resemble the way your software is used, the more confidence they can give you.”

So rather than dealing with instances of rendered React components, your tests will work with actual DOM nodes. The utilities this library provides facilitate querying the DOM in the same way the user would. Finding form elements by their label text (just like a user would), finding links and buttons from their text (like a user would). It also exposes a recommended way to find elements by a data-testid as an "escape hatch" for elements where the text content and label do not make sense or are not practical.

Here is how we have tested the WatchlistForm component with just the frontend, no backend involved.

```

import React from 'react'
import '@testing-library/jest-dom/extend-expect'
import userEvent from '@testing-library/user-event';
import { render, fireEvent, screen } from '@testing-library/react'
import WatchlistForm from './WatchlistForm'

describe('<WatchlistForm/>', () => {
  const createWatchlist = jest.fn()

  test('watchlist form submission simulated successfully', () => {
    render(<WatchlistForm createWatchlist={createWatchlist} />

    const watchlistNameInput = screen.getByLabelText('Watchlist Name:')
    const isMFInput = screen.getByTestId('mf-radio-button')

    const form = screen.getByTestId('watchlist-form')

    fireEvent.change(watchlistNameInput, {
      target: {
        value: 'Joe Mama'
      }
    })

    userEvent.click(isMFInput)

    fireEvent.submit(form)

    expect(createWatchlist.mock.calls).toHaveLength(1)
    expect(createWatchlist.mock.calls[0][0].name).toBe('Joe Mama')
    expect(createWatchlist.mock.calls[0][0].isMF).toBe(true)
  })
})
}

```

We run these tests using the `npm run test` command as well,

```

PASS  src/components/Watchlist.test.js
PASS  src/components/WatchlistForm.test.js

Test Suites: 2 passed, 2 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.457 s, estimated 3 s
Ran all test suites.

Watch Usage: Press w to show more.

```



4.3. Source Control Management (SCM)

Source Control Management is used for tracking the file change history, source code, etc. It helps us in many ways in keeping the running project in a structured and organized way.

Repository Link: <https://github.com/james-jasvin/FinTrack>

The frontend is created in the *frontend/* directory and the backend is created in the *backend/* directory.

Initializing the project with git:

```
git init  
git remote add origin https://github.com/james-jasvin/FinTrack.git
```

Workflow:

```
git add <files>  
git commit -m "commit message"  
git pull origin master  
git push origin master
```

The code is first pulled before making a push to make sure that our project is in the latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “master” branch.

Working on a feature/issue:

```
git checkout master  
git checkout -b "<your_branch_name>"
```

After creating a branch, required changes are done and a pull request to the main is created.

```
git add <files>  
git commit -m "commit message"  
git pull origin master  
git push origin master
```

Then merge the pull request from Github.



4.4. Containerization with Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.

Frontend Dockerfile

```
# Dockerfile for React client

# Use node's alpine variant to save on space and version number is 17 (latest)
FROM node:17-alpine

# Create the folder "app" under the /usr/src path and set it to be the working directory
# for the further COPY, RUN and CMD instructions
WORKDIR /usr/src/app

# Copy the package.json and package-lock.json files to the "app" folder
COPY package*.json ./

# Install the dependencies mentioned in package.json
RUN npm install

# Copy the local files to the "app" folder
COPY . .

# Expose port 3000 on the host machine to the container for listening to external connections
EXPOSE 3000

# Start the React applications
CMD ["npm", "start"]
```

Backend Dockerfile

```
# Dockerfile for Express Backend

# Use node's alpine variant to save on space and version number is 17 (latest)
FROM node:17-alpine

# Create the folder "app" under the /usr/src path and set it to be the working directory
# for the further COPY, RUN and CMD instructions
WORKDIR /usr/src/app

# Copy the package.json and package-lock.json files to the "app" folder
COPY ./package*.json .

# Install the dependencies mentioned in package.json
RUN npm install

# Copy the local files to the "app" folder
COPY . .

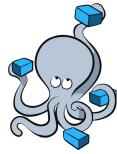
# Expose port 3001 on the host machine to the container for listening to external connections
EXPOSE 3001

CMD ["npm", "start"]
```

Running *docker build* using this Dockerfile as the source creates the required Docker image that is ready to run our application.

We need to build the Docker image and push it to Docker Hub which requires logging in to the DockerHub account as well.

This will be covered in the Continuous Integration section.



4.5. Docker Compose

Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.

The big advantage of using Compose is you can define your application stack in a file, keep it at the root of your project repo (it's now version controlled), and easily enable someone else to contribute to your project. Someone would only need to clone your repo and start the Compose app.

On the host machine, follow the instruction of this link

<https://docs.docker.com/compose/install/>

After installation, you should be able to run the following and see version information.

```
docker-compose version
```

Compose File

At the root of the app project, create a file named *docker-compose.yml*

You can look at the Compose file reference for Compose file syntax and features.

<https://docs.docker.com/compose/compose-file/>

Note: “*version*” parameter is not required in Docker Compose YAML files since version 1.27.

Next, we'll define the list of services (or containers) we want to run as part of our application.

Create two containers for running the app, one for the frontend and one for the backend.

Map port 3000 on the host machine to port 3000 on the frontend container because that's where React runs and do the same with the backend but with port 3001 because we've configured the Express app to listen on port 3001.

Attach both containers to the same network which is named *fintrack*, this enables the frontend container to communicate with the backend container and vice-versa. We fix

the subnet of the *fintrack* network and then apply static IP addresses to both the frontend and backend containers. Because of the static IP addresses, the frontend can fire queries to the backend using a fixed backend URL.

The reason for using a Docker volume is already specified in the code itself.

```
services:
  fintrack-frontend:
    image: jasvinjames/fintrack-frontend:latest
    container_name: fintrack-frontend
    # Keep stdin_open because we don't want this container to exit immediately which it
    # otherwise would have because there's no foreground processes
    stdin_open: true
    ports:
      - "3000:3000"
    networks:
      fintrack:
        ipv4_address: 172.28.1.6

  fintrack-backend:
    image: jasvinjames/fintrack-backend:latest
    container_name: fintrack-backend
    ports:
      - "3001:3001"
    networks:
      fintrack:
        ipv4_address: 172.28.1.7
    # Using Docker volume for persisting logs
    # ./backend/logs refers to the path /home/backend/logs on the host machine
    # and we're linking it to logs folder on the container machine where Morgan logger logs its output
    # volumes:
    #   - ./backend/logs:/usr/src/app/logs/

networks:
  fintrack:
    ipam:
      driver: default
      config:
        - subnet: 172.28.1.0/24
```



4.6. Ansible

Ansible is an open-source automation tool, or platform, used for IT tasks such as configuration management, application deployment, intraservice orchestration and provisioning.

Ansible is mainly used to perform a lot of tasks that otherwise are time-consuming, complex, repetitive, and can make a lot of errors or issues.

Note: We are going to be pulling the Docker Hub image to the host system for Ansible deployment.

Creating Inventory file

The inventory file is used to specify the list of managed hosts/server machines.

The inventory file looks as,

```
<host-machine-IP-address> ansible_user=jasvin
```

jasvin is the host machine's user on which Ansible shall execute the specified commands.

Create this file in the root directory of your project repository with the name *inventory*.

Configuring OpenSSH Server

Install openssh-server on the host machine and because it is the Jenkins user that is going to be doing the pulling of Docker image, we need to SSH from the Jenkins user to the user that is specified in the inventory file, i.e. *jasvin*.

```
apt-get install openssh-server
service ssh restart
su jenkins
ssh-keygen -t rsa
ssh-copy-id <host-username>@<host-ip-address>
sudo chmod 666 /var/run/docker.sock
```

The *chmod* command is required so that the Jenkins user has access to the Docker socket for performing the docker build and push operations.

Because we use MongoDB Atlas for storing our database, we require the MongoDB URI in order to access the database and this URI should be kept secret. However we have to send this URI in a .env type of file to the backend container, how to do this and not leak the URI in any way? This is where Ansible Vault comes in the picture.

Ansible Vault

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

First we have to encrypt the environment file using *ansible-vault*

```
jasvin@jasvin-Bravo-15:~/Work/MTech/SPE/FinTrack/backend$ ansible-vault encrypt env-local.yaml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Command for decrypting the environment file using *ansible-vault*

```
jasvin@jasvin-Bravo-15:~/Work/MTech/SPE/FinTrack/backend$ ansible-vault decrypt env-local.yaml
Vault password:
Decryption successful
```

Encrypted YAML file

```
$ANSIBLE_VAULT;1.1;AES256
396430613335309313838396237653638363330340309373233636231666364336663343338323661
3632633234663835313063263363326337396137653661640a663032643432626532333138366134
3665626530383353635383334393331643236636637306630373961316630646236306332353261
3133643166623133650a363435376165643763383638666532663837623134393139343636353661
393137343530663661346464613861626136353535309336435663264643631373332633933333730
66366239336353634613733361353763623830313135333635309306434373763353732643866
35666532316135343965616165383432613437646534373631626333653332343930326337386564
623065633834656661363537323937653134316438326430303163353363306665623166393637
3035373264353343362616334343339376561323463356237626239346331616635643234336631
653939353865313537326436535389333636434313732356561316433313535363762656330
3865616565313064333833343130376530393861383832393639373436663866396663383346165
3033653738646437303135343731353335356339393834313663636336306661623339663343035
3336353765363736643161358646137363435333763313138313434656666343631313662646137
3164666463653837663765313666303534393365313338264363963343965313834343564363530
616637336435396433393138303856238653461624613966623563643334643730636333623638
33343231623361653531626236373734653566316664364613335316661653935343364623032
61373131643265333166613934303464333164326534363633337372388663832653430373037
3063662336430665639663738323730313431373134643037623163626334366393135336365
37323162316234396434313463343935663738643265396564306265356335313765363332666335
643062356136613835366361646236537373161306462365316233353139393865626235383030
32383065373436363676534565132643838313337653731396266383765633761333563396261
3031323339366165376163323396330353065323035626239626665393436636438373263376635
3237316435353563366365636263346330623937623636465656132366430386161333735
30656234306136646234313336356639656531333664653734636433343139323030313364653130
3630313432343466383366626530323265643334363033635613064653331613861326239303832
663838616266333436333563336363165373565313963343534326365306564393316331343066
3963663235633365306366353236373934633631666535353566653830376161653266136623636
62623337356634383939653933383962396265333466313862643765313737643165373439663066
6562343633646664343453535393063366332663233623062326236666335326235326165333861
3766616637383661326532373123963316130376530666337316638653035633037323939326432
646333137613538386137343937663932386303637338736396435316334623532613062336333
61323264386663643062643961383436626236353265653731303163313930633835313433393431
3832316466313932383336231643436633363623130633166316566333566653461356335353431
6366662666364373361643336623834376336316663731323037
```

We are using yaml files for storing environment configuration instead of a .env file because Ansible Vault requires a yaml or JSON type file for encrypting and decrypting. This will become more clear when the playbook is explained.

Environment YAML decrypted

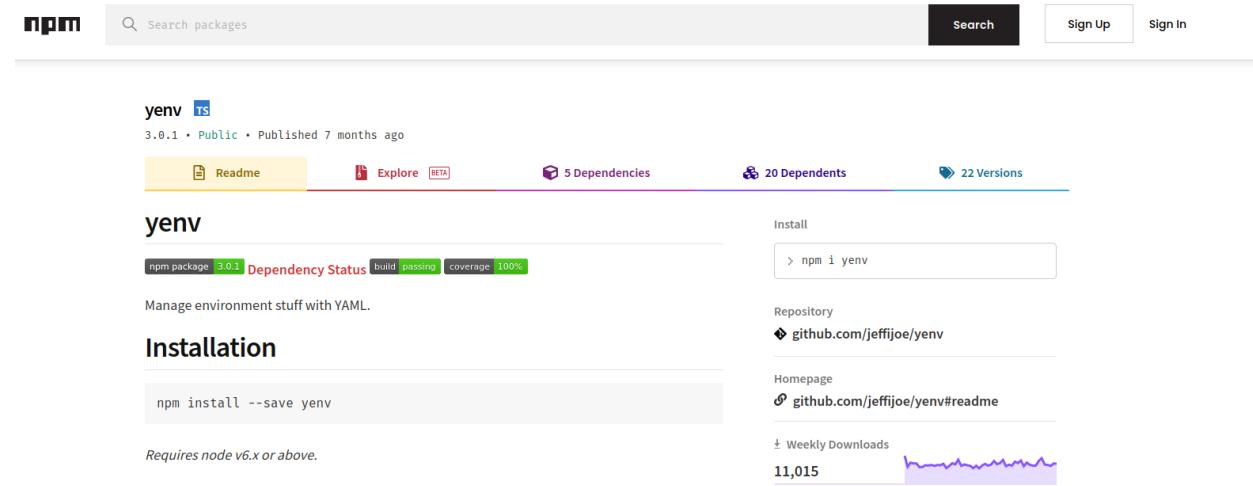
```
production:
  PORT: '3001'
  MONGODB_URI: '<URI here>'
  SECRET: '<Secret here>'

local-test:
  PORT: '3001'
  MONGODB_URI: '<URI here>'
  SECRET: '<Secret here>'

development:
  PORT: '3001'
  MONGODB_URI: '<URI here>'
  SECRET: '<Secret here>'
```

The environment tags like production, local-test and development are what will be used to separate the environment variables from each other, because some values like URI will be different for production and testing environments and so on.

yenv library for reading environment variables,



The screenshot shows the npm package page for 'yenv'. At the top, there's a search bar and navigation links for 'Search', 'Sign Up', and 'Sign In'. Below the header, the package details are shown: 'yenv' (version 3.0.1, published 7 months ago), 'Readme' (selected), 'Explore (BETA)', '5 Dependencies', '20 Dependents', and '22 Versions'. The 'Readme' tab contains the package description: 'Manage environment stuff with YAML.' Under the 'Installation' section, the command 'npm install --save yenv' is listed. A note below it says 'Requires node v6.x or above.' To the right of the main content, there's an 'Install' section with a search bar containing the command 'npm i yenv', a 'Repository' link to 'github.com/jeffjoe/yenv', a 'Homepage' link to 'github.com/jeffjoe/yenv#readme', and a 'Weekly Downloads' chart showing 11,015 downloads.

Using the `yenv` library in the `config.js` file for setting up environment variables

```
const yenv = require('yenv')

let env = null

/*
 * If production environment then use the env.yaml file (generated via Ansible Vault + Playbook)
 * for the environment variables.
 * If testing environment then use the env-enc.yaml file (which will be decrypted in the Jenkins pipeline).
 * And if in development environment, use the env-local.yaml file.
 */
if (process.env.NODE_ENV == 'production')
  env = yenv()
else if (process.env.NODE_ENV == 'test')
  env = yenv('env-enc.yaml')
else if (process.env.NODE_ENV == 'production-heroku')
  env = process.env
else
  env = yenv('env-local.yaml')

const PORT = env.PORT
const MONGODB_URI = env.MONGODB_URI
const SECRET = env.SECRET

module.exports = { PORT, MONGODB_URI, SECRET }
```

Jinja2 Template that will be required for decrypting environment variables via Ansible Vault,

```
production:
  PORT: {{ production.PORT }}
  MONGODB_URI: {{ production.MONGODB_URI }}
  SECRET: {{ production.SECRET }}
```

The Ansible playbook will be explained after the next section.



4.7. Continuous Integration: Jenkins

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

Create a Jenkins Pipeline project,

The screenshot shows the Jenkins dashboard with a search bar and user information at the top. Below is a list of items under 'Dashboard' and 'All'. A modal window is open for creating a new item, titled 'Enter an item name'. The name 'Fintrack' is entered, and a red message indicates that a job already exists with that name. Below the name, there are three project types: 'Freestyle project', 'Maven project', and 'Pipeline'. The 'Pipeline' option is selected, and its description is visible: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.'

Reading Jenkins pipeline script, *Jenkinsfile* from the SCM repository

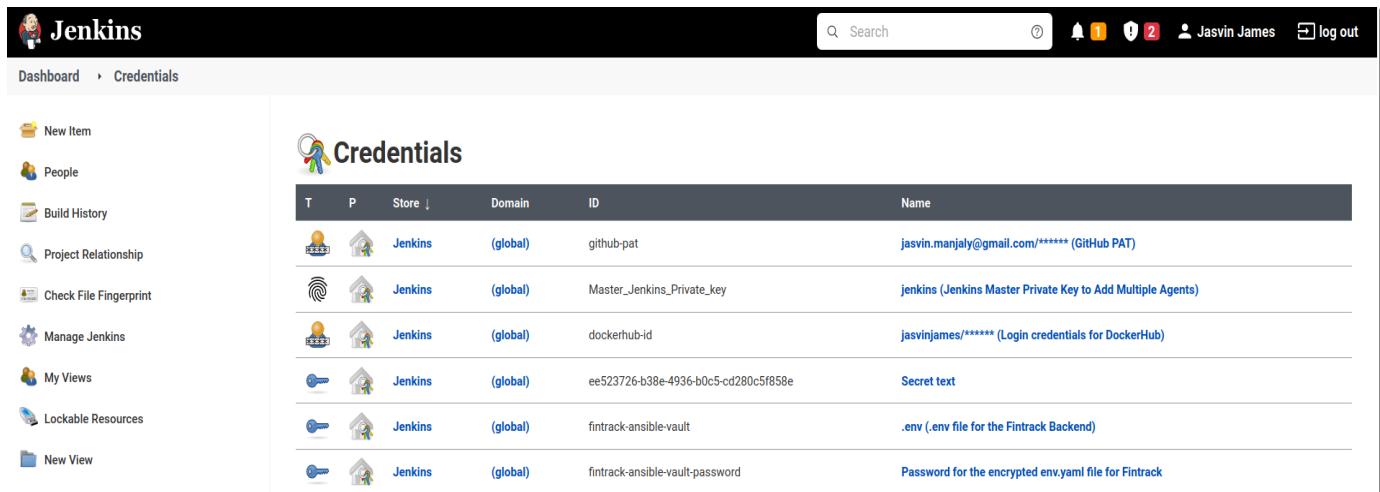
The screenshot shows the 'Definition' section of a Jenkins pipeline configuration. Under 'Pipeline script from SCM', 'Git' is selected as the provider. In the 'Repositories' section, the 'Repository URL' is set to 'https://github.com/james-jasvin/FinTrack.git'. Under 'Credentials', a GitHub PAT (Personal Access Token) is listed as 'jasvin.manjaly@gmail.com/***** (GitHub PAT)'. There is an 'Advanced...' button. At the bottom, there are 'Add Repository' and 'Branches to build' sections. The 'Branch Specifier' field contains the value '/master'.

Credentials & Jenkins

Github PAT: For accessing private Github repositories.

Docker Hub Credentials: For logging into Docker Hub account and pushing Docker images (it is preferred to use Access Token instead of your actual Docker Hub password).

Ansible Vault Password: Save the password that was used to encrypt the `env-enc.yaml` file as a *Secret Text* credential on Jenkins.



The screenshot shows the Jenkins interface with the title "Credentials". On the left, there is a sidebar with links like "New Item", "People", "Build History", etc. The main area displays a table of credentials:

T	P	Store	Domain	ID	Name
🔒	🔑	Jenkins	(global)	github-pat	jasvin.manjaly@gmail.com/******** (GitHub PAT)
⌚	🔑	Jenkins	(global)	Master_Jenkins_Private_key	jenkins (Jenkins Master Private Key to Add Multiple Agents)
🔒	🔑	Jenkins	(global)	dockerhub-id	jasvinjames/******** (Login credentials for DockerHub)
🔑	🔑	Jenkins	(global)	ee523726-b38e-4936-b0c5-cd280c5f858e	Secret text
🔑	🔑	Jenkins	(global)	fintrack-ansible-vault	.env (.env file for the Fintrack Backend)
🔑	🔑	Jenkins	(global)	fintrack-ansible-vault-password	Password for the encrypted env.yaml file for Fintrack

Pipeline Script

First we set up the environment variables which includes importing the Docker Hub and Ansible Vault passwords as credentials.

```
pipeline {
    // Declare variables that will be used by the later stages
    environment {
        DOCKERHUB_REGISTRY = "jasvinjames/fintrack"
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-id')
        ANSIBLE_VAULT_CREDENTIALS = credentials('fintrack-ansible-vault-password')
    }

    agent any
```

Then we perform a Git pull on the repository's master branch with the Github PAT as the `credentialsId`.

```
stages {  
  
    stage('Git Pull') {  
        steps {  
            // credentials are required because its a private repository  
            git url: 'https://github.com/james-jasvin/FinTrack.git',  
            branch: 'master',  
            credentialsId: 'github-pat'  
        }  
    }  
}
```

After this we run the frontend testing which is done via the React Testing Library and for this we use a multi-line shell script (triple quoted string) in which we `cd` to the `frontend` directory, install all the dependencies and run the tests.

```
stage ('Running React Tests (Jest)') {  
    steps {  
        sh '''  
            cd frontend  
            npm ci  
            npm run test  
        '''  
    }  
}
```

Now we run backend testing via the Supertest library. An important point to note here is that we need to connect to the MongoDB Atlas cluster in order to access the database while testing. So we have set up a separate database in the same collection for testing.

The screenshot shows the MongoDB Compass interface. At the top, it displays the project name 'phonebook-app-cluster' and the region 'AWS Mumbai (ap-south-1)'. Below the header, there are tabs for Overview, Real Time, Metrics, Collections (which is selected), Search, Profiler, Performance Advisor, Online Archive, and Cmd Line Tools. A sub-header indicates 'DATABASES: 5' and 'COLLECTIONS: 13'. On the left, a sidebar lists databases: bloglist-app, bloglist-test-app, fintrack-app, and fintrack-test-app (which is expanded to show collections: instruments, users, watchlistinstruments, and watchlists). The main panel is titled 'fintrack-test-app' and shows collection statistics:

Collection Name	Documents	Documents Size	Documents Avg	Indexes	Index Size	Index Avg
instruments	7	1.03KB	151B	2	44KB	22KB
users	2	294B	147B	2	76KB	38KB
watchlistinstruments	6	468B	78B	1	36KB	36KB
watchlists	4	356B	89B	1	36KB	36KB

But a big problem now is that we have encrypted the yaml file which contains this testing database's URI, so what to do?

We decrypt the `env-enc.yaml` file using the Ansible Vault password which we have saved as a credential in Jenkins, store it in a file called `secret.txt`, run the backend tests, encrypt the file once again using the `secret.txt` file and now delete the `secret.txt` file if it exists.

```
stage ('Running API Tests (Supertest)') {
    steps {
        sh '''
            cd backend
            echo $ANSIBLE_VAULT_CREDENTIALS > secret.txt
            ansible-vault decrypt env-enc.yaml --vault-password-file secret.txt
            npm ci
            npm run test
            ansible-vault encrypt env-enc.yaml --vault-password-file secret.txt
            if [ -f secret.txt ] ; then
                rm secret.txt
            fi
        '''
    }
}
```

After this we build the Docker images, push them to Docker Hub and remove the local Docker images.

```
stage('Build Fintrack Backend Docker Image') {
    steps {
        sh "docker build -t $DOCKERHUB_REGISTRY-backend:latest backend/"
    }
}

stage('Build Fintrack Frontend Docker Image') {
    steps {
        sh "docker build -t $DOCKERHUB_REGISTRY-frontend:latest frontend/"
    }
}

stage('Login to Docker Hub') {
    steps {
        sh "echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin"
    }
}

stage('Push Backend Docker Image to Docker Hub') {
    steps {
        sh "docker push $DOCKERHUB_REGISTRY-backend:latest"
    }
}

stage('Push Frontend Docker Image to Docker Hub') {
    steps {
        sh "docker push $DOCKERHUB_REGISTRY-frontend:latest"
    }
}

stage('Removing Docker Images from Local') {
    steps {
        sh "docker rmi $DOCKERHUB_REGISTRY-frontend:latest"
        sh "docker rmi $DOCKERHUB_REGISTRY-backend:latest"
    }
}
```

Now we finally call the Ansible playbook, note that we also supply the Ansible Vault password to it.

```
// Ansible Deploy to remote server (managed host)
stage('Ansible Deploy') {
    steps {
        ansiblePlaybook becomeUser: 'null',
        colorized: true,
        installation: 'Ansible',
        inventory: 'inventory',
        playbook: 'ansible-playbook.yml',
        sudoUser: 'null',
        vaultCredentialsId: 'fintrack-ansible-vault-password'
    }
}
```

4.8. Ansible Playbook

The main purpose of creating playbooks is that it encapsulates all the tasks under one playbook file.

In this playbook, the toughest task is to use the encrypted `env-enc.yaml` file and decrypt it somehow to send to the managed nodes and somehow place it inside the running container.

That is why we use the `vars_files` module with the Ansible playbook. We specify the `env-enc.yaml` as the file to look for and because Jenkins invoked the Ansible playbook with the Vault credentials, the file will now be decrypted for direct use but we cannot copy this file into the Docker container, which is why we use templating and this is where that `env.j2` template comes into the picture.

The `backend/env.j2` file which contains Jinja2 template variables will now be replaced by the decrypted environment variables in `env-enc.yaml` and we use the `template` module to store the created file under the name, `env.yaml` under the root directory of the managed nodes.

Now all we have to do is copy this `env.yaml` file into the backend container.

Everything else is explained in the comments of the playbook whose image is attached below.

```

---  

- name: Deploy Docker Images  

  hosts: all  

  vars_files:  

  - ./backend/env-enc.yaml  

  tasks:  

    - name: Copy Docker Compose file from host machine to remote host  

      copy:  

        src: ./docker-compose.yml  

        dest: ./  

    - name: Replace Jinja2 template with decrypted environment variables into env.yaml file  

      template:  

        src: backend/env.j2  

        dest: env.yaml  

    # Pull the Docker images from Docker Hub  

    - name: Pull the Docker images specified in docker-compose  

      command: docker-compose pull  

    # We don't start the containers because we need to copy the env.yaml file into the backend container  

    # before we can start anything.  

    # So we first create the containers so that the next command "docker cp" can run properly  

    - name: Create containers for the pulled Docker images  

      command: docker-compose up --no-start  

    - name: Copy the env.yaml file to backend Docker container  

      command: docker cp env.yaml fintrack-backend:/usr/src/app/env.yaml  

    # Now we actually run the Docker containers because the env.yaml has been copied and the app will no longer crash  

    # Detached mode is required, otherwise Jenkins build never exits  

    # even though the docker-compose up command has successfully executed  

    - name: Run the pulled Docker images in detached mode  

      command: docker-compose up -d  

    # This is added so that the Docker images of the previous builds  

    # which will now become dangling images are removed  

    - name: Prune the dangling Docker images  

      command: docker image prune --force

```

Running Jenkins build,

The screenshot shows the Jenkins Pipeline Fintrack dashboard. On the left, there's a sidebar with various options like Back to Dashboard, Status, Changes, Build Now (which is highlighted), Configure, Delete Pipeline, Full Stage View, Rename, Pipeline Syntax, and GitHub Hook Log. Below this is the Build History section, which lists three builds: #63 (May 7, 2022, 8:44 PM), #62 (May 6, 2022, 7:43 PM), and #61 (May 6, 2022, 7:37 PM). The main area is titled "Pipeline Fintrack" and "Stage View". It displays a timeline of stages: Declarative: Checkout SCM, Git Pull, Running React Tests (Jest), Running API Tests (Supertest), Build Fintrack Backend Docker Image, Build Fintrack Frontend Docker Image, Login to Docker Hub, Push Backend Docker Image to Docker Hub, Push Frontend Docker Image to Docker Hub, Removing Docker Images from Local, and Ansible Deploy. Each stage has a progress bar indicating its duration. A tooltip for the first stage says "Average stage times: (Average full run time: ~3min 46s)".

Running Docker containers after build,

```
jasvin@jasvin-Bravo-15:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
3edcdc2b22b9        jasvinjames/fintrack-backend:latest   "docker-entrypoint.s..."   8 minutes ago     Up 8 minutes      0.0.0.0:3001->3001/tcp, :::3001->3001/tcp   fintrack-backend
0fb2de566d3f        jasvinjames/fintrack-frontend:latest   "docker-entrypoint.s..."   8 minutes ago     Up 8 minutes      0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   fintrack-frontend
jasvin@jasvin-Bravo-15:~$
```



HEROKU



netlify



4.9. PaaS Deployment with Netlify & Heroku & Github Actions

We can deploy a MERN stack based application directly to Heroku and serve the React frontend as a static production build, however this is only applicable when the React app is a SPA (Single Page Application), whereas we have used React Router in order to create a Multi Page Application. So that is where Netlify helps with its ease of use for deployment as well as handling React Router based applications.

Following describes the steps that have to be followed for deploying the Express backend to Heroku and the React frontend to Netlify.

Heroku Configuration

Create a Heroku account and create a new app.

We can deploy to Heroku using several methods but we have used the Github Actions route. Before we could directly connect a Github repository to a Heroku app but a recent vulnerability with Heroku has forced them to disable this option.

So we use a third-party Github Action for deploying to Heroku

Marketplace / Actions / Deploy to Heroku

GitHub Action
Deploy to Heroku
v3.12.12 [Latest version](#)

Use latest version

Heroku Deploy

Issues 40 open License MIT Tests passing

This is a very simple GitHub action that allows you to deploy to Heroku. The action works by running the following commands in shell via NodeJS:-

Table of Contents

- 1. Getting Started
- 2. Important Note
- 3. Options
- 4. Examples
 - Deploy with Docker
 - Deploy with custom Buildpacks
 - Deploy Subdirectory
 - Deploy Custom Branch

Stars
Star 720

Contributors

Categories
Deployment Publishing

Links
[AkhileshNS/heroku-deploy](#)
[Open issues](#) 40
[Pull requests](#) 7
[Report abuse](#)

For the Action to work, it should be able to login to your Heroku account and we also need a new way to manage secret environment variables because Ansible will not be

involved with this deployment. We configure these secret environment variables in the repository's secrets which can be set by going to the *Settings* menu of your repository.

Also to login to Heroku we save the Heroku API key (available in the *Account Settings* section of your Heroku account) in the Github secrets as well.

The screenshot shows the GitHub settings interface. At the top, there is a navigation bar with links for 'Jump to Favorites, Apps, Pipelines, Spaces...'. On the right, there is a user profile sidebar with options for 'Account settings', 'Notifications', and 'Sign out'.

Multi-Factor Authentication: Shows that Multi-Factor Authentication is enabled. There is a link to 'Manage Multi-Factor Authentication'.

SSH Keys: Shows a message stating 'There are no SSH keys yet' and 'You can register new SSH keys to enable command line access to your apps.' A button to 'Add New SSH Key' is present.

API Key: Shows an API key value represented by a series of dots ('.....'). There is a 'Reveal' button to show the full key and a 'Regenerate API Key...' button.

The screenshot shows the 'Actions secrets' section of a GitHub repository settings page. The left sidebar has a 'Secrets' section selected. The main area displays a list of secrets:

Secret	Last Updated	Action
HEROKU_API_KEY	Updated yesterday	Update Remove
MONGODB_URI	Updated 6 days ago	Update Remove
PORT	Updated 6 days ago	Update Remove
SECRET	Updated 6 days ago	Update Remove

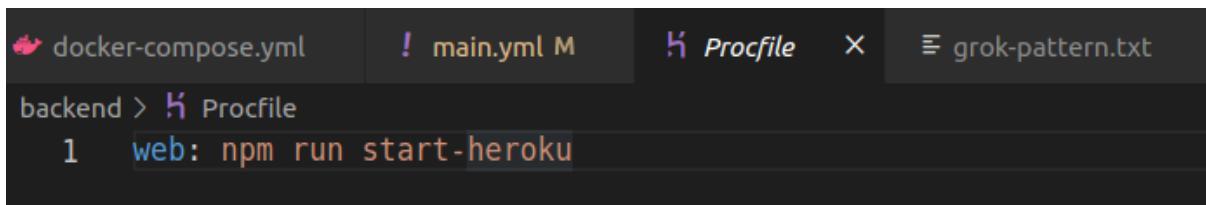
And this is what the yml file looks like for the Workflow,

```
name: Deploy

on:
  push:
    branches:
      - master

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: akhileshns/heroku-deploy@v3.12.12
        with:
          heroku_api_key: ${{secrets.HEROKU_API_KEY}}
          heroku_app_name: "fintrack-420"
          heroku_email: "jamesjasvin5@gmail.com"
          # Specifies the directory in the repository that will be treated as the root by Heroku
          # because we are only deploying the backend, that is our appdir
          appdir: "backend"
    env:
      # All environment variables for this Action must be specified with the HD_ prefix
      HD_MONGODB_URI: ${{secrets.MONGODB_URI}}
      HD_PORT: ${{secrets.PORT}}
      HD_SECRET: ${{secrets.SECRET}}
```

Finally in order to get a Heroku app running we need to add a *Procfile* which in this case will be in the *backend* directory.

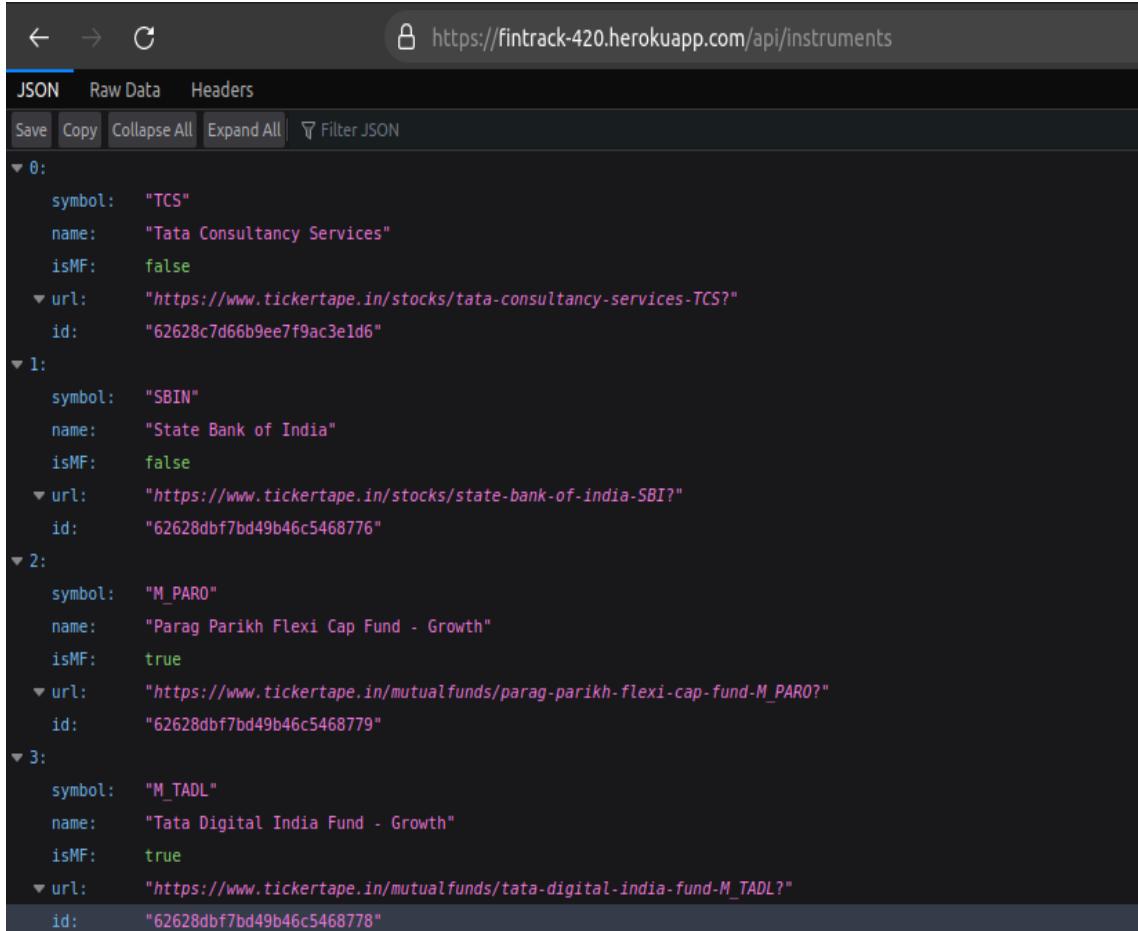


start-heroku is a *NODE_ENV* value that is set in *package.json* and it was also used in the *config.js* file to specify the fact that we'll be reading the environment variables from the *process.env* JSON.

```
"scripts": {
  "start": "NODE_ENV=production node index.js",
  "start-heroku": "NODE_ENV=production-heroku node index.js",
  "test": "NODE_ENV=test jest --verbose --runInBand",
  "local-test": "NODE_ENV=local-test jest --verbose --runInBand",
  "dev": "NODE_ENV=development nodemon index.js",
  "lint": "eslint ."
},|
```

Now pushing these changes to your Github repo will automatically trigger the Action and build the Heroku app.

Verifying that the Heroku app is indeed serving the backend.



The screenshot shows a browser developer tools Network tab with a JSON response. The URL is <https://fintrack-420.herokuapp.com/api/instruments>. The response is a JSON array of four objects (0, 1, 2, 3) containing information about financial instruments:

```
0:
  symbol: "TCS"
  name: "Tata Consultancy Services"
  isMF: false
  url: "https://www.tickertape.in/stocks/tata-consultancy-services-TCS?"
  id: "62628c7d66b9ee7f9ac3e1d6"

1:
  symbol: "SBIN"
  name: "State Bank of India"
  isMF: false
  url: "https://www.tickertape.in/stocks/state-bank-of-india-SBI?"
  id: "62628dbf7bd49b46c5468776"

2:
  symbol: "M_PARO"
  name: "Parag Parikh Flexi Cap Fund - Growth"
  isMF: true
  url: "https://www.tickertape.in/mutualfunds/parag-parikh-flexi-cap-fund-M_PARO?"
  id: "62628dbf7bd49b46c5468779"

3:
  symbol: "M_TADL"
  name: "Tata Digital India Fund - Growth"
  isMF: true
  url: "https://www.tickertape.in/mutualfunds/tata-digital-india-fund-M_TADL?"
  id: "62628dbf7bd49b46c5468778"
```

Netlify Configuration

Create a Netlify account and link your Github repository in order to deploy the frontend app.

Update the Deploy build settings as follows,

Build settings

Base directory: `frontend`

For monorepos or sites built from a subdirectory of a repository, the directory to change to before starting a build.

Build command: `npm run build`

Publish directory: `frontend/build`

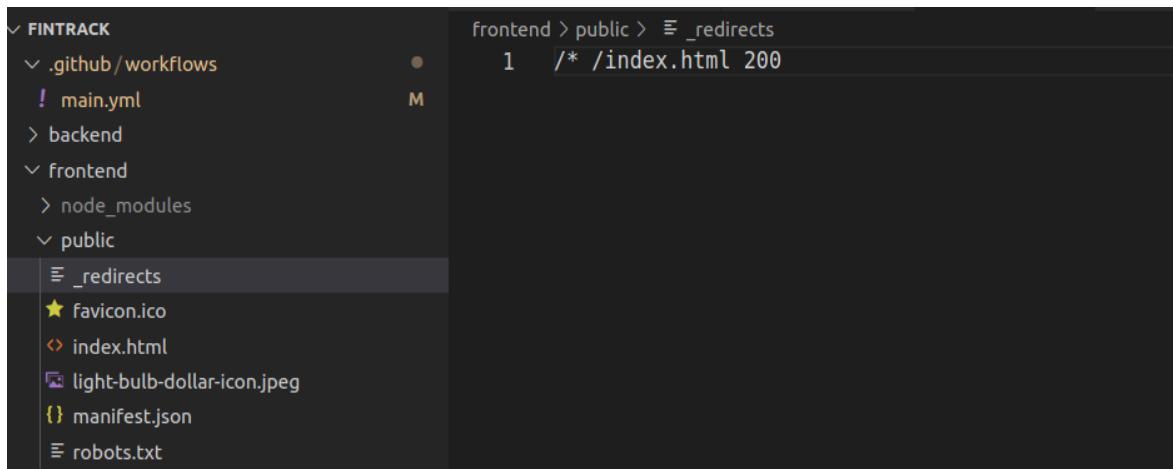
Builds:

- Activate builds**
Netlify will build your site according to your continuous deployment settings when you push to your Git provider.
- Stop builds**
Netlify will never build your site. You can build locally via the CLI and then publish new deploys manually via the CLI or the API.

The *Publish directory* is `frontend/build` otherwise Netlify will try to read the *build* directory in the root directory itself which is not where `npm run build` will save the *build* directory and crash the app.

Now the frontend will also automatically deploy but there will be a problem with React Router. You cannot directly access any routed paths such as <https://fintrack-420.netlify.app/search> without going through the home page first.

For this you have to add a file called `_redirects` under the *public* folder of your React app which also contains the `index.html` file.



This basically tells Netlify that if the URL to serve has anything after the '/' then redirect to rendering `index.html` with a 200 status code.

And now React Router will be able to pick up the specified URL and appropriately route the user to the correct page without having to go through the home page, nor result in an error.

We will also have to add the same environment variables that we have used in the `.env.development` file in the frontend in order for this deployment to work.

Environment

Control the environment your site builds in and/or gets deployed to.

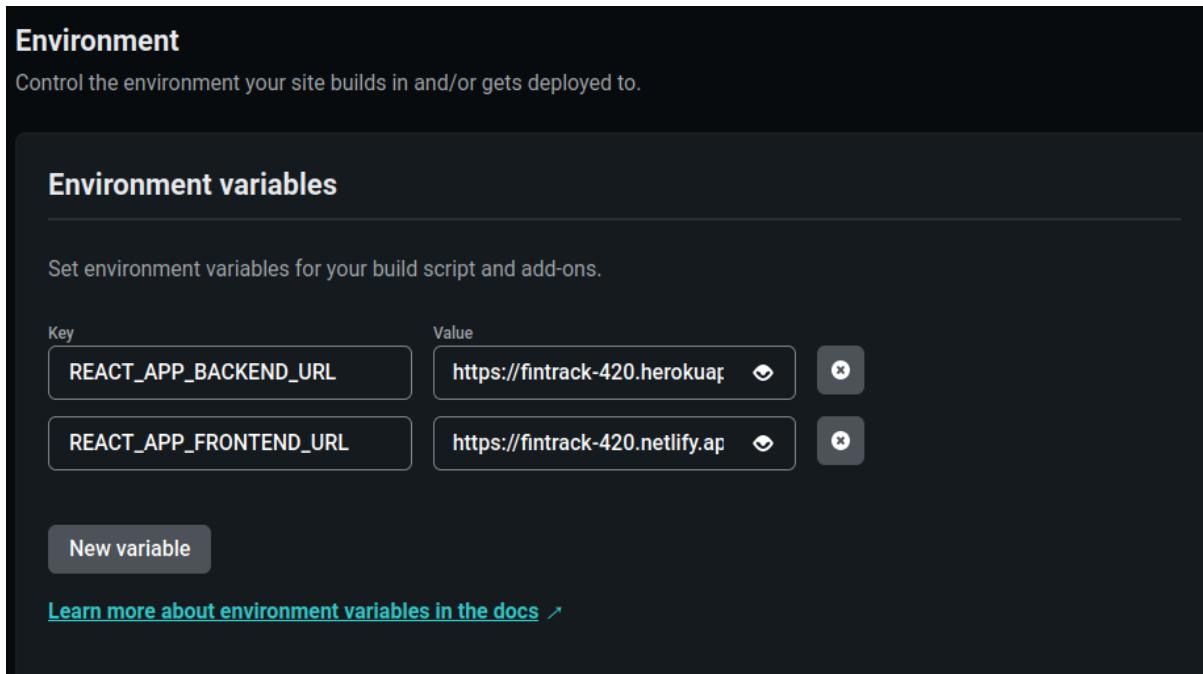
Environment variables

Set environment variables for your build script and add-ons.

Key	Value	Eye icon	X icon
REACT_APP_BACKEND_URL	https://fintrack-420.herokuapp.com	eye	x
REACT_APP_FRONTEND_URL	https://fintrack-420.netlify.app	eye	x

New variable

[Learn more about environment variables in the docs ↗](#)



And that does it for the entire Heroku + Netlify configuration!



4.10. Monitoring - ELK Stack

"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. ELK stack gives us the ability to aggregate logs from all the systems and applications, analyze these logs, and create visualizations for application and infrastructure monitoring, faster troubleshooting, security analytics, and more.

To use the ELK stack we first require the logs that are generated from the application. Because we have used Docker volumes for persisting the logs as specified in the Docker Compose file, we can get the logs at the path `/home/backend/logs/access.log`.

Now we can upload this log file into our Elastic Cloud cluster.

The screenshot shows the Elastic Cloud interface for creating a new data view. The 'Data view name' is set to 'fintrack'. Under 'Index settings', there is a 'number_of_shards': 1 setting. The 'Mappings' section contains the following JSON code:

```
5  },  
6  "content_length": {  
7    "type": "keyword"  
8  },  
9  "http_request": {  
10   "type": "keyword"  
11 },  
12  "message": {  
13   "type": "text"  
14 },  
15  "path": {  
16   "type": "keyword"  
17 },  
18  "request_data": {  
19   "type": "keyword"  
20 },  
21  "response_time": {  
22   "type": "double"  
23 },  
24  "status_code": {  
25   "type": "long"  
26 }  
27 }  
28 }
```

The 'Ingest pipeline' section contains the following JSON code:

```
1  {  
2    "description": "Ingest pipeline created by text  
3      structure finder",  
4    "processors": [  
5      {  
6        "grok": {  
7          "field": "message",  
8          "patterns": [  
9            "%{WORD:http_request} %{URIPATHPARAM:path} %{BASE10NUM:  
10           :status_code} %{DATA:content_length}  
11           - %{BASE16FLOAT:response_time} ms %{GREEDYDATA:request_data}"  
12        ]  
13      }  
14    },  
15    {  
16      "date": {
```

A large blue 'Import' button is located at the bottom of the interface.

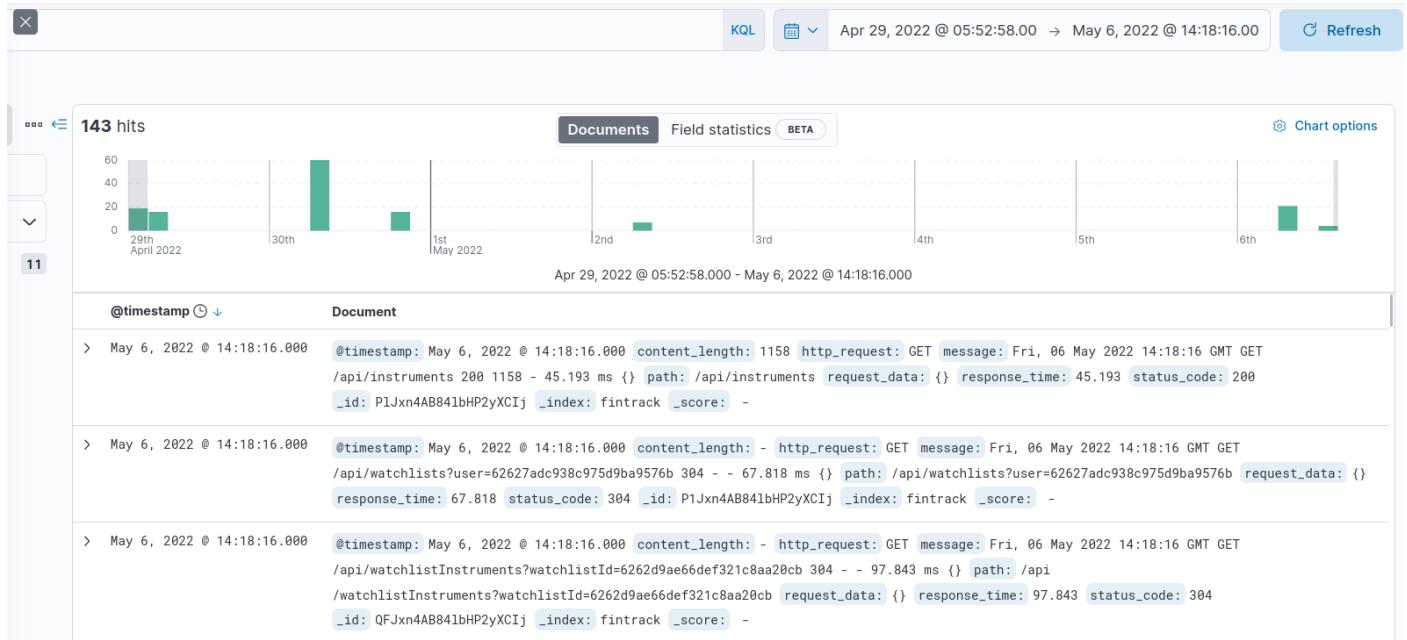
To infer the proper fields from the log file we have to use the following Grok pattern

```
"CUSTOM_TIMESTAMP": "%{DAY}, %{MONTHDAY} %{MONTH} %{YEAR} %{TIME}"
```

```
"%{CUSTOM_TIMESTAMP:timestamp} GMT %{WORD:http_request}  
%{URIPATHPARAM:path} %{BASE10NUM:status_code} %{DATA:content_length}  
- %{BASE16FLOAT:response_time} ms %{GREEDYDATA:request_data}"
```

Visualizations

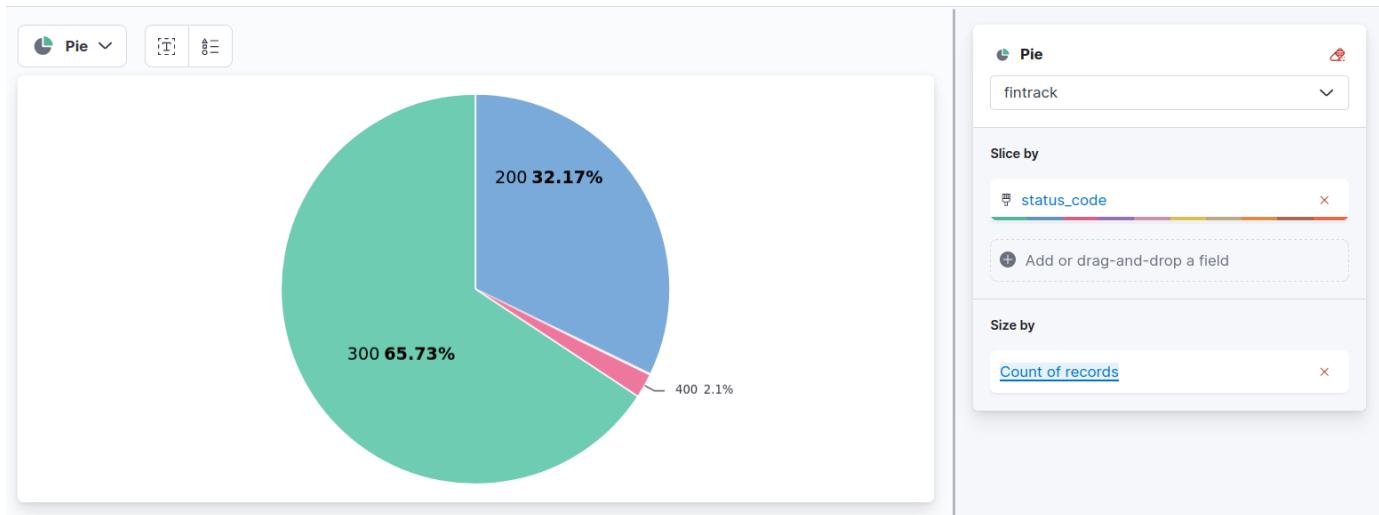
1. Time Series View



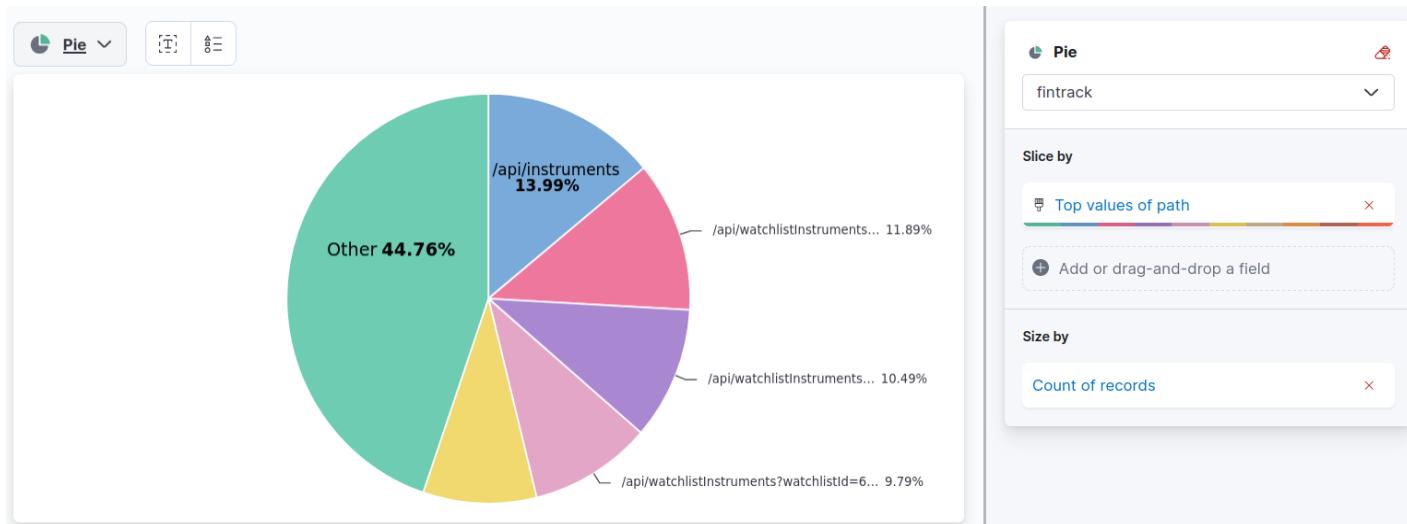
2. Visualizing http_request field



3. Visualizing status_code field



4. Visualizing path field



We also created a Kibana Dashboard containing these three visualizations.

5. Experimental Setup

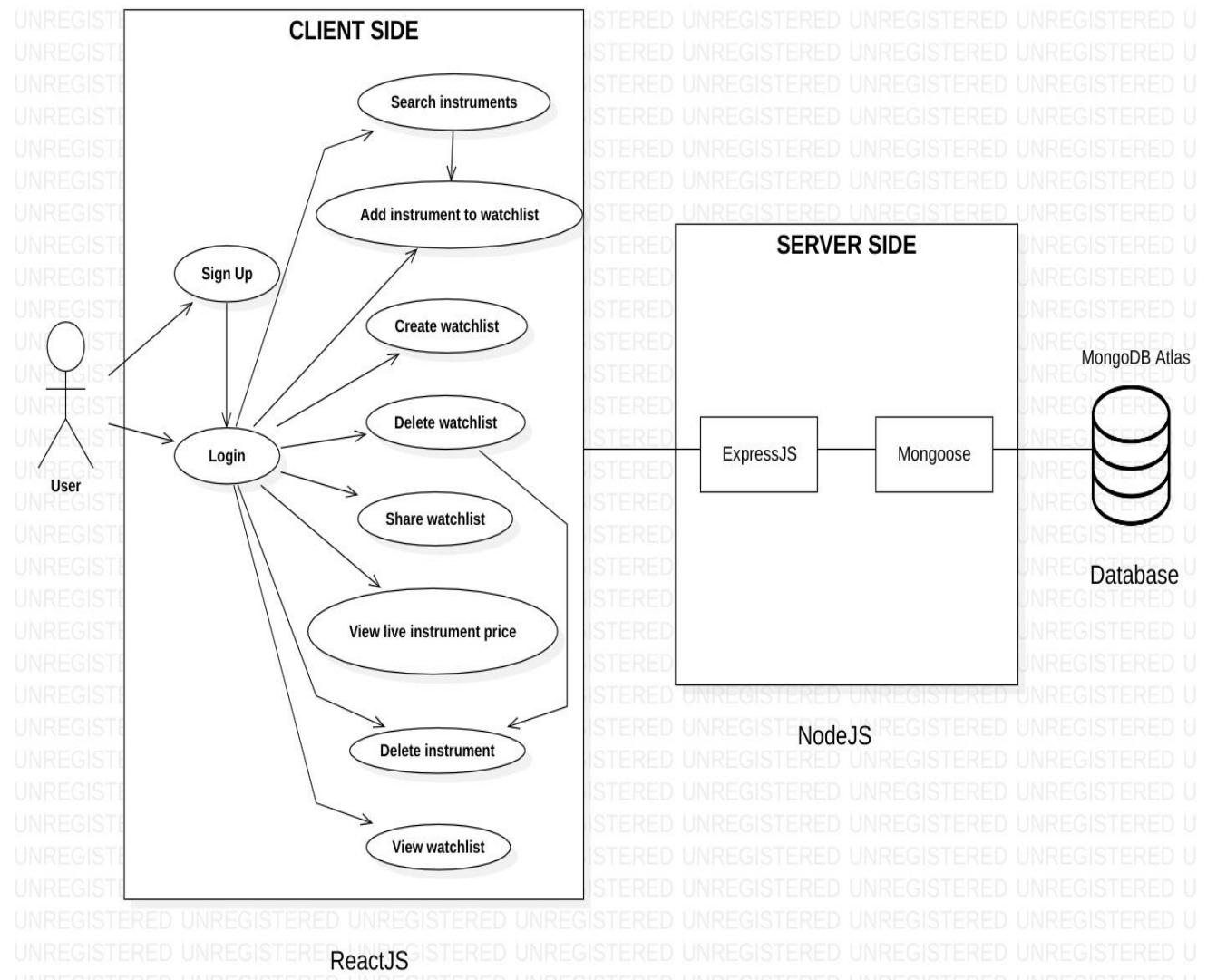
5.1. Functional Requirements:

1. Users can register and create their account.
2. Users can create watchlists without creating duplicate entries.
3. Users can view their watchlists and its contained instruments.
4. Users can edit the desired watchlist (delete the instruments or add the instruments) as per their wish.
5. Users can search for the instruments for adding them to their watchlist.
6. Each watchlist will have a share button that will send a publicly shareable link to anyone that visits that URL.
7. To view a shared watchlist, a user must be logged in to a Fintrack account, which can be any account but a login is required.
8. Watchlists cannot be amalgamated. ‘Mutual Funds’ watchlists must only contain Mutual Funds, whereas ‘Stocks’ watchlists must contain only Stocks.

5.2. Non-Functional Requirements:

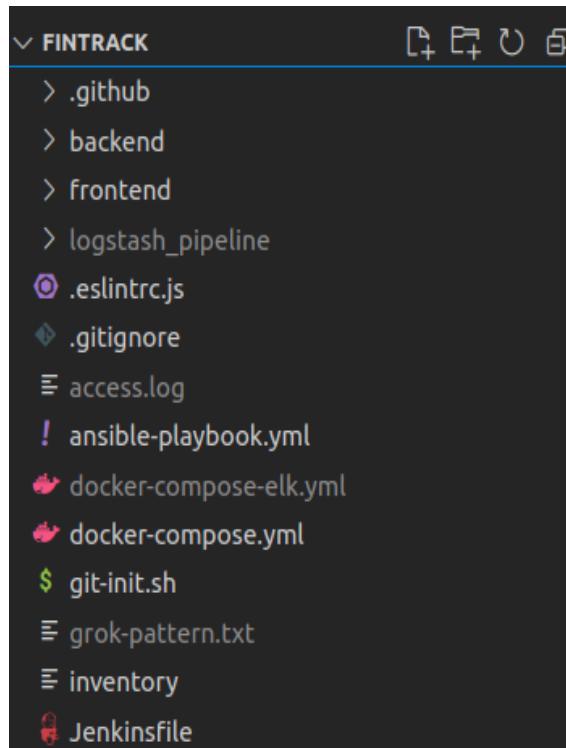
1. **Portability:** To ensure portability, Docker images have been built for the frontend and backend.
2. **Scalability:** The database is created on MongoDB Atlas and in case of high traffic, Atlas will automatically handle this via scaling and thus scalability is achieved.
3. **Security:** To improve security, JWT tokens are being used for setting up a session and checking the authorization.
4. **User Friendly:** The website has to be user friendly and error messages should pop up when relevant.
5. **Performance:** The performance of the website should not degrade in case of multiple users.

5.3. Architectural Diagram:



5.4. Code Walkthrough

The folder structure that we have followed includes the client-side *frontend*/ and the server-side *backend*/ with subfolders within them to ensure proper flow of data in the website.



A screenshot of a file explorer window titled "FINTRACK". The tree view shows the following structure:

- .github
- backend
- frontend
- logstash_pipeline
- .eslintrc.js (highlighted with a purple circle)
- .gitignore
- access.log
- ansible-playbook.yml
- docker-compose-elk.yml
- docker-compose.yml
- git-init.sh
- grok-pattern.txt
- inventory
- Jenkinsfile

Directory structure for the project

5.4.1. Backend

Directory structure for the Backend

The screenshot shows a file explorer window with the following directory structure:

- FINTRACK (root)
 - backend
 - controllers
 - JS instruments.js
 - JS login.js
 - JS users.js
 - JS watchlistInstruments.js
 - JS watchlists.js
 - logs
 - models
 - JS instrument.js
 - JS user.js
 - JS watchlist.js
 - JS watchlistInstrument.js
 - node_modules
 - requests
 - tests
 - utils
 - JS config.js
 - JS logger.js
 - JS middleware.js
 - .dockerignore
 - JS app.js
 - Dockerfile
 - ! env-enc.yaml
 - ! env-local.yaml
 - env.j2
 - JS index.js
 - { package-lock.json
 - { package.json
 - H Procfile

We have incorporated MVC architecture (Model View Controller) in the project.

Model:

- It is known as the lowest level which means it is responsible for maintaining data.
- The Model is actually connected to the database so anything you do with data - adding or retrieving data is done in the Model component.
- It responds to the controller requests because the controller never talks to the database by itself. The Model talks to the database back and forth and then it gives the needed data to the controller.

Here are the Models that we have used in our project.

1. User Model

```
const mongoose = require('mongoose')
const uniqueValidator = require('mongoose-unique-validator')

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, minLength: 3, unique: true },
  name: { type: String, required: true },
  passwordHash: { type: String, required: true },
})
```

2. Instrument Model

```
const instrumentSchema = new mongoose.Schema({
  symbol: { type: String, required: true, unique: true },
  name: String,
  isMF: { type: Boolean, required: true },
  url: { type: String, required: true }
})
```

3. Watchlist Model

```
const watchlistSchema = new mongoose.Schema({
  name: { type: String, required: true },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  isMF: { type: Boolean, required: true }
})
```

4. WatchlistInstrument Model

```
const mongoose = require('mongoose')

const watchlistInstrumentSchema = new mongoose.Schema({
  watchlist: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Watchlist',
    required: true
  },
  instrument: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Instrument',
    required: true
  }
})
```

View:

- Data representation is done by the View component.
- It actually generates UI or user interface for the user.
- So in web applications when you think of the View component just think of it as the HTML/CSS part.
- Views are created by the data which is collected by the model component but this data isn't taken directly from the Model and instead obtained through the Controller.
- View only speaks to the Controller.

Controller:

- It's known as the main man because the controller is the component that enables the interconnection between the View and the Model, so it acts as an intermediary.
- The Controller doesn't have to worry about handling data logic, it just tells the Model what to do.
- After receiving data from the Model, the Controller processes it and sends the information to the View.
- **Note:** Views and Models cannot talk directly.

Express & NodeJS do all the functional programming and will be used to write the Business Tier.

This tier represents the Application Server that acts as the bridge of communication between the Client and Database. This tier will serve the React components to the

user's device and accept HTTP requests from the user and follow with the appropriate response.

Here is an overview of the Controllers in our project.

1. Instruments Controller

```
const instrumentRouter = require('express').Router()
const Instrument = require('../models/instrument')

/*
 * Return all instruments stored in the Database
 */
instrumentRouter.get('/', async (request, response) => {
  const instruments = await Instrument.find({})
  response.status(200).json(instruments)
})

module.exports = instrumentRouter
```

2. Users Controller

Handles the signup of users into our web-app.

```
/*
 * Create a new user in the database using the data provided in the request
 * Do validation checks in the password specified so that it satisfies the criteria for a good password
 * Encrypt the password using bcrypt to obtain the hash value and store this data in the DB
 * Return the saved user (which will also include user id added by MongoDB) back
 * as a response with 201 Status (Content Created)
*/
userRouter.post('/', async (request, response) => {
  const body = request.body

  if (body.password.length < 3)
    return response.status(400).json({ error: 'password should have length atleast 3' })

  const saltRounds = 10
  const passwordHash = await bcrypt.hash(body.password, saltRounds)

  const userObject = new User({
    username: body.username,
    passwordHash,
    name: body.name
  })

  const savedUser = await userObject.save()

  const userForToken = {
    username: savedUser.username,
    id: savedUser.id
  }

  const token = jwt.sign(userForToken, config.SECRET)

  response.status(201).send({ token, username: savedUser.username, name: savedUser.name, id: savedUser.id })
})

module.exports = userRouter
```

3. Login Controller

```
const jwt = require('jsonwebtoken')
const bcrypt = require('bcrypt')
const User = require('../models/user')
const loginRouter = require('express').Router()
const config = require('../utils/config')

/*
* For the provided username, check whether such a user exists in the DB
* If there is a user, then check whether the hash of the password sent and stored passwordHash match
* If they do, then login is valid and generate a token using the username, user id & SECRET env variable
* as input to the jwt.sign() method
* Return the token, username, name and user id upon successful login with 200 status code
* Otherwise return 401 status code with error message in response
*/
loginRouter.post('/', async (request, response) => {
  const body = request.body

  const user = await User.findOne({ username: body.username })
  const passwordCorrect = user === null? false: await bcrypt.compare(body.password, user.passwordHash)

  if (!passwordCorrect)
    return response.status(401).json({ error: 'Invalid username or password' })

  const userForToken = {
    username: user.username,
    id: user.id
  }

  const token = jwt.sign(userForToken, config.SECRET)

  response.status(200).send({ token, username: user.username, name: user.name, id: user.id })
})

module.exports = loginRouter
```

4. Watchlists Controller

```
/*
* Create a new watchlist in the database with data provided in request body.
* Check whether user has sent jwt token in packet header
* Otherwise, return 401 status code with error "invalid token"
* Return watchlist name, type, user-id upon successful creation with 201 status code
*/
watchlistRouter.post('/', async (request, response) => {
  const body = request.body
  const user = request.user

  if(!user)
    return response.status(401).json({ error: 'token missing or invalid' })

  const watchlistObject = new Watchlist({
    name: body.name,
    isMF: body.isMF,
    user: user._id
  })

  const savedWatchlist = await watchlistObject.save()
  response.status(201).json(savedWatchlist)
})
```

5. WatchlistInstruments Controller

```
/*
 * Return all watchlist-instruments which belong to given watchlist-id
 * Check if watchlist-id exists. If it does, populate instrument database in a new object using mongoose' populate() method.
 * Otherwise, return 401 status code with error "Invalid Watchlist-id"
*/
watchlistInstrumentRouter.get('/', async (request, response) => {
  const watchlistId = request.query.watchlistId

  const user = request.user
  if(!user)
    return response.status(401).json({ error: 'token missing or invalid' })

  const watchlistInstrumentsList = await WatchlistInstrument.find({watchlist: watchlistId}).populate('instrument')

  if(!watchlistInstrumentsList)
    return response.status(404).json({ error: 'Invalid watchlist' })

  const modifiedWatchlistInstrumentsList = watchlistInstrumentsList.map((watchlistInstrument) =>{
    return {
      symbol: watchlistInstrument.instrument.symbol,
      name: watchlistInstrument.instrument.name,
      isMF: watchlistInstrument.instrument.isMF,
      url: watchlistInstrument.instrument.url,
      instrumentId: watchlistInstrument.instrument.id,
      watchlistId: watchlistInstrument.watchlist,
      id: watchlistInstrument.id
    }
  })
  response.status(200).json(modifiedWatchlistInstrumentsList)
})
```

utils directory

Contains middleware, config and logger files to support controller's functionalities.

tests directory

Contains the supertest test files as described in the SDLC section.

Here is a snippet of the `watchlists_api.test.js` file which shows how we use the `beforeEach()` method to login as an existing user and perform authorized operations in the tests with the JWT token.

```
let user = null

beforeEach(async () => [
  // Increasing timeout otherwise sometimes a timeout error can wreck the whole testing phase
  jest.setTimeout(30000)

  await User.deleteMany({})
  await User.insertMany(helper.initialUsers)
  await Watchlist.deleteMany({})

  user = await api
    .post('/api/login')
    .send(helper.loginUser)

  for (let watchlist of helper.initialWatchlists) {
    await api
      .post('/api/watchlists')
      .send(watchlist)
      .set('Authorization', `Bearer ${user.body.token}`)
  }
])

describe('viewing a user\'s watchlists', () => {
  test('watchlist object has id property instead of _id', async () => {
    const response = await api
      .get(`/api/watchlists?user=${user.body.id}`)
      .set('Authorization', `Bearer ${user.body.token}`)

    expect(response.body[0].id).toBeDefined()
    expect(response.body[0]._id).toBeUndefined()
    expect(response.body[0].__v).toBeUndefined()
  })

  test('total watchlists contained are 4', async () => {
    const response = await api
      .get(`/api/watchlists?user=${user.body.id}`)
      .set('Authorization', `Bearer ${user.body.token}`)
```

package.json

This file is the heart of our Node project. It records important metadata about our project which is required before publishing to npm and also defines functional attributes of our project that npm uses to install dependencies, run scripts, and identify the entry point to our package.

```
{  
  "name": "fintrack",  
  "version": "1.0.0",  
  "description": "Create shareable investment watchlists!",  
  "main": "index.js",  
  "scripts": {  
    "start": "NODE_ENV=production node index.js",  
    "start-heroku": "NODE_ENV=production-heroku node index.js",  
    "test": "NODE_ENV=test jest --verbose --runInBand",  
    "local-test": "NODE_ENV=local-test jest --verbose --runInBand",  
    "dev": "NODE_ENV=development nodemon index.js",  
    "lint": "eslint ."  
  },  
  "engines": {  
    "node": "16.x"  
  },  
  "author": "Jasvin James",  
  "license": "ISC",  
  "dependencies": {  
    "bcrypt": "^5.0.1",  
    "cors": "^2.8.5",  
    "dotenv": "^14.3.0",  
    "express": "^4.17.2",  
    "express-async-errors": "^3.1.1",  
    "jsonwebtoken": "^8.5.1",  
    "mongoose": "^6.1.7",  
    "mongoose-unique-validator": "^3.0.0",  
    "morgan": "^1.10.0",  
    "run": "^1.4.0",  
    "yenv": "^3.0.1"  
  },  
  "devDependencies": {  
    "eslint": "^8.7.0",  
    "jest": "^27.4.7",  
    "nodemon": "^2.0.15",  
    "supertest": "^6.2.3"  
  },  
  "jest": {  
    "testEnvironment": "node"  
  }  
}
```

API Documentation

Endpoint	HTTP method	Input	Description
/api/users	GET	–	Get all the users from the database
/api/users	POST	username, name, password	User signup → Store user details in database → Do validation checks in the password specified → Store the encrypted password in the database → Return token (JWT specification), name, username, id
/api/login	POST	username, password	User login → Check username → do password validation → Return token, name, username, id
/api/watchlists/:watchlistid	GET	watchlist-id	Return watchlist data with given watchlist ID
/api/watchlists?user =user.id	GET	user-id	Return all the watchlists of the given user
/api/watchlists	POST	watchlist-name , type, user-id	Check JWT token → Create watchlist → Return watchlist name, type, user-id
/api/watchlists/:watchlistid	DELETE	watchlist-id	Delete watchlist with given watchlist-id
/api/watchlistInstruments	POST	Watchlist-id, instrument-id	Check if watchlist-instrument already exists in the given watchlist → Check if added instrument is of same type as as watchlist → Return symbol, name, type, url, id , watchlist-id, instrument-id
/api/watchlistInstruments?watchlistId=watchlist.id	GET	watchlist-id	Return all watchlist-instruments (symbol, name, type, url, id , watchlist-id, instrument-id, watchlistInstrument-id) which belong to given watchlist-id
/api/watchlistInstruments/:watchlistInstrument.id	DELETE	watchlistInstrument-id	Delete the watchlist-instrument with given watchlistInstrument-id
/api/watchlistInstruments?watchlistId=watchlist.id	DELETE	watchlist-id	Delete all instruments which belong to given watchlist-id
/api/instruments	GET	–	Return all the instruments stored in database

5.4.2. Frontend

Directory structure for frontend

```
└ FINTRACK
  └ .github
  └ backend
  └ frontend
    └ node_modules
    └ public
  └ src
    └ assets
    └ components
      └ Instrument.js
      └ LoginForm.js
      └ LoginMessage.js
      └ NavBar.js
      └ Notification.js
      └ Search.js
      └ SignupForm.js
      └ Toggleable.js
      └ Watchlist.js
    └ services
      └ login.js
      └ watchlists.js
    └ App.css
    └ App.js
    └ config.js
    └ index.css
    └ index.js
```

We use functional components, React state and React hooks in order to build the frontend. In order to enable routing we have used React Router (BrowserRouter).

Components

Start of the App component (App.js)

```
const App = () => {
  // user state will store the logged in user object, if no login has been done yet then it will be null
  const [ user, setUser ] = useState(null)
```

How React Router is used to match routes appropriately

```
// A useRouteMatch that will check whether the current page's route matches the pattern /watchlists/:id
// match = Object that contains various information about route matched including the "id" in path,
// if route matched successfully
// Else, match = null
const match = useRouteMatch('/watchlists/:id')
```

Effect Hook usage

```
// Effect hook that fetches all the instrument data from the backend and sets the appropriate state
useEffect(() => {
  watchlistService
    .getInstrumentData()
    .then(data => {
      setInstruments(data)
    })
}, [])
```

Effect Hook + React Router to fetch correct watchlist data

```
// Fetch specific watchlist data only if we are single watchlist view mode, i.e. /watchlists/:id route, so match !== null
// Providing "match" as a parameter in the dependency array of the useEffect hook
// results in an infinite loop of getWatchlistData() being called
// and crashing the system, so that's why dependency array is kept as empty
useEffect(() => {
  if (match && user !== null) {
    watchlistService
      .getWatchlistData(match.params.id)
      .then(data => setWatchlist(data))
  }
}, [user])
```

JSX that the App component returns

```
return (
  <div>
    <div className='text-center page-header p-2 regular-text-shadow regular-shadow'>
      <div className='display-4 font-weight-bold'>
        F<small>IN</small>T<small>RACK</small> {' '}
        <i className='fa fa-cog fa-spin' aria-hidden='true'></i> {' '}
      </div>
      <h5>Share investment watchlists easily!</h5>
    </div>

    <Notification notification={notification} type={notificationType} />

    [
      user === null && showLoginForm &&
      <LoginForm startLogin={handleLogin} showLoginForm={showLoginForm} setShowLoginForm={setShowLoginForm}>/>
    ]
    {
      user === null && showLoginForm === false &&
      <SignupForm startSignup={handleLogin} showLoginForm={showLoginForm} setShowLoginForm={setShowLoginForm}>/>
    }

    {
      user !== null &&
      <NavBar user={user} setUser={setUser}>/>
    }

    {
      user !== null &&
      <Switch>
        <Route path='/watchlists/:id'>
          <Watchlist watchlist={watchlist} removeWatchlist={null} removeWatchlistInstrument={null} />
        </Route>

        <Route path='/search'>
          {
            // If watchlists or instruments haven't loaded or if even one watchlist doesn't have instruments
            // property loaded then redirect to home page instead of remaining on search page
            // It's a stopgap solution but it is what it is :(
            watchlists && instruments && checkWhetherWatchlistInstrumentsLoaded(watchlists)?
              <Search instruments={instruments} watchlists={watchlists} addToWatchlist={addToWatchlist}>/>
              <Redirect to='/' />
            }
          </Route>
        
```



```
        <Route path='/'>
          {/* <Toggleable buttonLabel={'Click Here!'} ref={watchlistFormRef}> */}
            <WatchlistForm createWatchlist={createWatchlist} />
          {/* </Toggleable> */}

          <Watchlists
            watchlists={watchlists}
            removeWatchlist={removeWatchlist}
            removeWatchlistInstrument={removeWatchlistInstrument}
          />
        </Route>

      </Switch>
    ]
  
```

Services

Login & Signup Service

```
import axios from 'axios'
const config = require('../config')

const loginBaseUrl = `${config.BACKEND_URL}/api/login`
const signupBaseUrl = `${config.BACKEND_URL}/api/users/`

const login = async (credentials, isLogin) => {
  let response = ""

  if (isLogin)
    response = await axios.post(loginBaseUrl, credentials)
  else
    response = await axios.post(signupBaseUrl, credentials)

  return response.data
}
```

Create & Delete Watchlist Services with Token Authentication

```
const createWatchlist = async (watchlist) => {
  setToken()
  const config = {
    headers: {
      Authorization: `Bearer ${token}`
    }
  }

  const response = await axios.post(watchlistUrl, watchlist, config)
  return response.data
}

const deleteWatchlist = async (watchlist) => {
  setToken()
  const config = {
    headers: {
      Authorization: `Bearer ${token}`
    }
  }

  const response = await axios.delete(`/${watchlistUrl}/${watchlist.id}`, config)
  return response.data
}
```

Similar structure is then followed for all other required services.

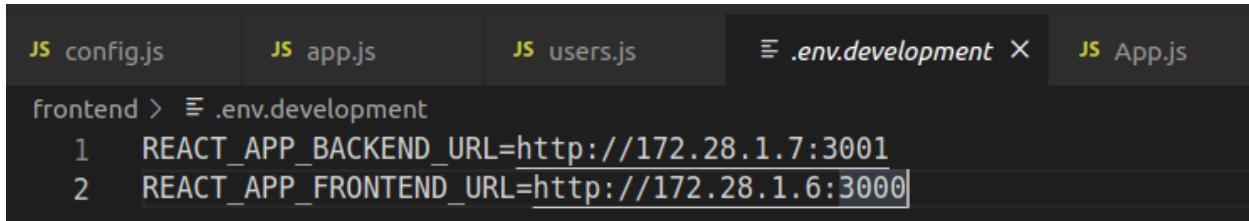
URL Management with Environment variables: config.js

```
/*
  React Apps created with Create-React-App can use environment variables without the dotenv package
  But such variable names must begin with REACT_APP and that's why
  the environment variable is named such here
*/

const BACKEND_URL = process.env.REACT_APP_STAGE === 'local'? 'http://localhost:3001': process.env.REACT_APP_BACKEND_URL
const FRONTEND_URL = process.env.REACT_APP_STAGE === 'local'? 'http://localhost:3000': process.env.REACT_APP_FRONTEND_URL

export { BACKEND_URL, FRONTEND_URL }
```

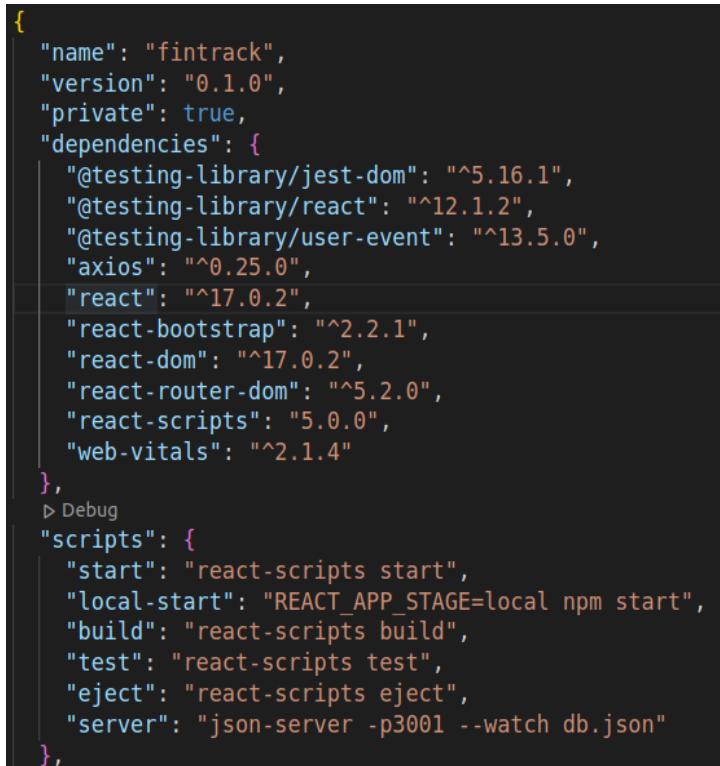
.env.development file



```
JS config.js      JS app.js      JS users.js      ≡ .env.development X      JS App.js

frontend > ≡ .env.development
1   REACT_APP_BACKEND_URL=http://172.28.1.7:3001
2   REACT_APP_FRONTEND_URL=http://172.28.1.6:3000
```

How the REACT_APP_STAGE variable is used in the *package.json* scripts,



```
{
  "name": "fintrack",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.1",
    "@testing-library/react": "^12.1.2",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^0.25.0",
    "react": "^17.0.2",
    "react-bootstrap": "^2.2.1",
    "react-dom": "^17.0.2",
    "react-router-dom": "^5.2.0",
    "react-scripts": "5.0.0",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "local-start": "REACT_APP_STAGE=local npm start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "server": "json-server -p3001 --watch db.json"
  }
}
```

We know the IP addresses of the backend and frontend because these are made static in the Docker-Compose of the backend and frontend containers respectively.

Testing

Snippet of the `Watchlist.test.js` file which tests the `Watchlist` component along with mock data.

```
describe('<Watchlist />', () => {
  const removeWatchlistInstrument = jest.fn()
  const removeWatchlist = jest.fn()

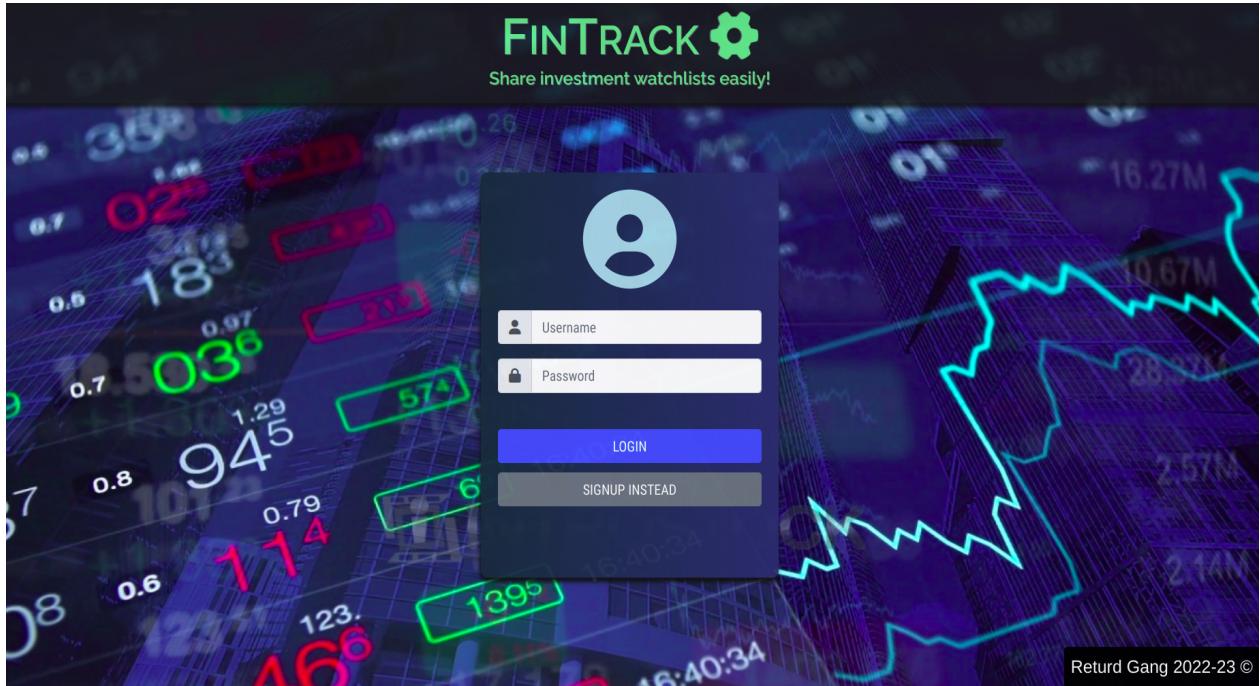
  const watchlist = {
    name: 'Test Watchlist 1',
    isMF: false,
    user: {
      username: 'Joe',
      name: 'Joe Mama'
    },
    instruments: [
      {
        "id": "1",
        "symbol": "TCS",
        "name": "Tata Consultancy Services",
        "isMF": false,
        "url": "https://www.tickertape.in/stocks/tata-consultancy-services-TCS?"
      },
      {
        "id": "2",
        "symbol": "INFY",
        "name": "Infosys",
        "isMF": false,
        "url": "https://www.tickertape.in/stocks/infosys-INFY?"
      }
    ]
  }

  test('watchlist rendered with name and watchlist instruments are not visible', () => {
    const { container } = render(
      <Watchlist watchlist={watchlist} removeWatchlist={removeWatchlist} removeWatchlistInstrument={removeWatchlistInstrument} />
    )
    expect(container).toHaveTextContent('Test Watchlist 1')
    expect(screen.queryByText('TCS')).toBeNull()
  })
})
```

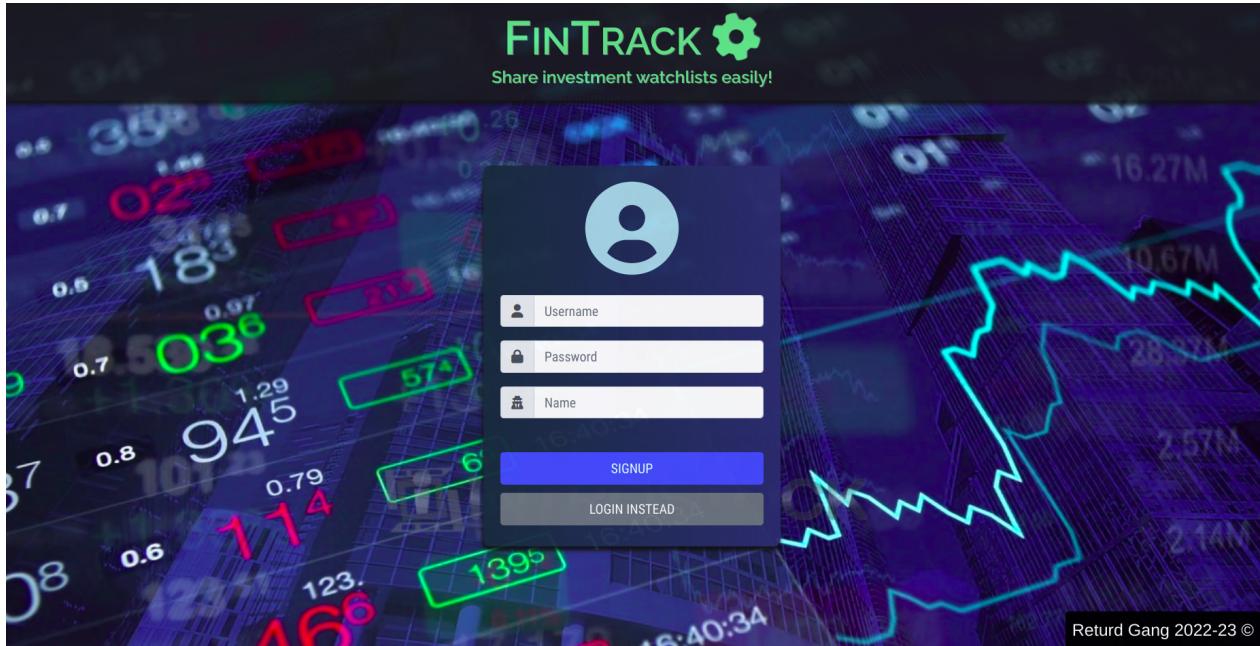
Similar testing format is used for other component tests with `react-testing-library`.

6. Result and Discussion

6.1. Login Page



6.2. Signup Page



6.3. Home Page

The screenshot shows the FINTRACK home page. At the top, there is a navigation bar with "Welcome, Devendra", "Home", "Search", and "Logout". Below the navigation bar, a modal window titled "Create a New Watchlist" is open. It has a "Watchlist Name:" input field containing "Enter watchlist name", two radio button options ("Mutual Funds Watchlist" and "Stocks Watchlist"), and a green "Submit" button. In the background, there is a dark-themed dashboard with a chart showing stock prices and volumes. Below the modal, a section titled "Your Watchlists" lists two entries: "messi" (Mutual Funds Watchlist) and "Dev" (Stocks Watchlist). Each entry has a dropdown arrow, a gear icon, and a trash bin icon. In the bottom right corner of the dashboard, there is a copyright notice: "Returd Gang 2022-23 ©".

6.4. Watchlist Form

The screenshot shows the FINTRACK watchlist creation form. The title is "Create a New Watchlist". The "Watchlist Name:" field contains "TrialWatchlist". The "Stocks Watchlist" radio button is selected. A green "Submit" button is at the bottom. The background features a stock market chart. After submission, a success message "Watchlist created successfully" appears below the form.

Watchlist created successfully

The screenshot shows the confirmation message "A new Stocks watchlist "TrialWatchlist" has been successfully created" displayed prominently in a green bar at the top. Below this, the FINTRACK logo and the tagline "Share investment watchlists easily!" are visible. The navigation bar at the bottom includes "Welcome, Devendra", "Home", "Search", and "Logout".

6.5. Delete a Watchlist

Your Watchlists

- messi
Mutual Funds Watchlist
- Dev
Stocks Watchlist
- TrialWatchlist
Stocks Watchlist

Returd Gang 2022-23 ©

Mutual Funds Watchlist
Stocks Watchlist

fintrack-420.netlify.app says
You're about to delete the "TrialWatchlist" watchlist

Cancel OK

Submit

Your Watchlists

Watchlist deleted successfully

FINTRACK Share investment watchlists easily!

Successfully deleted the "TrialWatchlist" watchlist

Welcome, Devendra Home Search Logout

6.6. Search Instruments

The screenshot shows a search interface for stocks and mutual funds. At the top, there is a header with 'Welcome, Devendra' and navigation links for 'Home' and 'Search'. On the right, there is a 'Logout' button. Below the header is a search bar with the placeholder 'Search for Stocks or MFs'. The main area is titled 'Search Results' and lists several instruments:

- TCS - Tata Consultancy Services
- SBIN - State Bank of India
- M_PARO - Parag Parikh Flexi Cap Fund - Growth
- M_TADL - Tata Digital India Fund - Growth
- M_AXLB - Axis Bluechip Fund
- INFY - Infosys
- LT - Larsen & Toubro Ltd

Each instrument entry includes a small icon and a 'Add to Watchlist' button with a dropdown menu. In the bottom right corner, there is a copyright notice: 'Return Gang 2022-23 ©'.

6.7. Adding an Instrument in a Watchlist

The screenshot shows the FINTRACK application interface. At the top, it displays the logo 'FINTRACK' with the tagline 'Share investment watchlists easily!'. The header includes 'Welcome, Devendra' and navigation links for 'Home' and 'Search', along with a 'Logout' button. Below the header is a search bar containing the query 'm_tadl'. The main area is titled 'Search Results' and shows one result: 'M_TADL - Tata Digital India Fund - Growth'. To the right of this result is a 'Add to Watchlist' button, which has a dropdown menu open, showing the option 'messi' highlighted.

Successfully added instrument to watchlist

The screenshot shows the FINTRACK application interface again. At the top, it displays the logo 'FINTRACK' with the tagline 'Share investment watchlists easily!'. The header includes 'Welcome, Devendra' and navigation links for 'Home' and 'Search', along with a 'Logout' button. A green horizontal bar at the top of the main content area displays the message 'Successfully added "M_TADL" to watchlist "messi"'. Below this bar, the search interface from the previous screenshot is visible, showing the same search results and the 'messi' watchlist entry.

6.8. View a Watchlist

The screenshot shows the 'Your Watchlists' section of the FINTRACK app. A watchlist named 'messi' is selected, which is a 'Mutual Funds Watchlist'. The list contains three items: 'M_AXLB - Axis Bluechip Fund', 'M_TADL - Tata Digital India Fund - Growth', and 'M_PARO - Parag Parikh Flexi Cap Fund - Growth'. Each item has a small icon and a trash can icon for deletion.

6.9. Delete an Instrument from a Watchlist

This screenshot is similar to the previous one, showing the 'Your Watchlists' section. The 'messi' watchlist is still selected. However, the third item, 'M_PARO - Parag Parikh Flexi Cap Fund - Growth', is no longer visible, indicating it has been deleted.

Successfully deleted the instrument from watchlist

The screenshot shows the FINTRACK app's header with the logo 'FINTRACK' and the tagline 'Share investment watchlists easily!'. Below the header, a green banner displays the message 'Successfully removed "M_PARO" from watchlist "messi"'. At the bottom, there is a navigation bar with links for 'Welcome, Devendra', 'Home', 'Search', and 'Logout'.

6.10. Share Watchlist

The screenshot shows the FINTRACK application interface. At the top, there is a navigation bar with 'Welcome, Jasvin' and links for 'Home' and 'Search'. On the right, there is a 'Logout' button. The main area has a dark background with a stock market chart overlay. A modal window titled 'Create a New Watchlist' is open, asking for a 'Watchlist Name' (with 'TrialWatchlist' entered) and a choice between 'Mutual Funds Watchlist' and 'Stocks Watchlist'. A 'Submit' button is at the bottom of the modal. Below the modal, the main dashboard shows 'Your Watchlists' with two entries: 'Hojabhai Stocks Watchlist' and 'IT Stocks Stocks Watchlist'. A message indicates that a shareable URL has been copied to the clipboard. The system status bar at the bottom shows the date as 'Sunday 8 May, 11:21 AM'.

Viewing shared watchlist from another account

The screenshot shows the FINTRACK application interface from the perspective of a different user, Devendra. The top navigation bar shows 'Welcome, Devendra' and 'Logout'. The main dashboard displays a 'Your Watchlists' section with one entry: 'Hojabhai Stocks Watchlist'. Below it, under 'Shared Watchlists', are two entries: 'SBIN - State Bank of India' and 'TCS - Tata Consultancy Services'. The system status bar at the bottom shows the date as 'Sunday 8 May, 11:21 AM'.

7. Scope for Future Work

Rename a watchlist

Users can rename their already existing watchlists.

Rearrangement of instruments in watchlist

Users can rearrange the instruments in any of their watchlists as per their priority.

Dynamic presentation of actual price of instrument

Users can view the dynamic rate of change of prices of stock/MFs.

Add end-to-end testing

Test the entire application from frontend to backend via automation tools like Selenium, Cypress, etc.

Deployment on EC2 instance

Deploy the project on Amazon EC2 instance or a proper IaaS deployment basically.

8. Conclusion

We have successfully created the web-app *Fintrack* that will be helpful for users to share their watchlists with anyone they would want to.

We have integrated the entire DevOps CI/CD pipeline with the help of Jenkins.

Used Docker containers for increasing portability and adopting a microservice architecture development.

Used Docker Compose to orchestrate container deployment and manage them.

Used supertest for backend API testing and the React Testing Library for frontend testing.

Used Ansible for deployment and Ansible Vault with Jenkins for secrets management with templating via Jinja2.

We also visualized the logs on the ELK stack in an Elastic Cloud cluster.

And finally we also provided a PaaS deployment with the Heroku + Netlify + Github Actions alternative.

9. References

- [1] <https://fullstackopen.com/>
- [2] <https://github.com/john-smilga/node-express-course>
- [3] <https://reactjs.org/docs/getting-started.html>
- [4] <https://expressjs.com/>
- [5] <https://mongoosejs.com/docs/api.html>
- [6] <https://www.npmjs.com/>
- [7] <https://docs.ansible.com/>
- [8] <https://blog.ktz.me/secret-management-with-docker-compose-and-ansible/>
- [9] <https://www.freecodecamp.org/news/how-to-deploy-react-router-based-app-to-netlify/>
- [10] <https://github.com/marketplace/actions/deploy-to-heroku>