International Institute of Information Technology, Bangalore
(IIIT Bangalore)

Software Production Engineering
Project Report

CryptDash(Dashboard for Cryptocurrency)

Under the Guidance of Prof. B. Thangaraju

Teaching Assistant: Biswadeep

Sheetal Agarwal
MT2022109

Sahil Khatri
MT2022095

TABLE OF CONTENTS

Table of Contents

1. Abstract

CryptDash offers market data, price charts, and analysis of different cryptocurrencies. Users can create their own wishlist and track the prices etc to make informed investment decisions.

Cryptocurrencies have a vast range of options available, and it can be challenging to keep track of all the ones you are interested in. By creating a wishlist, you can compile a list of cryptocurrencies you are interested in or want to monitor closely. This helps you stay organized and easily refer back to the list when researching or making investment decisions.

A wish list allows you to track the price movements of specific cryptocurrencies on the go. By adding cryptocurrencies to your wishlist, you can quickly check their current prices, performance trends and other information quickly.. This can be useful for assessing potential buying or selling opportunities based on your desired price points or market conditions.

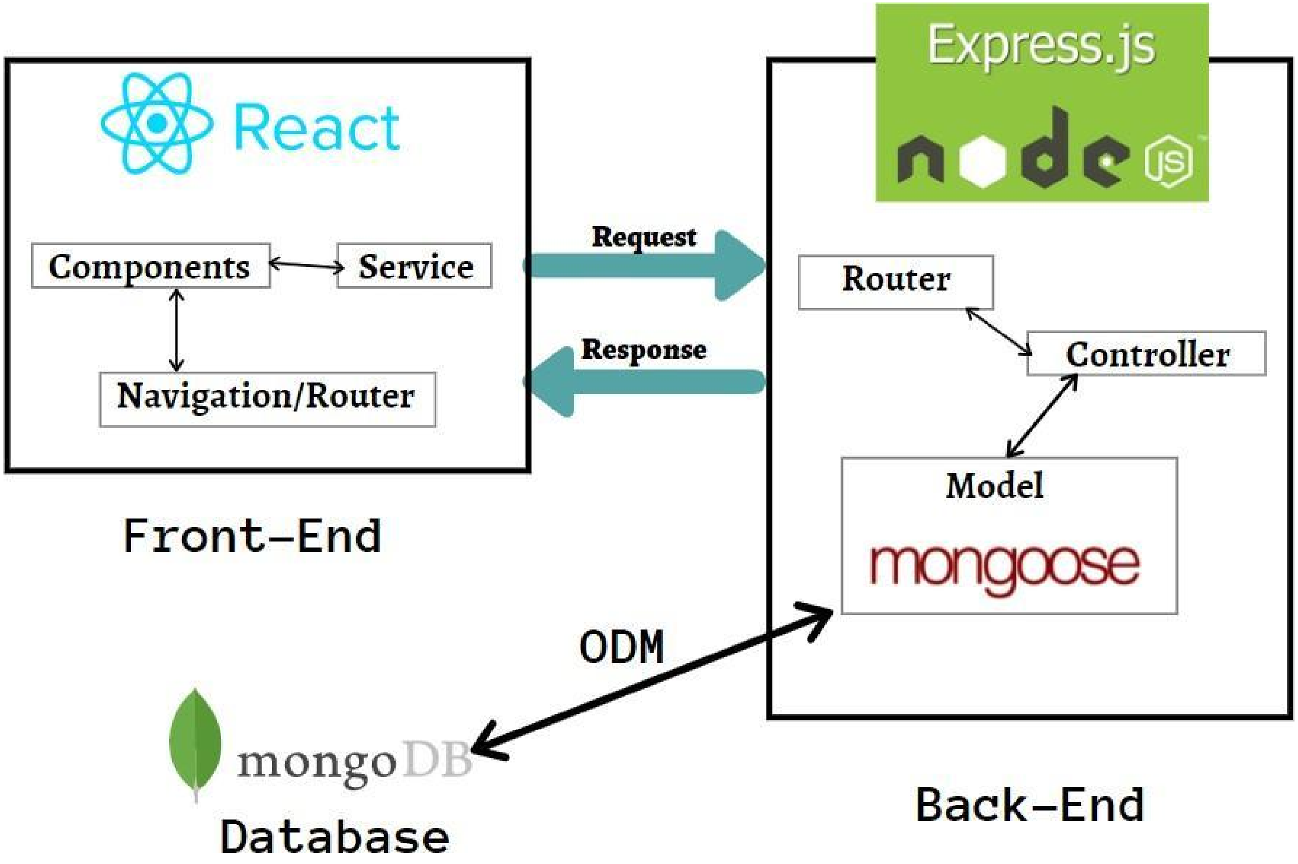
User can also download the details of all the cryptocurrency or that are in its wishlist.

2. Introduction

2.1. Overview

CRYPT DASH is a website that allows users to analyze different cryptocurrencies. Users can create their own wishlist and track the prices etc to make informed investment decisions.

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 cryptocurrencies data.

c. Search Cryptocurrency
Search for cryptocurrency on a search bar and can add them to the watchlist.

d. Create Watchlist
As the user login, for the first time a blank watchlist is created.

e. Download Watchlist
Users can download its watchlist by clicking the download button given in the home page.

f. View Watchlist
Watchlist is displayed in the side bar where users can remove the entries also.

g. Delete Watchlist
Users can delete their watchlists by deleting all the entries in it..

h. View real-time price → On click of specific cryptocurrency on home page take you to another page where details of the cryptocurrency is given.

i. Charts for each cryptocurrency
On Coin detail page charts are given as per timeline(day, week, monthly, yearly)

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, CSS)
Backend: Express (NodeJS, JavaScript)
Database: MongoDB (No-SQL)
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
Logger: Winston
Monitoring: ELK Stack (Elastic Search, Logstash, Kibana)
Secrets Management: Ansible Vault

4. Software Development Life Cycle

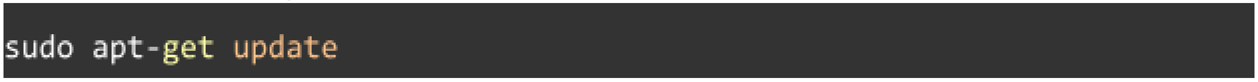
4.1. Installations

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:

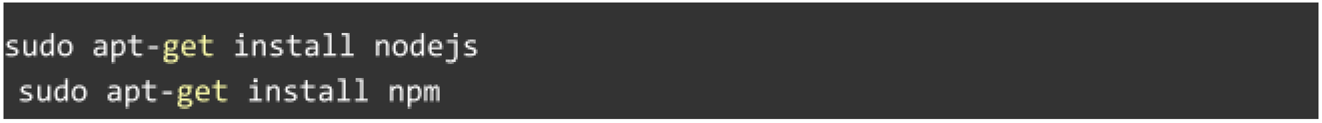
Keep the local packages and softwares updated.



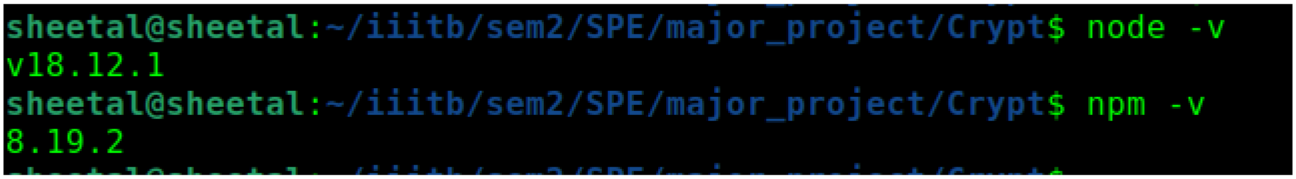
```
sudo apt-get update
```

Install NodeJS and NPM:

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

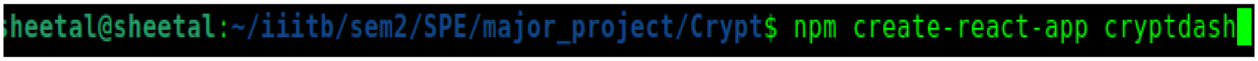


```
sudo apt-get install nodejs
 sudo apt-get install npm
```

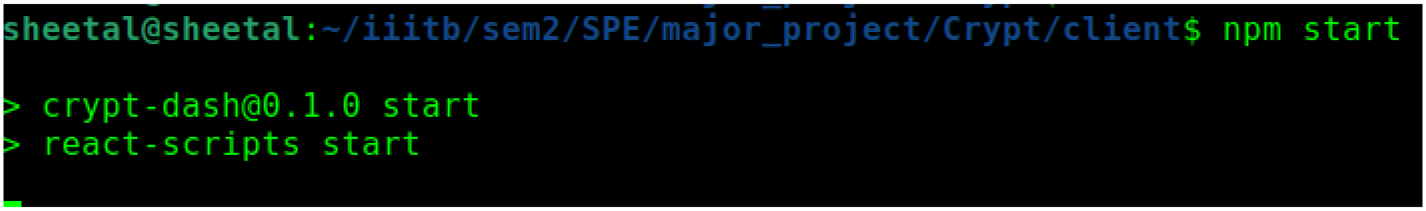


```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt$ node -v
v18.12.1
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt$ npm -v
8.19.2
```

React App



```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt$ npm create-react-app cryptdash
```



```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/client$ npm start

> crypt-dash@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

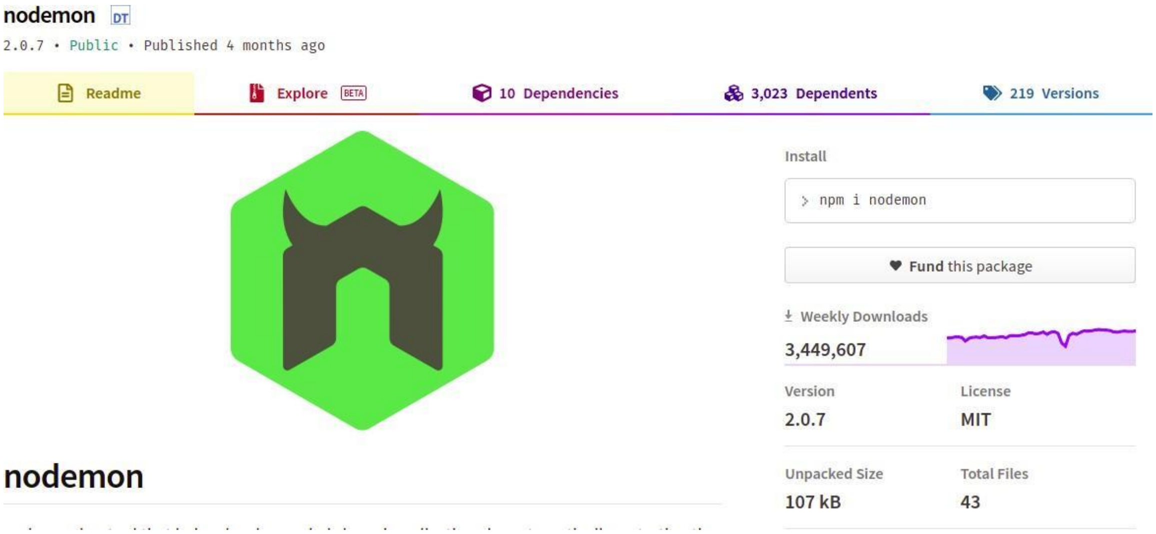
```
npm init -y
```

Initializing node project with the default config:
-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

2.0.7 • Public • Published 4 months ago

Install

```
> npm i nodemon
```

Weekly Downloads

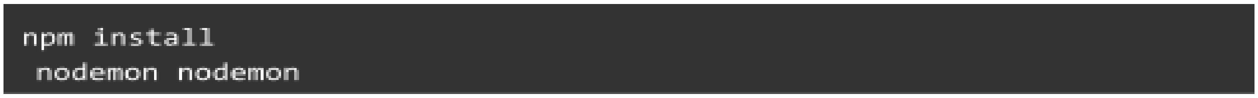
3,449,607

Version	License
2.0.7	MIT

Unpacked Size	Total Files
107 kB	43

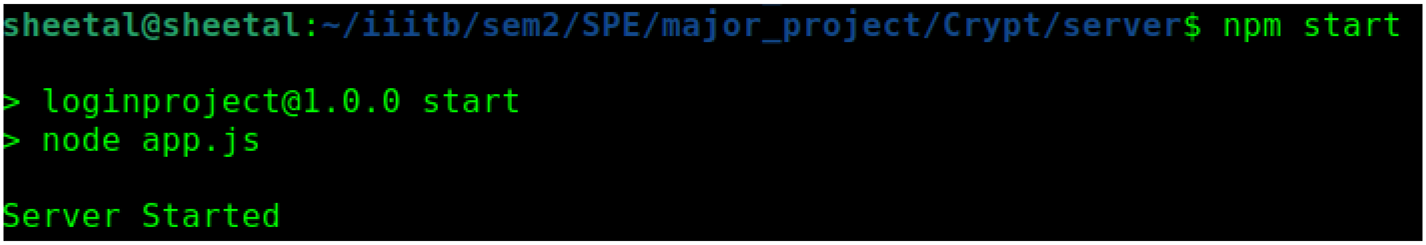
nodemon

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



```
npm install
 nodemon nodemon
```

The server is setup to run on localhost:5000



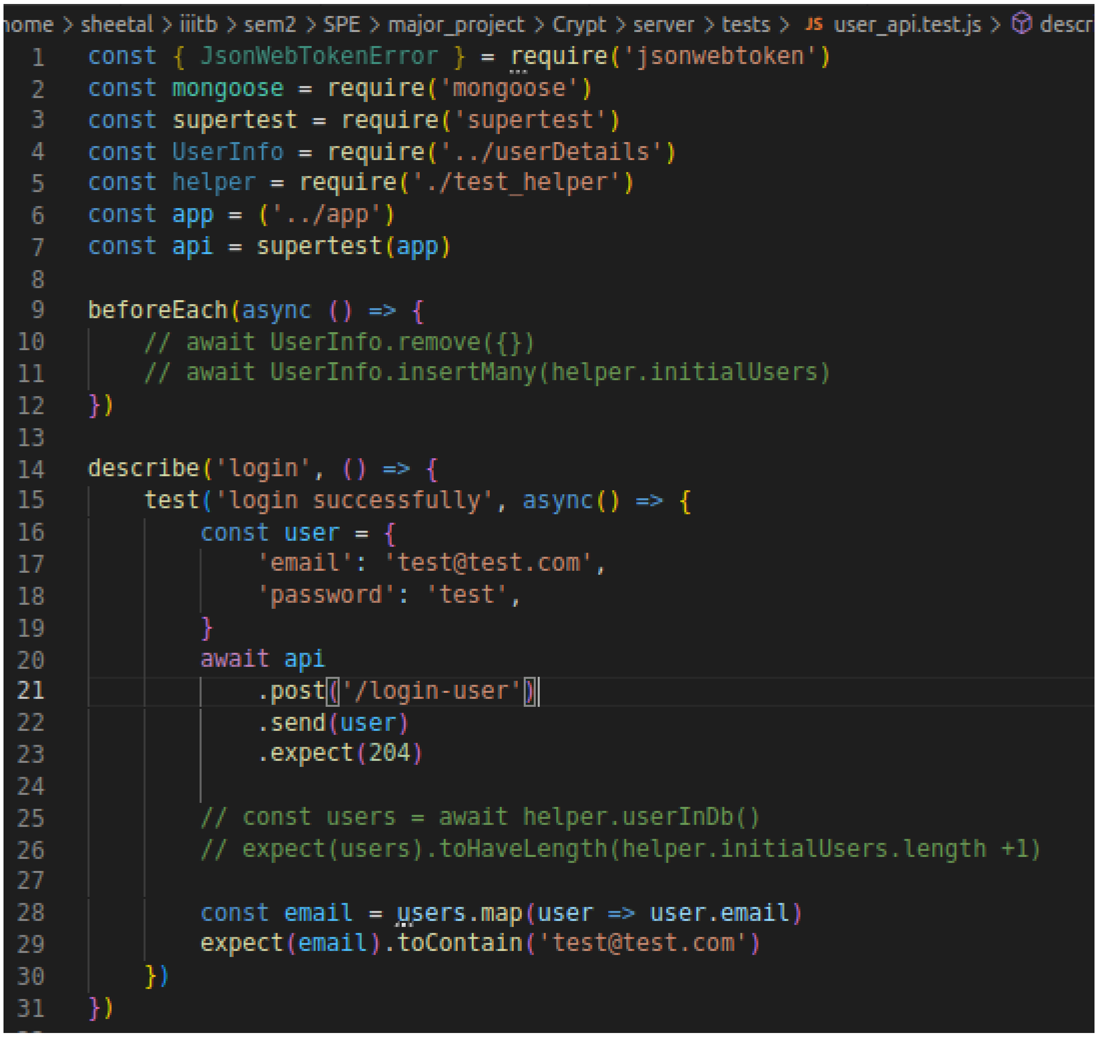
```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ npm start

> loginproject@1.0.0 start
> node app.js

Server Started
```

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.

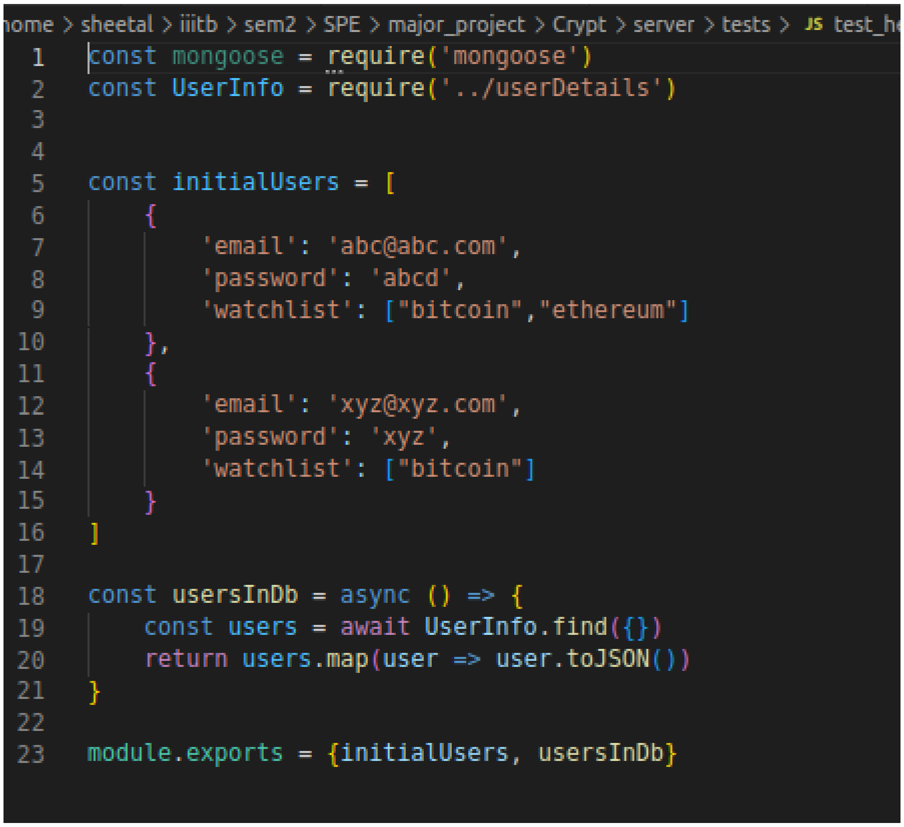
Here is a snippet of how to write tests with Supertest, in this case, we are testing the */users/* API.



ome > sheetal > iiitb > sem2 > SPE > major_project > Crypt > server > tests > JS user_api.test.js > descr

```
const { JsonWebTokenError } = require('jsonwebtoken')
const mongoose = require('mongoose')
const supertest = require('supertest')
const UserInfo = require('../userDetails')
const helper = require('./test_helper')
const app = ('../app')
const api = supertest(app)

beforeEach(async () => {
    // await UserInfo.remove({})
    // await UserInfo.insertMany(helper.initialUsers)
})

describe('login', () => {
    test('login successfully', async() => {
        const user = {
            'email': 'test@test.com',
            'password': 'test',
        }
        await api
            .post('/login-user')
            .send(user)
            .expect(204)

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

        const email = users.map(user => user.email)
        expect(email).toContain('test@test.com')
    })
})
```

testhelper.js


ome > sheetal > iiitb > sem2 > SPE > major_project > Crypt > server > tests > JS test_he

```javascript
const mongoose = require('mongoose')
const UserInfo = require('../userDetails')


const initialUsers = [
    {
        'email': 'abc@abc.com',
        'password': 'abcd',
        'watchlist': ["bitcoin","ethereum"]
    },
    {
        'email': 'xyz@xyz.com',
        'password': 'xyz',
        'watchlist': ["bitcoin"]
    }
]

const usersInDb = async () => {
    const users = await UserInfo.find({})
    return users.map(user => user.toJSON())
}

module.exports = {initialUsers, usersInDb}
```

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*

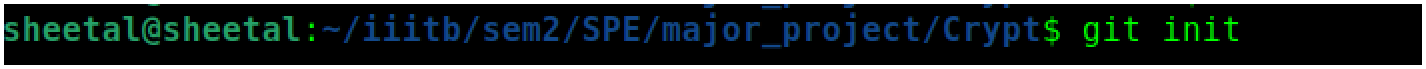
4.3. Source Control Management (SCM)

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.

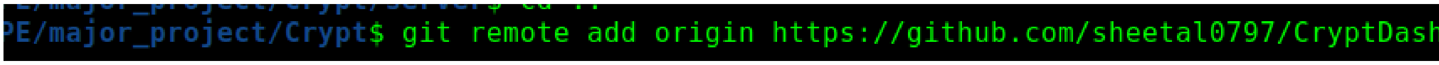
Repository Link: https://github.com/sheetal0797/CryptDash

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

the project with git:

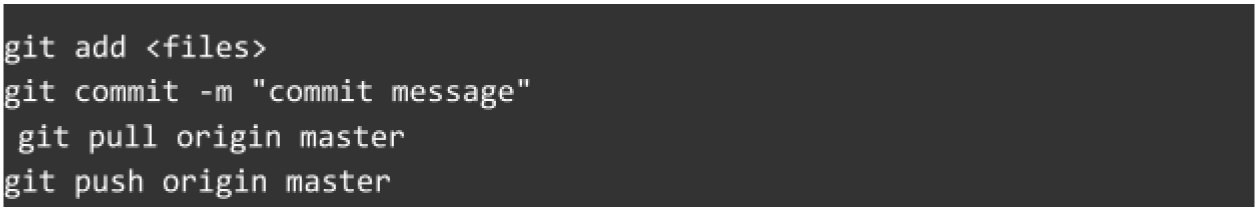


```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt$ git init
```



```
PE/major_project/Crypt$ git remote add origin https://github.com/sheetal0797/CryptDash
```

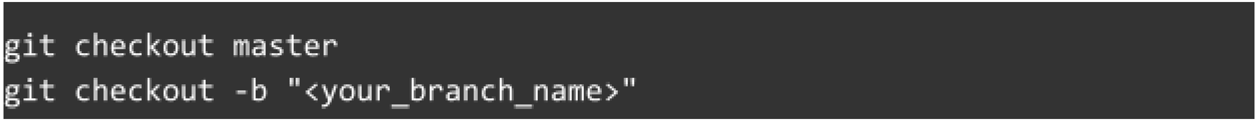
Workflow:



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

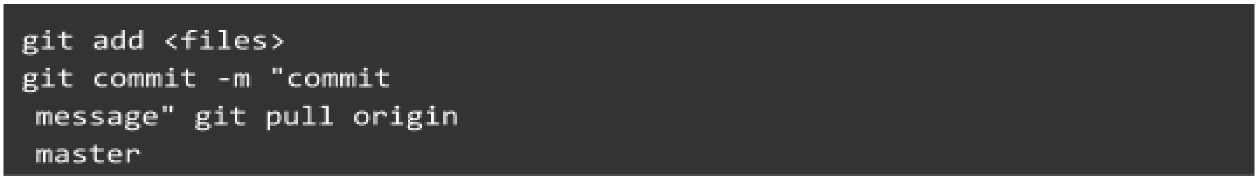
The code is first pulled before making a push to make sure that our project is in the latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “master” branch.

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
```

Then merge the pull request from Github.

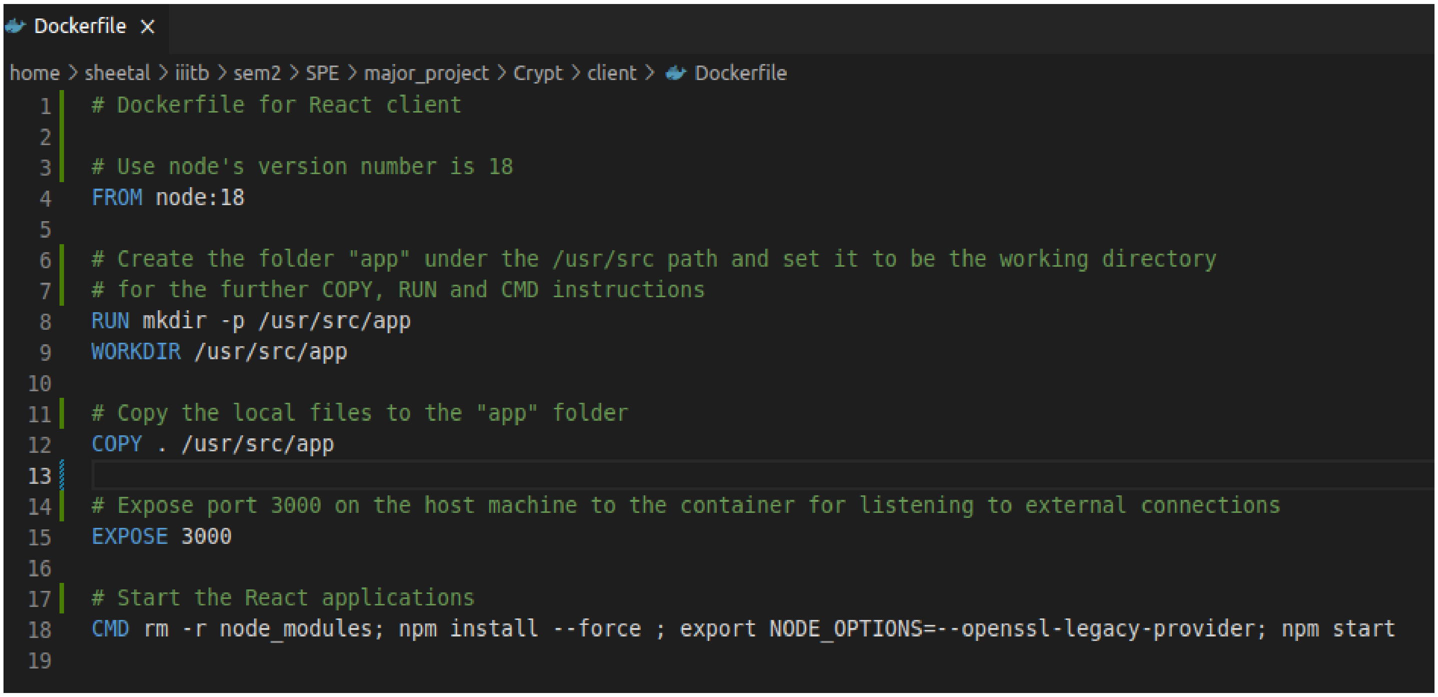
4.4. Containerization with Docker

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.

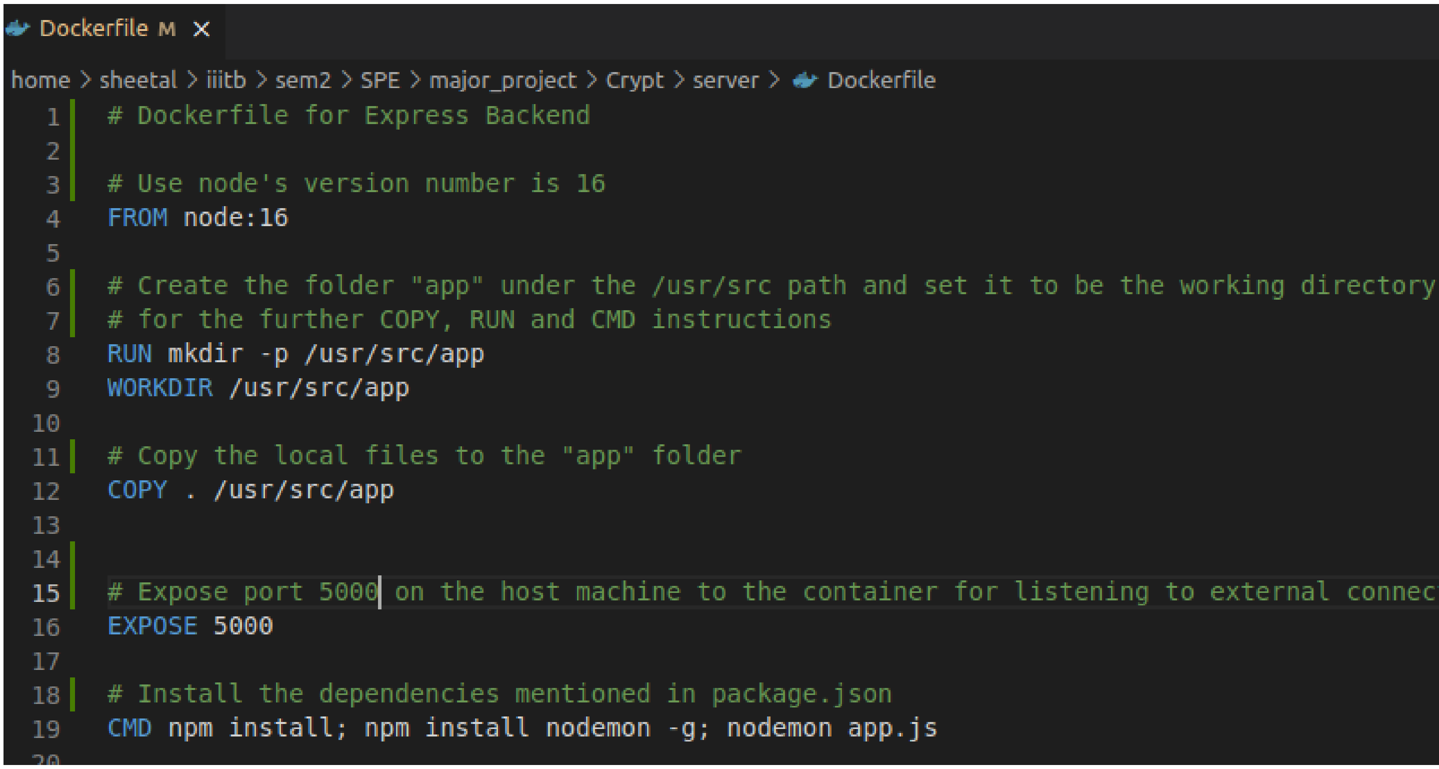
Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.

Frontend Dockerfile


Dockerfile ×
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > client > Dockerfile

```dockerfile
# Dockerfile for React client

# Use node's version number is 18
FROM node:18

# 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
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Copy the local files to the "app" folder
COPY . /usr/src/app

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

# Start the React applications
CMD rm -r node_modules; npm install --force ; export NODE_OPTIONS=--openssl-legacy-provider; npm start
```

Backend Dockerfile


Dockerfile M ×
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > server > Dockerfile

```
# Dockerfile for Express Backend

# Use node's version number is 16
FROM node:16

# 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
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Copy the local files to the "app" folder
COPY . /usr/src/app


# Expose port 5000 on the host machine to the container for listening to external connec
EXPOSE 5000

# Install the dependencies mentioned in package.json
CMD npm install; npm install nodemon -g; nodemon app.js
```

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.

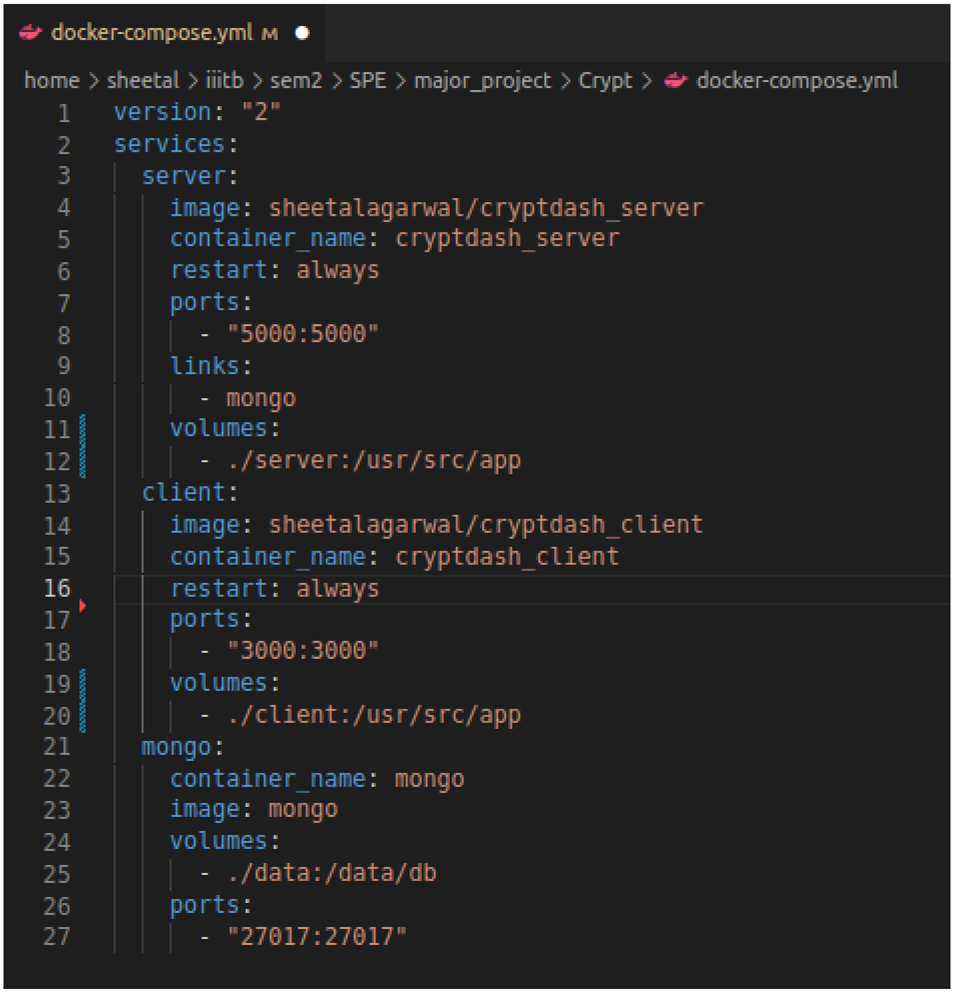
Map port 3000 on the host machine to port 3000 on the frontend container because that’s where React runs and do the same with the backend but with port 5000 because we’ve configured the Express app to listen on port 5000.

Attach both containers to the same network which is named *cryptdash*, this enables

the frontend container to communicate with the backend container and vice-versa. We
fix

the subnet of the *cryptdash* 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.


docker-compose.yml M
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > docker-compose.yml

```yaml
version: "2"
services:
  server:
    image: sheetalagarwal/cryptdash_server
    container_name: cryptdash_server
    restart: always
    ports:
      - "5000:5000"
    links:
      - mongo
    volumes:
      - ./server:/usr/src/app
  client:
    image: sheetalagarwal/cryptdash_client
    container_name: cryptdash_client
    restart: always
    ports:
      - "3000:3000"
    volumes:
      - ./client:/usr/src/app
  mongo:
    container_name: mongo
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"
```

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.

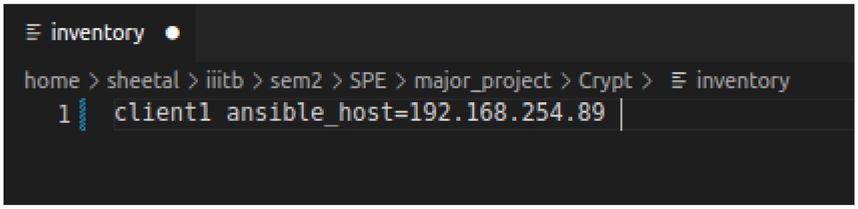
Ansible is mainly used to perform a lot of tasks that otherwise are time-consuming, complex, repetitive, and can make a lot of errors or issues.

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,


inventory ●
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > inventory

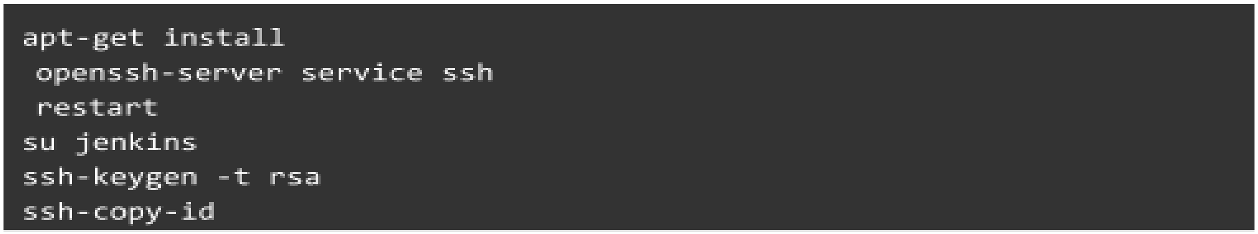
```
client1 ansible_host=192.168.254.89
```

ansible_user 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
```

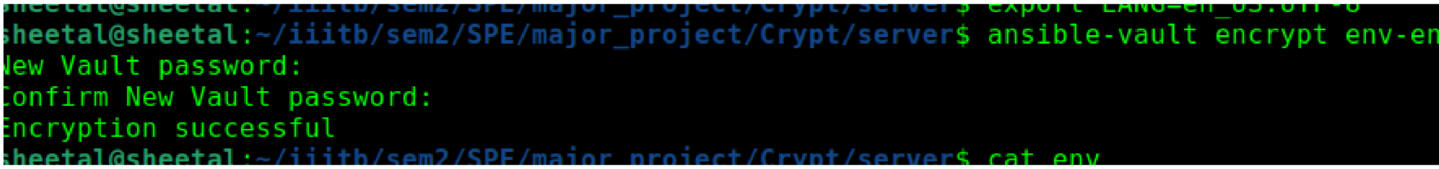
The *chmod* command is required so that the Jenkins user has access to the Docker socket for performing the docker build and push operations.

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

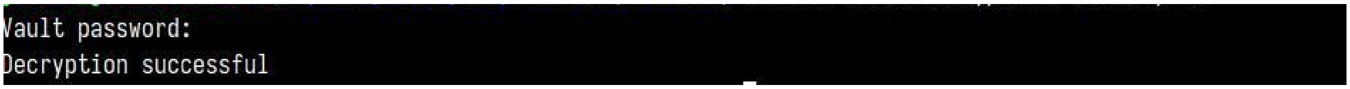
Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

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



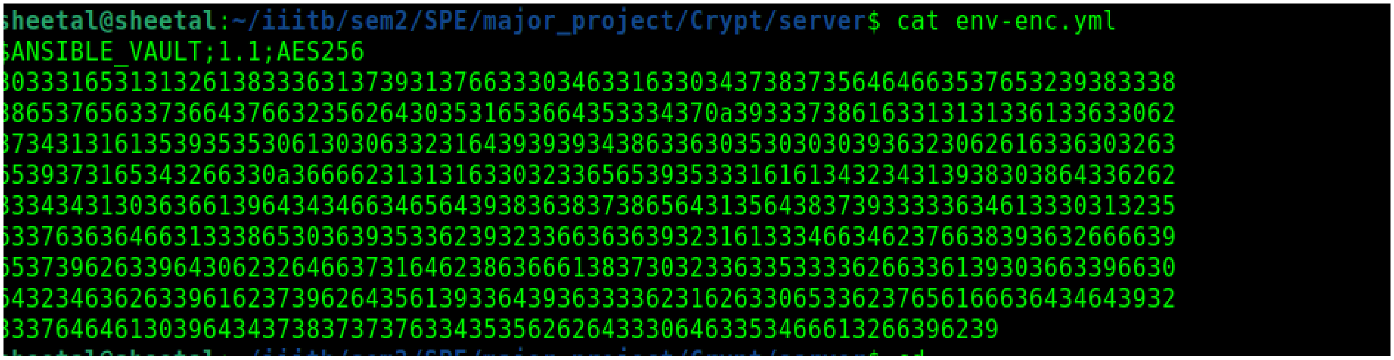
```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ export LANG=en_US.UTF-8
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ ansible-vault encrypt env-en
New Vault password:
Confirm New Vault password:
Encryption successful
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ cat env
```

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



```
Vault password:
Decryption successful
```

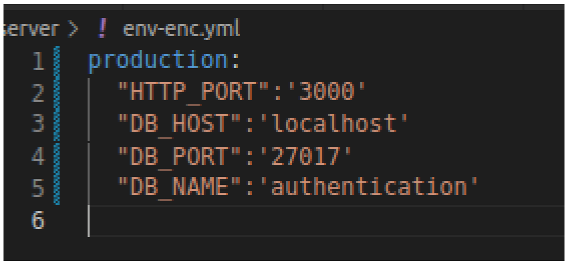
Encrypted YAML file



```
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ cat env-enc.yml
$ANSIBLE_VAULT;1.1;AES256
30333165313132613833363137393137663333034633163330343738373564646635376532393833338
38653765633736643766323562643035316536643533343370a393333738616331313133613363306 2
37343131613539353530613030633231643939393438633630353030303936323062616336303263
65393731653432663330a366662313131633032336565393533316161343234313938303864336262
33343431303636613964343466346564393836383738656431356438373933333634613330313235
63376363646631333865303639353362393233663636393231613334663462376638393632666639
65373962633964306232646637316462386366613837303233633533333626633613930366339663 0
64323463626339616237396264356139336439363333623162633065336237656166636434643932
33376464613039643437383737376334353562626433306463353466613266396239
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ cd
```

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.

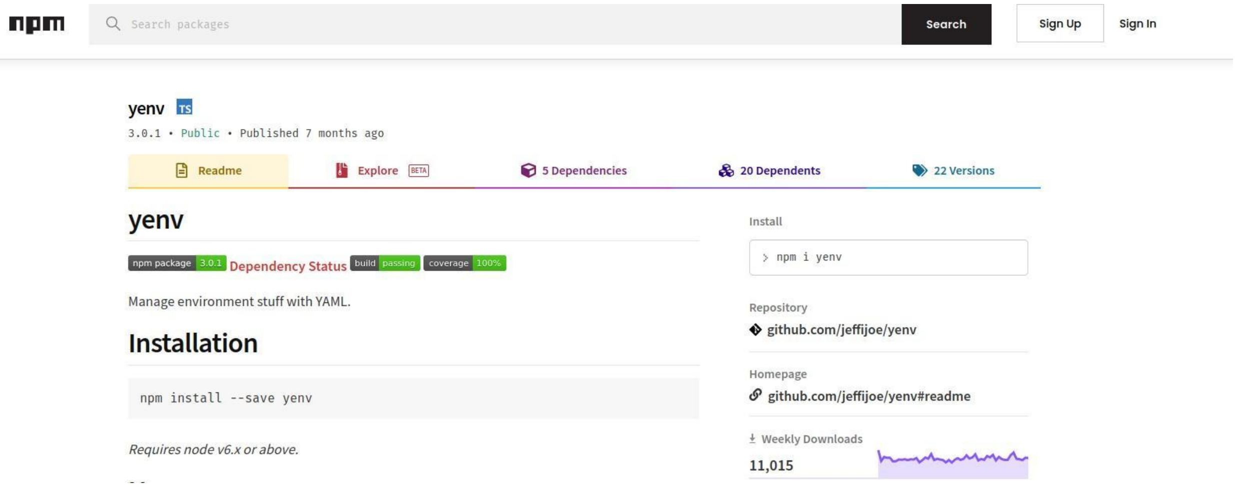
env-enc.yml



erver > ! env-enc.yml

```
production:
  "HTTP_PORT":'3000'
  "DB_HOST":'localhost'
  "DB_PORT":'27017'
  "DB_NAME":'authentication'

```

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 TS

3.0.1 • Public • Published 7 months ago

yenv

npm package 3.0.1 Dependency Status build passing coverage 100%

Manage environment stuff with YAML.

Installation

```
npm install --save yenv
```

Requires node v6.x or above.

Install

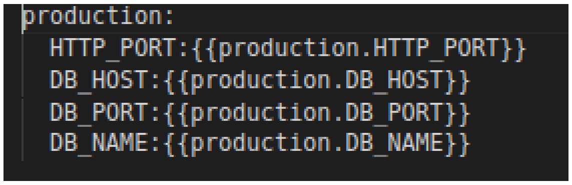
```
> npm i yenv
```

Repository
github.com/jeffijoe/yenv

Homepage
github.com/jeffijoe/yenv#readme

Weekly Downloads
11,015

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



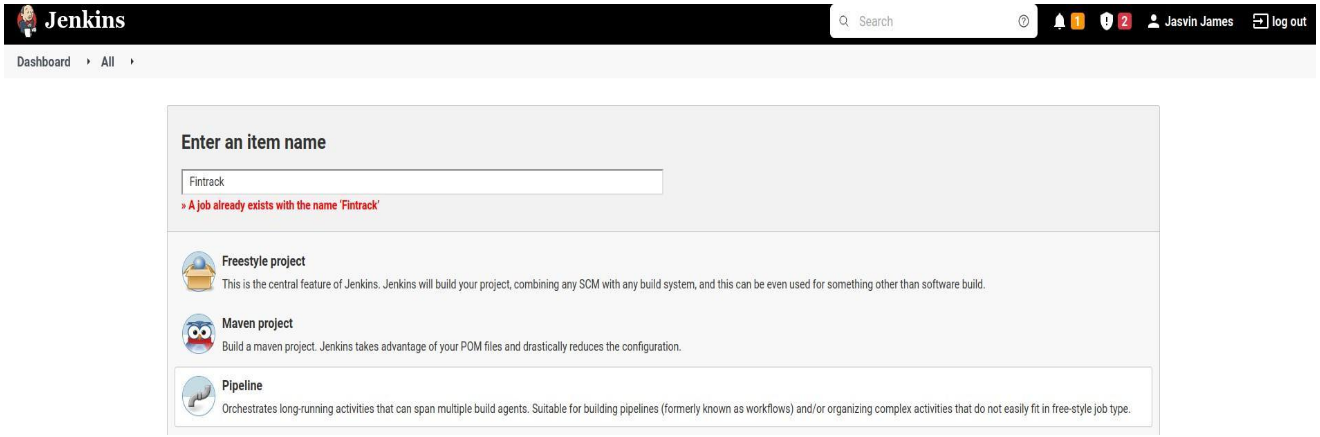
```
production:
  HTTP_PORT:{{production.HTTP_PORT}}
  DB_HOST:{{production.DB_HOST}}
  DB_PORT:{{production.DB_PORT}}
  DB_NAME:{{production.DB_NAME}}
```

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,



Enter an item name

Fintrack

» A job already exists with the name 'Fintrack'

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

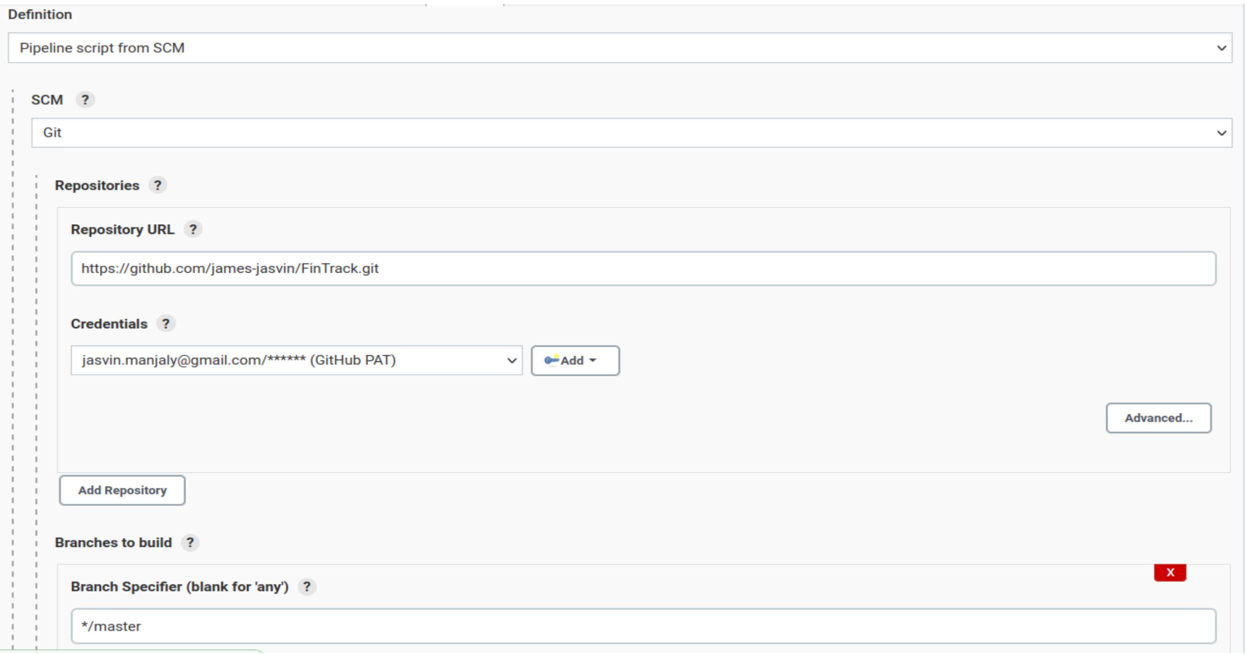
Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline

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



Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/james-jasvin/FinTrack.git

Credentials: jasvin.manjaly@gmail.com/****** (GitHub PAT)

Branches to build

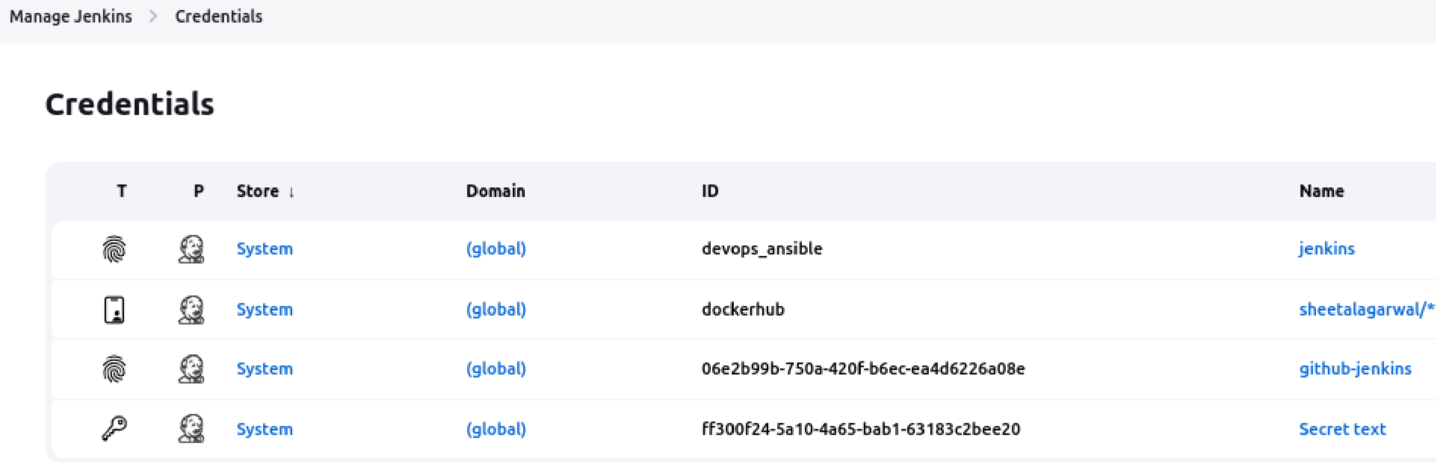
Branch Specifier (blank for 'any'): */master

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.

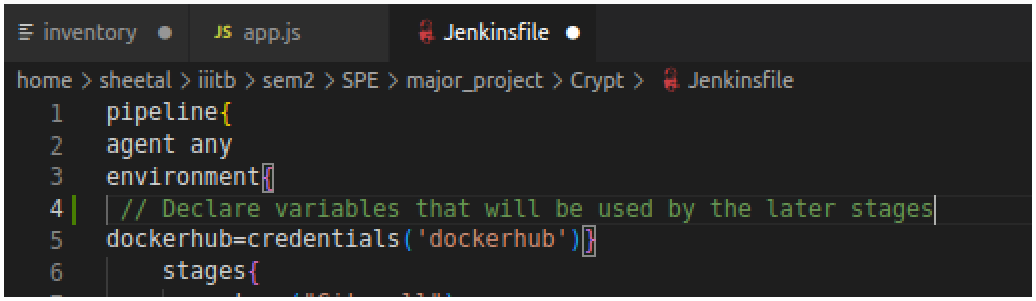


Manage Jenkins > Credentials

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	devops_ansible	jenkins
		System	(global)	dockerhub	sheetalagarwal/*
		System	(global)	06e2b99b-750a-420f-b6ec-ea4d6226a08e	github-jenkins
		System	(global)	ff300f24-5a10-4a65-bab1-63183c2bee20	Secret text

Pipeline Script

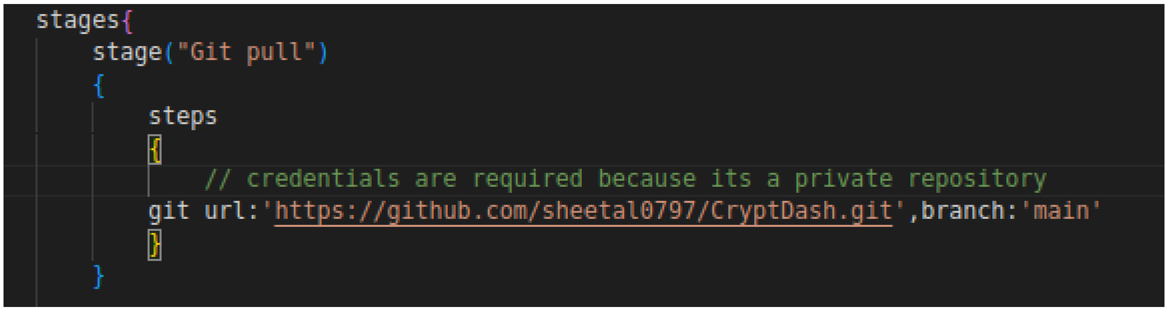


inventory ●
JS app.js
Jenkinsfile ●
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > Jenkinsfile

```
pipeline{
agent any
environment{
 // Declare variables that will be used by the later stages
dockerhub=credentials('dockerhub')}
    stages{
```

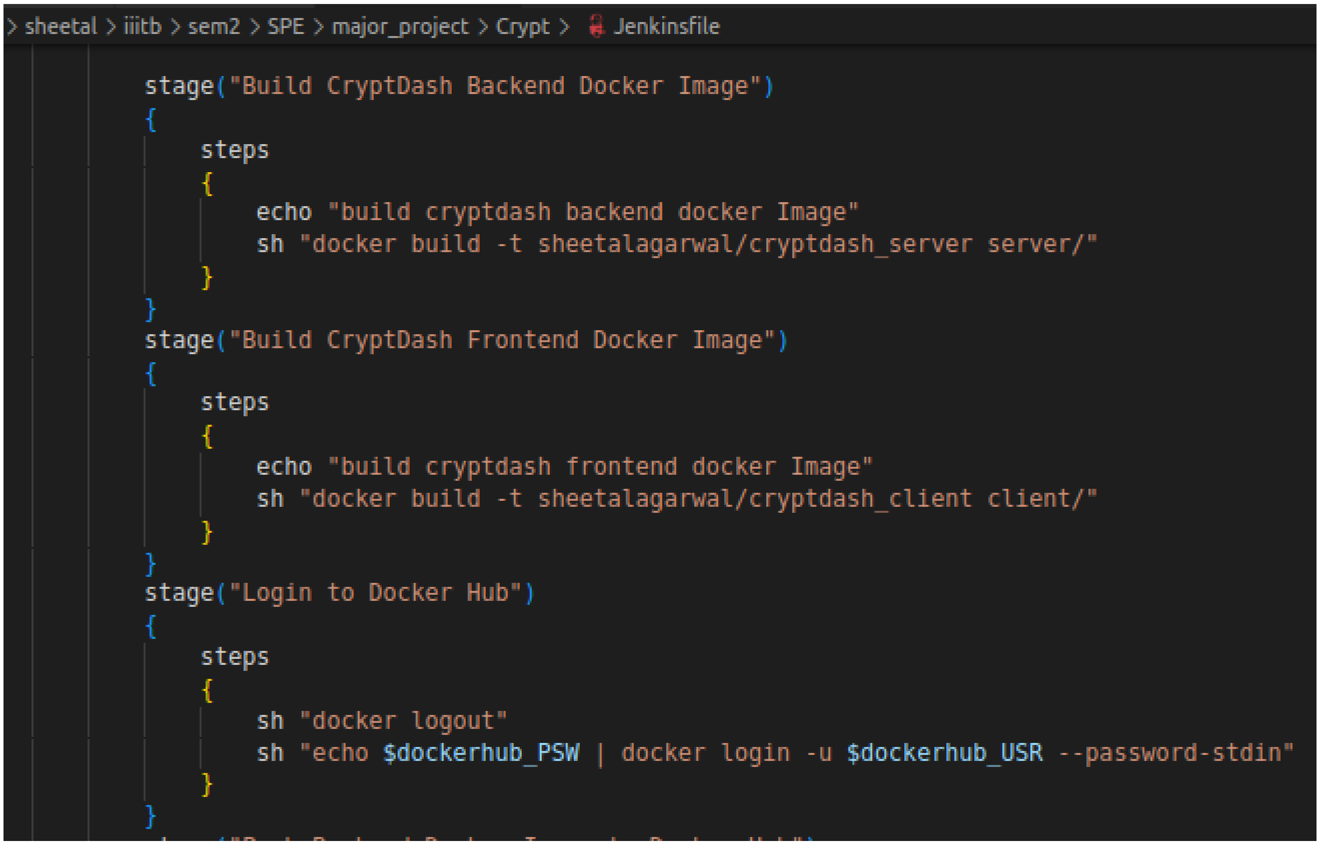
First we set up the environment variables which includes importing the Docker Hub and Ansible Vault passwords as credentials.

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



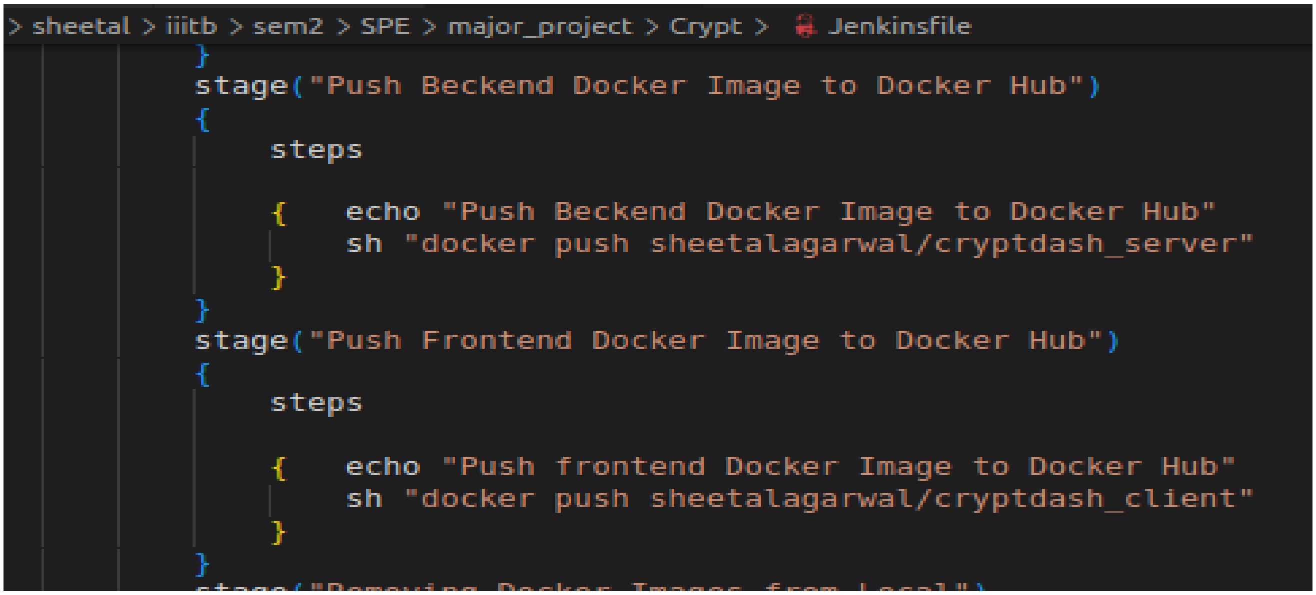
```
stages{
    stage("Git pull")
    {
        steps
        {
            // credentials are required because its a private repository
        git url:'https://github.com/sheetal0797/CryptDash.git',branch:'main'
        }
    }
```

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



> sheetal > iiitb > sem2 > SPE > major_project > Crypt > Jenkinsfile

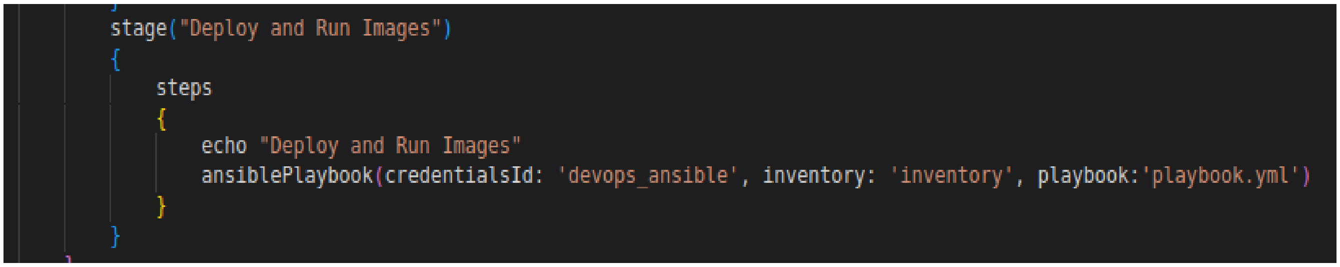
```
        stage("Build CryptDash Backend Docker Image")
        {
            steps
            {
                echo "build cryptdash backend docker Image"
                sh "docker build -t sheetalagarwal/cryptdash_server server/"
            }
        }
        stage("Build CryptDash Frontend Docker Image")
        {
            steps
            {
                echo "build cryptdash frontend docker Image"
                sh "docker build -t sheetalagarwal/cryptdash_client client/"
            }
        }
        stage("Login to Docker Hub")
        {
            steps
            {
                sh "docker logout"
                sh "echo $dockerhub_PSW | docker login -u $dockerhub_USR --password-stdin"
            }
        }
```



> sheetal > iiitb > sem2 > SPE > major_project > Crypt > Jenkinsfile

```
        }
        stage("Push Beckend Docker Image to Docker Hub")
        {
            steps

            {   echo "Push Beckend Docker Image to Docker Hub"
                sh "docker push sheetalagarwal/cryptdash_server"
            }
        }
        stage("Push Frontend Docker Image to Docker Hub")
        {
            steps

            {   echo "Push frontend Docker Image to Docker Hub"
                sh "docker push sheetalagarwal/cryptdash_client"
            }
        }
```

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



```
        }
        stage("Deploy and Run Images")
        {
            steps
            {
                echo "Deploy and Run Images"
                ansiblePlaybook(credentialsId: 'devops_ansible', inventory: 'inventory', playbook:'playbook.yml')
            }
        }
    }
```

4.8. Ansible Playbook

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

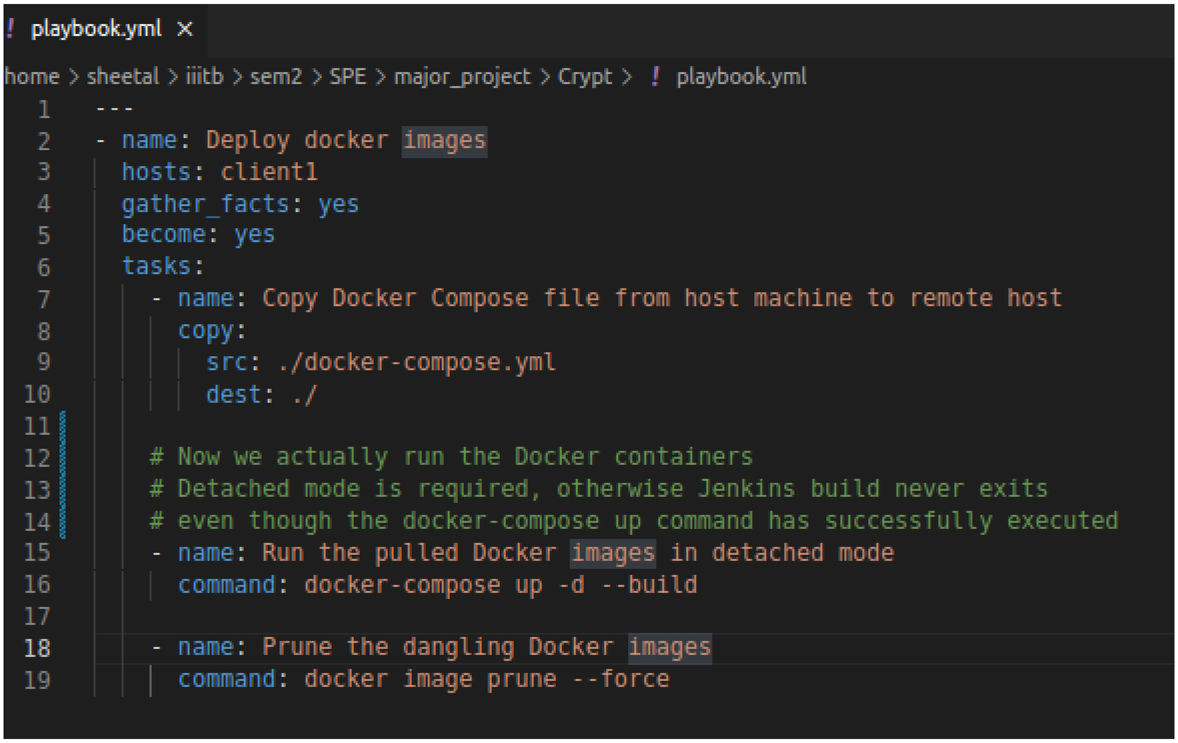
In this playbook, the toughest task is to use the encrypted *env-enc.yaml* file and decrypt it somehow to send to the managed nodes and somehow place it inside the running container.

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.

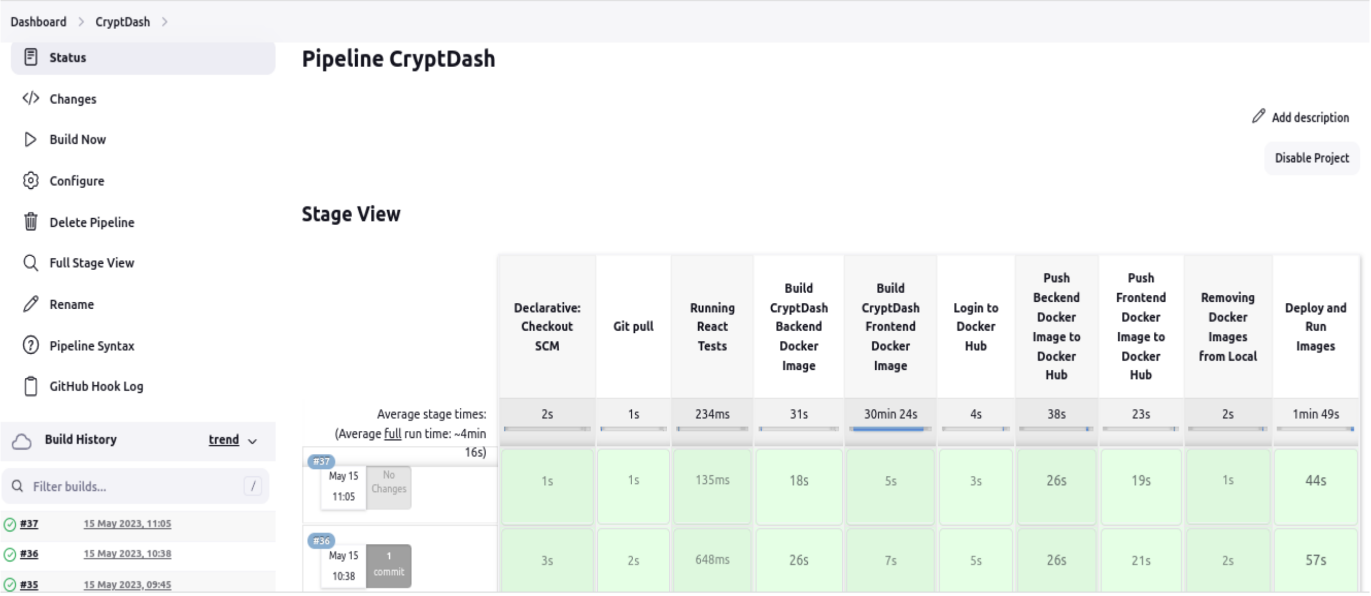
Now all we have to do is copy this *env.yaml* file into the backend container.

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


playbook.yml ×
home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > ! playbook.yml

```yaml
---
- name: Deploy docker images
  hosts: client1
  gather_facts: yes
  become: yes
  tasks:
    - name: Copy Docker Compose file from host machine to remote host
      copy:
        src: ./docker-compose.yml
        dest: ./

    # Now we actually run the Docker containers
    # 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 --build

    - name: Prune the dangling Docker images
      command: docker image prune --force
```

Running Jenkins build,



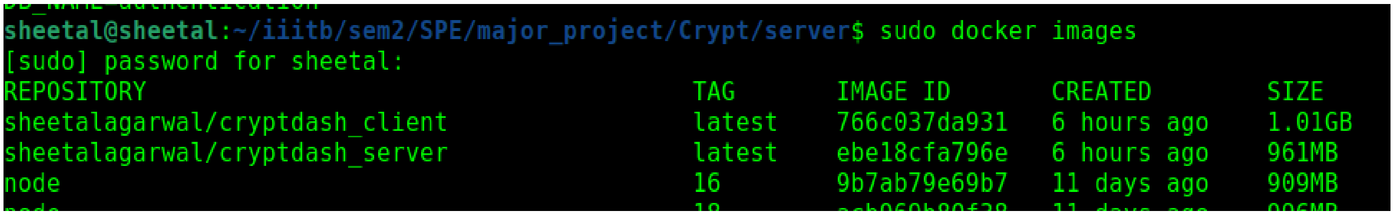
Pipeline CryptDash

Stage View

	Declarative: Checkout SCM	Git pull	Running React Tests	Build CryptDash Backend Docker Image	Build CryptDash Frontend Docker Image	Login to Docker Hub	Push Beckend Docker Image to Docker Hub	Push Frontend Docker Image to Docker Hub	Removing Docker Images from Local	Deploy and Run Images
Average stage times: (Average full run time: ~4min 16s)	2s	1s	234ms	31s	30min 24s	4s	38s	23s	2s	1min 49s
#37 May 15 11:05 No Changes	1s	1s	135ms	18s	5s	3s	26s	19s	1s	44s
#36 May 15 10:38 1 commit	3s	2s	648ms	26s	7s	5s	26s	21s	2s	57s

Build	Date
#37	15 May 2023, 11:05
#36	15 May 2023, 10:38
#35	15 May 2023, 09:45

Running Docker containers after build,



```
DB_NAME authentication
sheetal@sheetal:~/iiitb/sem2/SPE/major_project/Crypt/server$ sudo docker images
[sudo] password for sheetal:
REPOSITORY                           TAG       IMAGE ID       CREATED         SIZE
sheetalagarwal/cryptdash_client      latest    766c037da931   6 hours ago     1.01GB
sheetalagarwal/cryptdash_server      latest    ebe18cfa796e   6 hours ago     961MB
node                                 16        9b7ab79e69b7   11 days ago     909MB
node                                 18        acb060b80f38   11 days ago     906MB
```

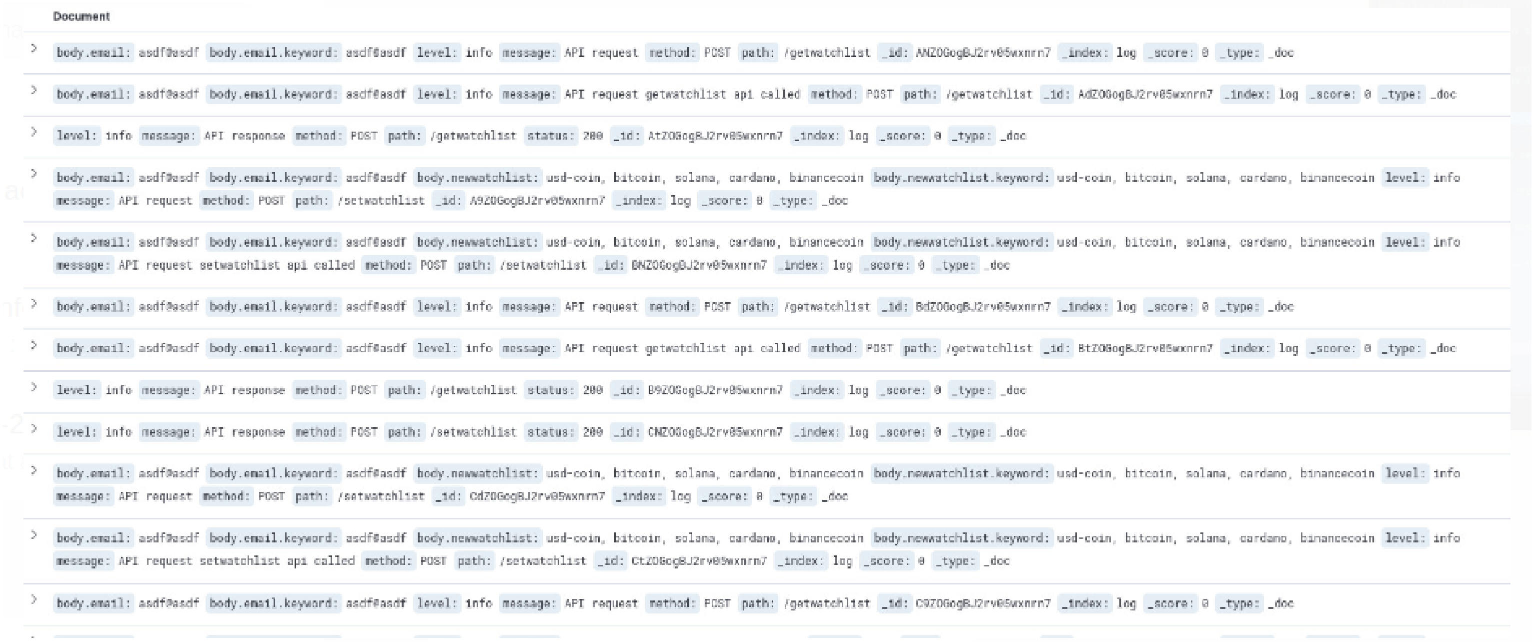
4.9. Monitoring - ELK Stack

"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.*

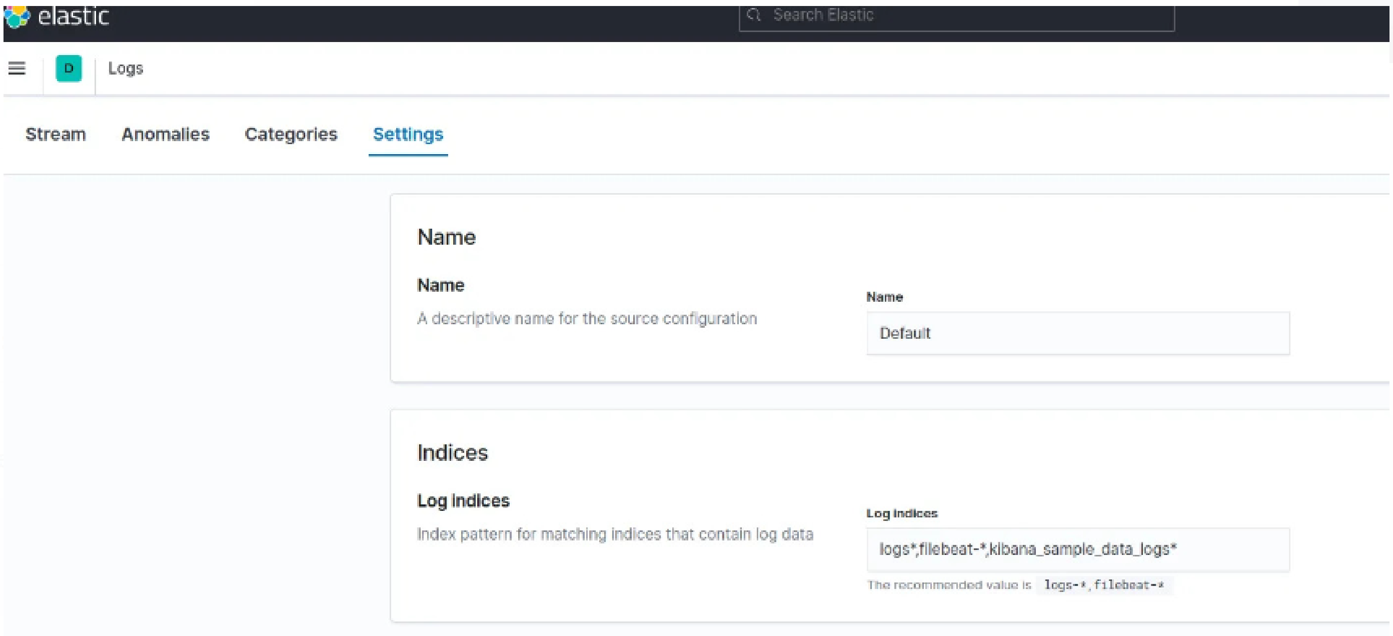
Now we can upload this log file into our Elastic Cloud cluster.

Logs:



Document

```
> body.email: asdf@asdf body.email.keyword: asdf@asdf level: info message: API request method: POST path: /getwatchlist _id: ANZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf level: info message: API request getwatchlist api called method: POST path: /getwatchlist _id: AdZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> level: info message: API response method: POST path: /getwatchlist status: 200 _id: AtZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf body.newwatchlist: usd-coin, bitcoin, solana, cardano, binancecoin body.newwatchlist.keyword: usd-coin, bitcoin, solana, cardano, binancecoin level: info
  message: API request method: POST path: /setwatchlist _id: A9ZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf body.newwatchlist: usd-coin, bitcoin, solana, cardano, binancecoin body.newwatchlist.keyword: usd-coin, bitcoin, solana, cardano, binancecoin level: info
  message: API request setwatchlist api called method: POST path: /setwatchlist _id: BNZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf level: info message: API request method: POST path: /getwatchlist _id: BdZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf level: info message: API request getwatchlist api called method: POST path: /getwatchlist _id: BtZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> level: info message: API response method: POST path: /getwatchlist status: 200 _id: B9ZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> level: info message: API response method: POST path: /setwatchlist status: 200 _id: CNZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf body.newwatchlist: usd-coin, bitcoin, solana, cardano, binancecoin body.newwatchlist.keyword: usd-coin, bitcoin, solana, cardano, binancecoin level: info
  message: API request method: POST path: /setwatchlist _id: CdZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf body.newwatchlist: usd-coin, bitcoin, solana, cardano, binancecoin body.newwatchlist.keyword: usd-coin, bitcoin, solana, cardano, binancecoin level: info
  message: API request setwatchlist api called method: POST path: /setwatchlist _id: CtZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
> body.email: asdf@asdf body.email.keyword: asdf@asdf level: info message: API request method: POST path: /getwatchlist _id: C9ZOGogBJ2rv05wxnrn7 _index: log _score: 0 _type: _doc
```



Name

Name
A descriptive name for the source configuration

Name: Default

Indices

Log indices
Index pattern for matching indices that contain log data

Log indices: logs*,filebeat-*,kibana_sample_data_logs*

The recommended value is `logs-*,filebeat-*`

1. Experimental Setup

1.1. Functional Requirements:

1. Users can register and create their account.
2. Users can create watchlists without creating duplicate entries.
3. Users can view their watchlist.
4. Users can edit the desired watchlist (delete the instruments or add the instruments) as per their wish.
5. Watchlist will have a download button (excel file is downloaded).
6. To view a watchlist, a user must be logged in to a crypt dash account.

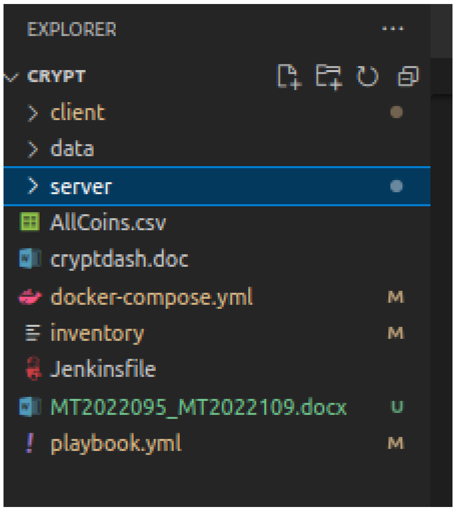
1.2. Non-Functional Requirements:

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 user

1.3. 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.

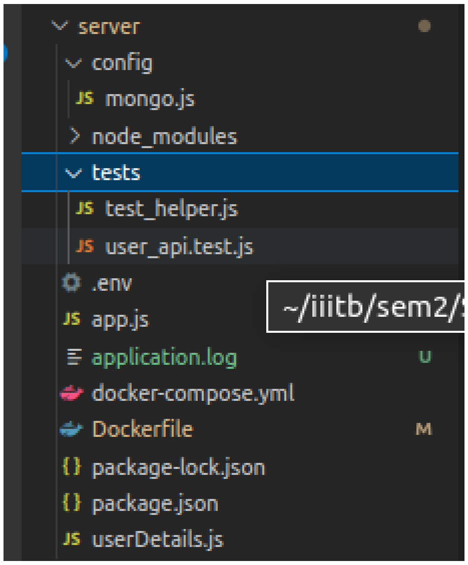
Directory structure for the project



```
EXPLORER                          ...
CRYPT
  > client                         ●
  > data
  > server                         ●
  AllCoins.csv
  cryptdash.doc
  docker-compose.yml               M
  inventory                        M
  Jenkinsfile
  MT2022095_MT2022109.docx         U
  playbook.yml                     M
```

5.4.1. Backend

Directory structure for the Backend



```
server
  config
    mongo.js
  node_modules
  tests
    test_helper.js
    user_api.test.js
  .env
  app.js                 ~/iiitb/sem2/
  application.log        U
  docker-compose.yml
  Dockerfile             M
  package-lock.json
  package.json
  userDetails.js
```

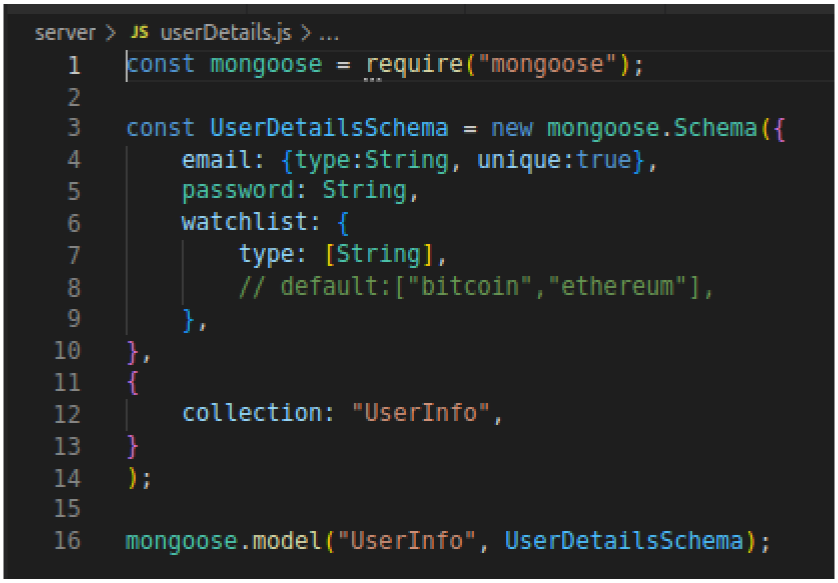
We have incorporated MVC architecture (Model View Controller) in the project.

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



```javascript
const mongoose = require("mongoose");

const UserDetailsSchema = new mongoose.Schema({
    email: {type:String, unique:true},
    password: String,
    watchlist: {
        type: [String],
        // default:["bitcoin","ethereum"],
    },
},
{
    collection: "UserInfo",
}
);

mongoose.model("UserInfo", UserDetailsSchema);
```

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:

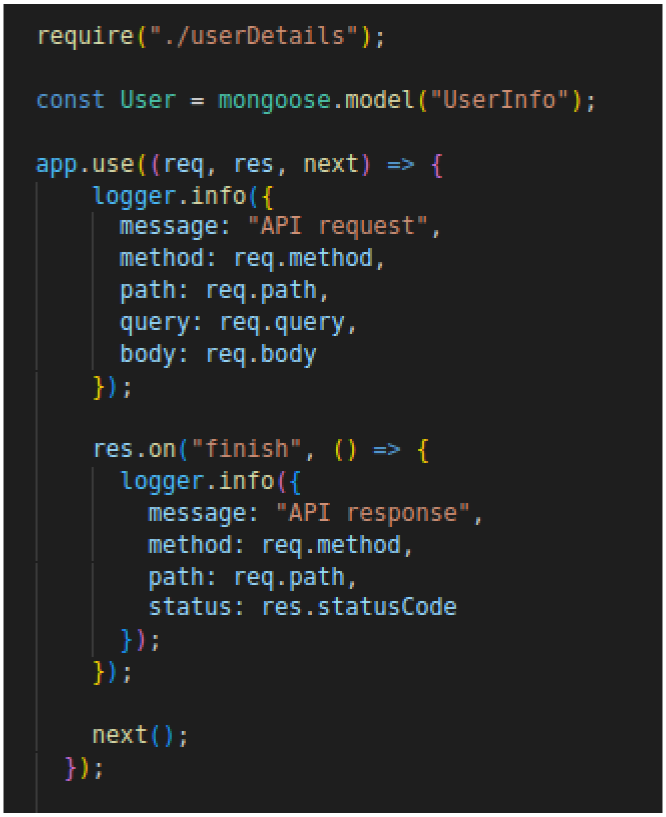
- It's known as the main man because the controller is the component that enables the interconnection between the View and the Model, so it acts as an intermediary.
- The Controller doesn't have to worry about handling data logic, it just tells the Model what to do.
- After receiving data from the Model,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

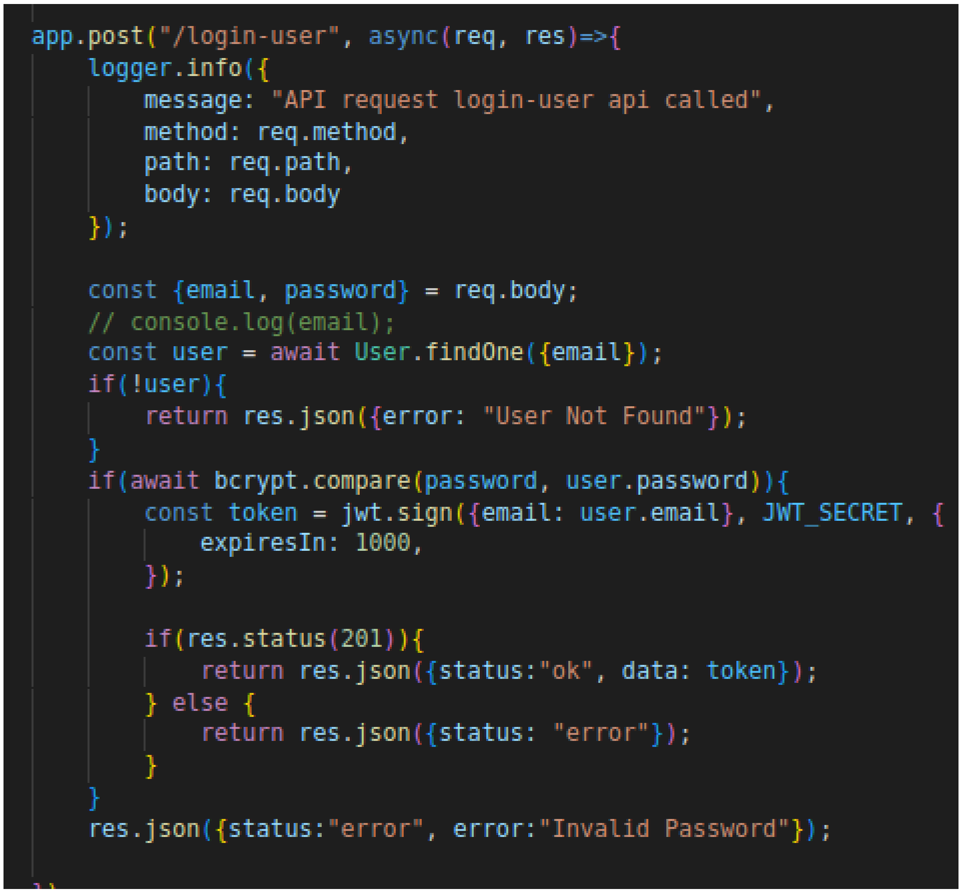
1. UserDetails Controller



```
require("./userDetails");

const User = mongoose.model("UserInfo");

app.use((req, res, next) => {
    logger.info({
      message: "API request",
      method: req.method,
      path: req.path,
      query: req.query,
      body: req.body
    });

    res.on("finish", () => {
      logger.info({
        message: "API response",
        method: req.method,
        path: req.path,
        status: res.statusCode
      });
    });

    next();
  });
```

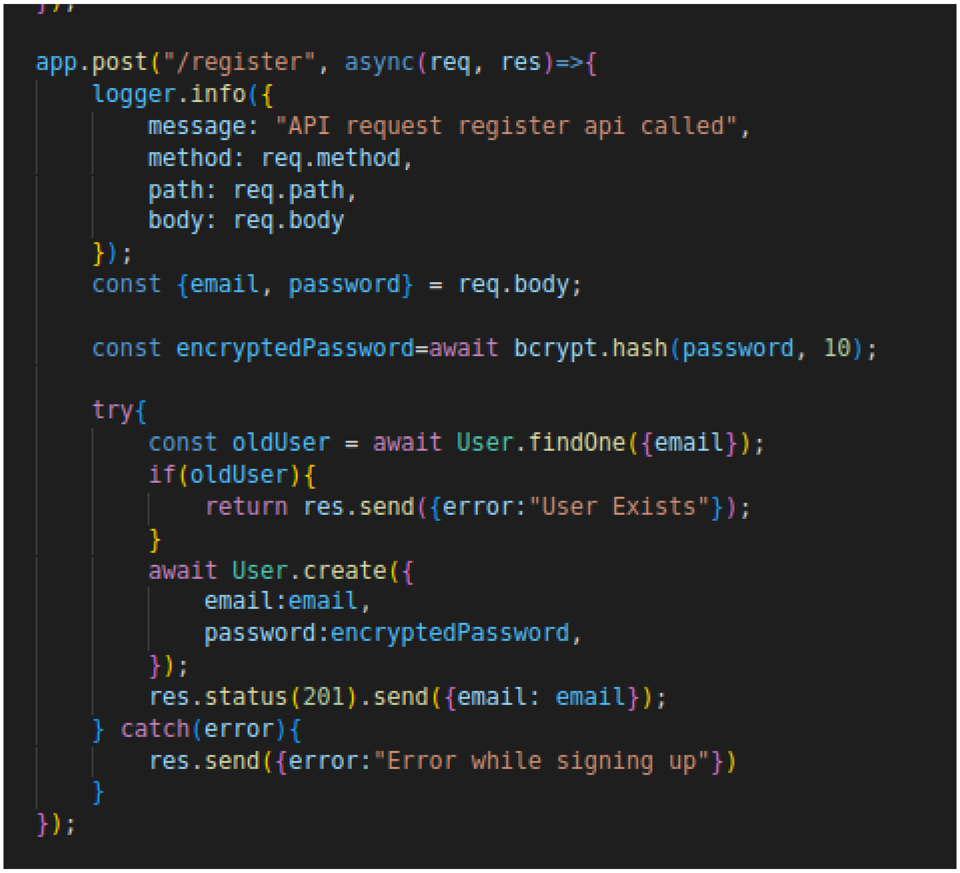
2. Login Controller

Handles the login of users into our web-app.



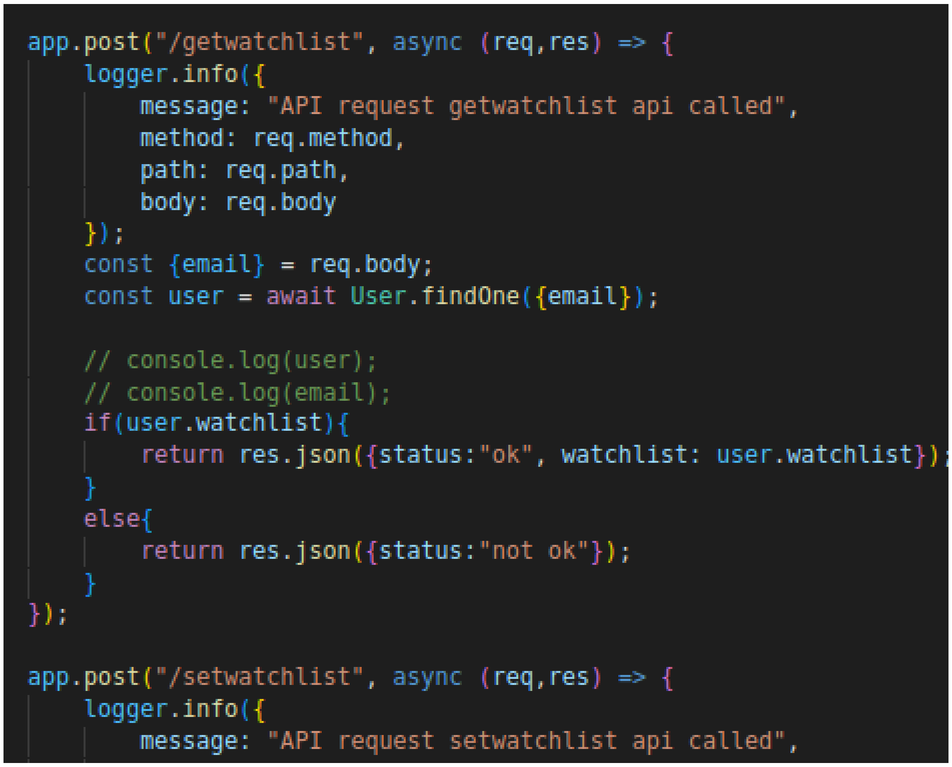
```
app.post("/login-user", async(req, res)=>{
    logger.info({
        message: "API request login-user api called",
        method: req.method,
        path: req.path,
        body: req.body
    });

    const {email, password} = req.body;
    // console.log(email);
    const user = await User.findOne({email});
    if(!user){
        return res.json({error: "User Not Found"});
    }
    if(await bcrypt.compare(password, user.password)){
        const token = jwt.sign({email: user.email}, JWT_SECRET, {
            expiresIn: 1000,
        });

        if(res.status(201)){
            return res.json({status:"ok", data: token});
        } else {
            return res.json({status: "error"});
        }
    }
    res.json({status:"error", error:"Invalid Password"});

})
```

3. Sign Up Controller



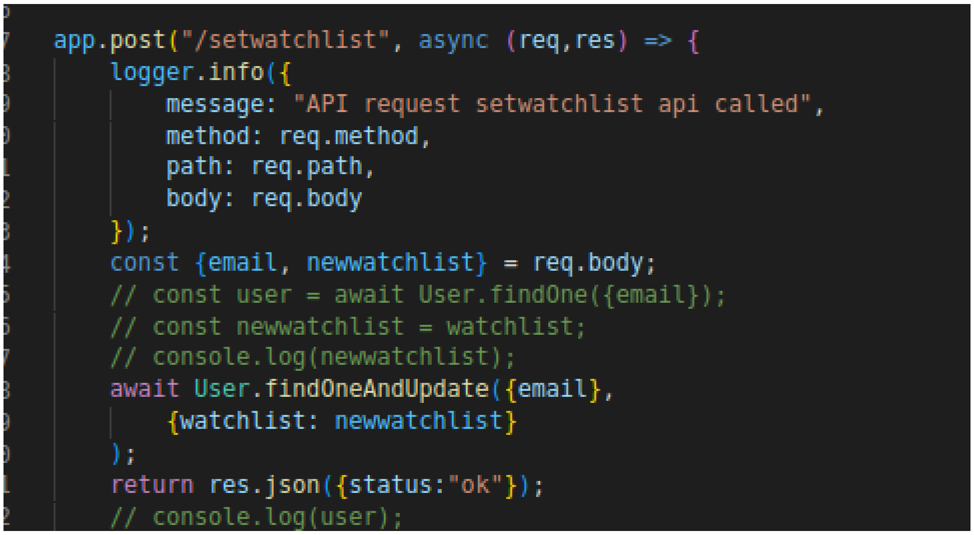
```
});

app.post("/register", async(req, res)=>{
    logger.info({
        message: "API request register api called",
        method: req.method,
        path: req.path,
        body: req.body
    });
    const {email, password} = req.body;

    const encryptedPassword=await bcrypt.hash(password, 10);

    try{
        const oldUser = await User.findOne({email});
        if(oldUser){
            return res.send({error:"User Exists"});
        }
        await User.create({
            email:email,
            password:encryptedPassword,
        });
        res.status(201).send({email: email});
    } catch(error){
        res.send({error:"Error while signing up"})
    }
});
```

4. get watchlist Controller



```
app.post("/getwatchlist", async (req,res) => {
    logger.info({
        message: "API request getwatchlist api called",
        method: req.method,
        path: req.path,
        body: req.body
    });
    const {email} = req.body;
    const user = await User.findOne({email});

    // console.log(user);
    // console.log(email);
    if(user.watchlist){
        return res.json({status:"ok", watchlist: user.watchlist});
    }
    else{
        return res.json({status:"not ok"});
    }
});

app.post("/setwatchlist", async (req,res) => {
    logger.info({
        message: "API request setwatchlist api called",
```

5. setwatchlist



```
app.post("/setwatchlist", async (req,res) => {
    logger.info({
        message: "API request setwatchlist api called",
        method: req.method,
        path: req.path,
        body: req.body
    });
    const {email, newwatchlist} = req.body;
    // const user = await User.findOne({email});
    // const newwatchlist = watchlist;
    // console.log(newwatchlist);
    await User.findOneAndUpdate({email},
        {watchlist: newwatchlist}
    );
    return res.json({status:"ok"});
    // console.log(user);
```

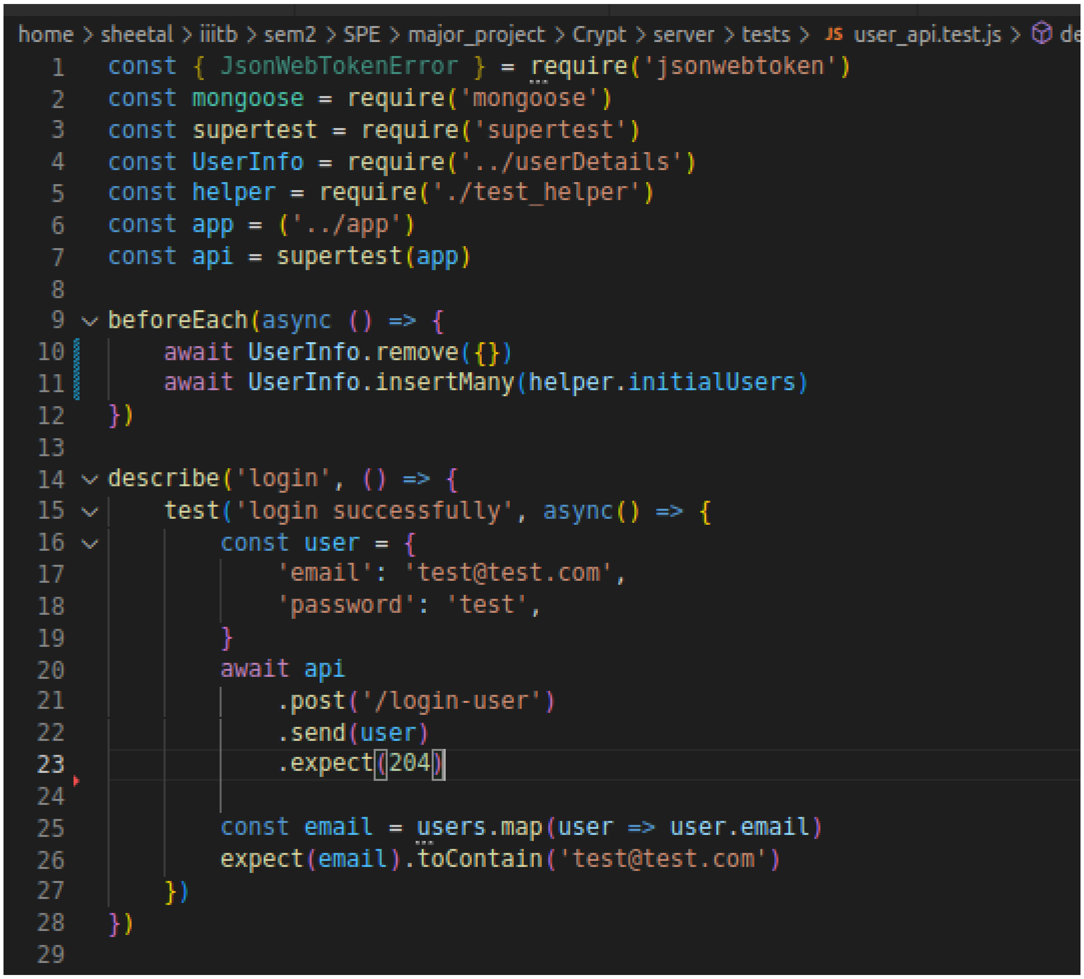
utils directory

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

tests directory

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

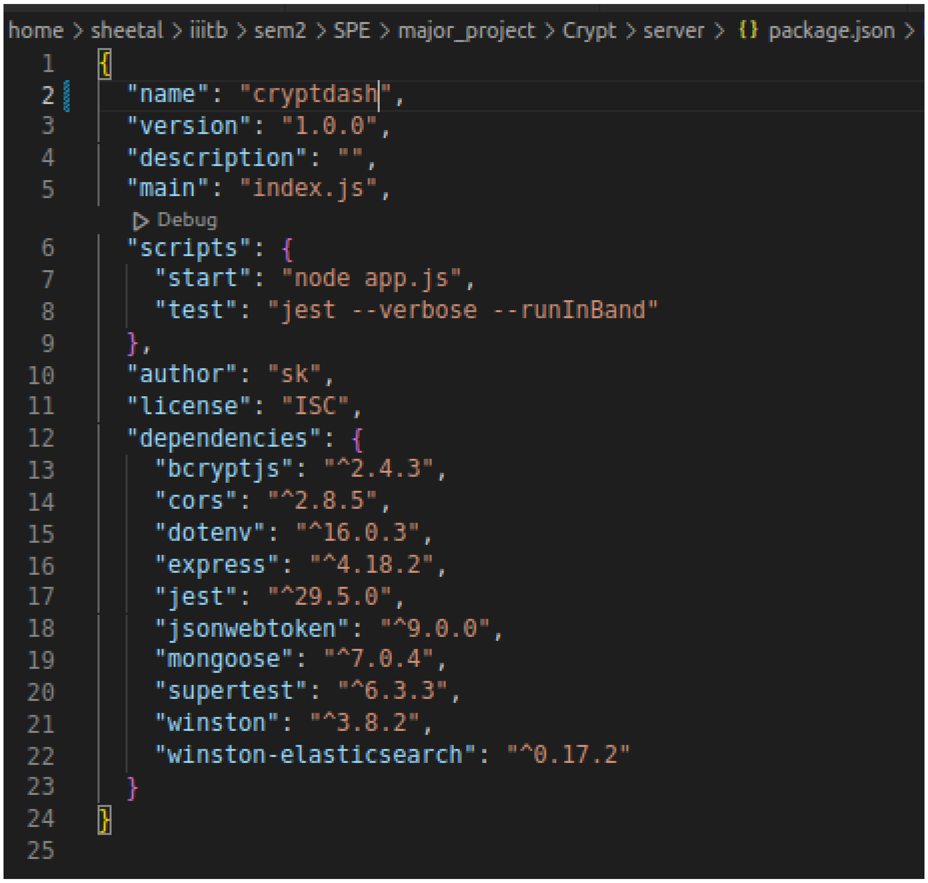
Here is a snippet of the *watchlists_api.test.js* file which shows how we use the beforeEach() method to login as an existing user and perform authorized operations in the tests with the JWT token.


home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > server > tests > JS user_api.test.js > de

```
const { JsonWebTokenError } = require('jsonwebtoken')
const mongoose = require('mongoose')
const supertest = require('supertest')
const UserInfo = require('../userDetails')
const helper = require('./test_helper')
const app = ('../app')
const api = supertest(app)

beforeEach(async () => {
    await UserInfo.remove({})
    await UserInfo.insertMany(helper.initialUsers)
})

describe('login', () => {
    test('login successfully', async() => {
        const user = {
            'email': 'test@test.com',
            'password': 'test',
        }
        await api
            .post('/login-user')
            .send(user)
            .expect(204)

        const email = users.map(user => user.email)
        expect(email).toContain('test@test.com')
    })
})

```

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.


home > sheetal > iiitb > sem2 > SPE > major_project > Crypt > server > {} package.json >

```json
{
  "name": "cryptdash",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node app.js",
    "test": "jest --verbose --runInBand"
  },
  "author": "sk",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "jest": "^29.5.0",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.0.4",
    "supertest": "^6.3.3",
    "winston": "^3.8.2",
    "winston-elasticsearch": "^0.17.2"
  }
}
```

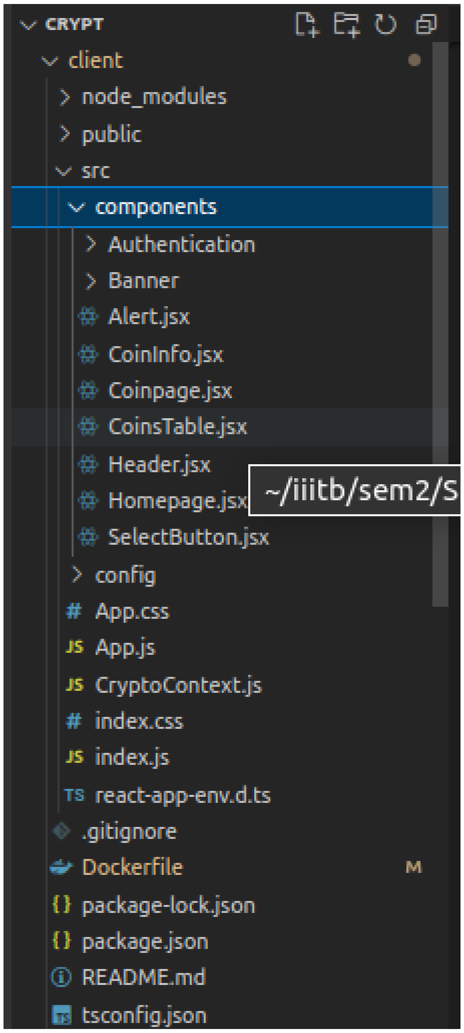
API Documentation

Endpoint	HTTP method	Input	Description
/api/register	POST	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-user	POST	name, password	User login → Check username → do password validation →Return token, name, username, id
/api/get-watchlists/:watc hlistid	GET	watchlist-id	Return watchlist data with given watchlist ID
/api/set-watchlists/:watc hlistid	POST	watchlist-id and array	Check if coin already exists in the given watchlist ans store it

5.4.2. Frontend

Directory structure for frontend

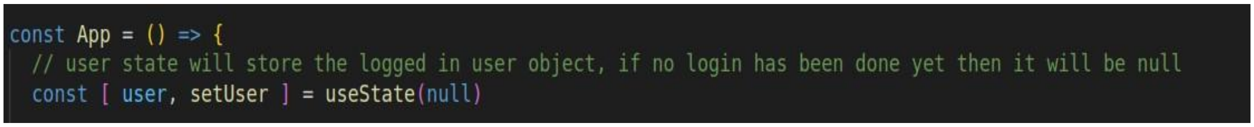
We use functional components, React state and React hooks in order to build the frontend. In order to enable routing we have used React Router (BrowserRouter).



```
CRYPT
  client
    node_modules
    public
    src
      components
        Authentication
        Banner
        Alert.jsx
        CoinInfo.jsx
        Coinpage.jsx
        CoinsTable.jsx
        Header.jsx
        Homepage.jsx
        SelectButton.jsx
      config
      App.css
      App.js
      CryptoContext.js
      index.css
      index.js
      react-app-env.d.ts
    .gitignore
    Dockerfile                      M
    package-lock.json
    package.json
    README.md
    tsconfig.json
```

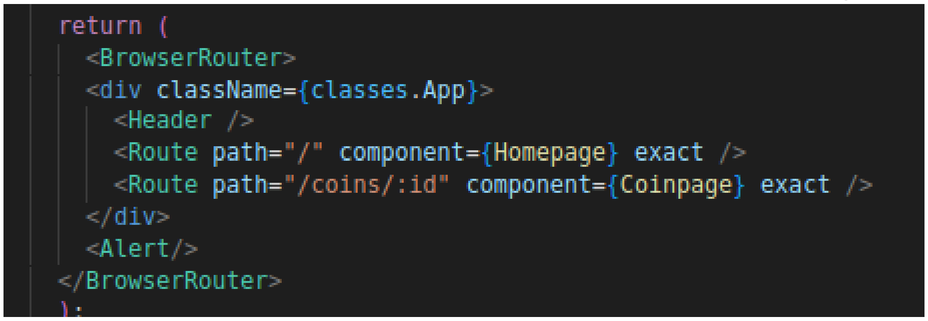
Components

Start of the App *component (App.js)*



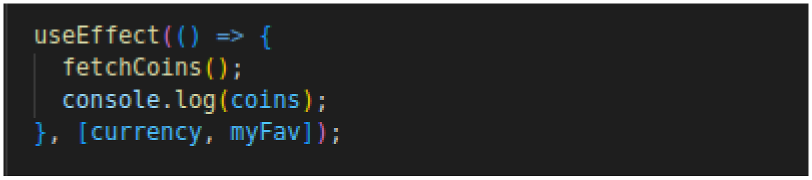
```
const App = () => {
  // user state will store the logged in user object, if no login has been done yet then it will be null
  const [ user, setUser ] = useState(null)
```

How React Router is used to match routes appropriately



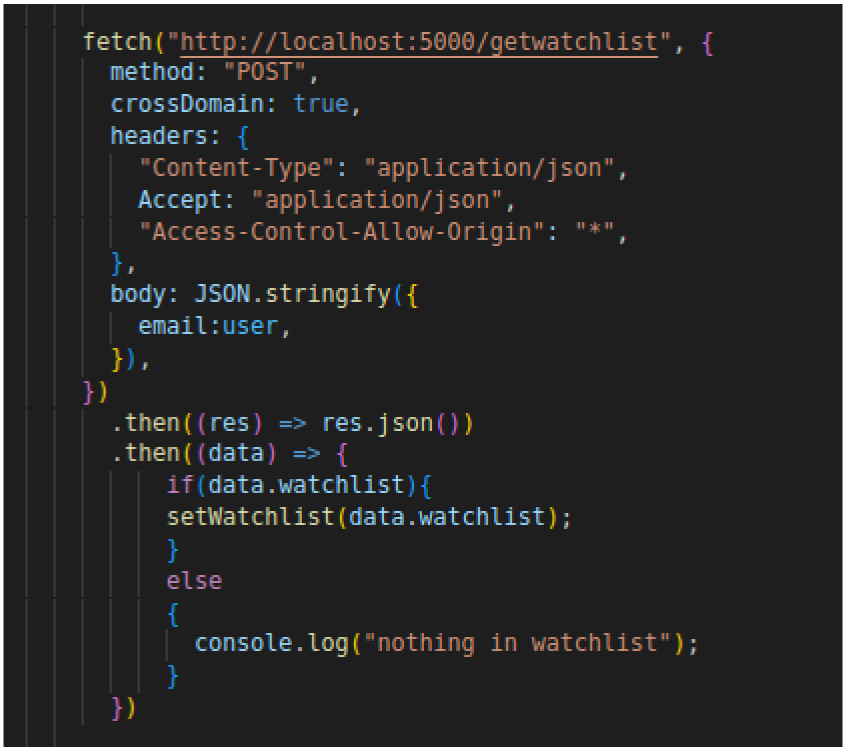
```
return (
  <BrowserRouter>
  <div className={classes.App}>
    <Header />
    <Route path="/" component={Homepage} exact />
    <Route path="/coins/:id" component={Coinpage} exact />
  </div>
  <Alert/>
</BrowserRouter>
);
```

Effect Hook usage



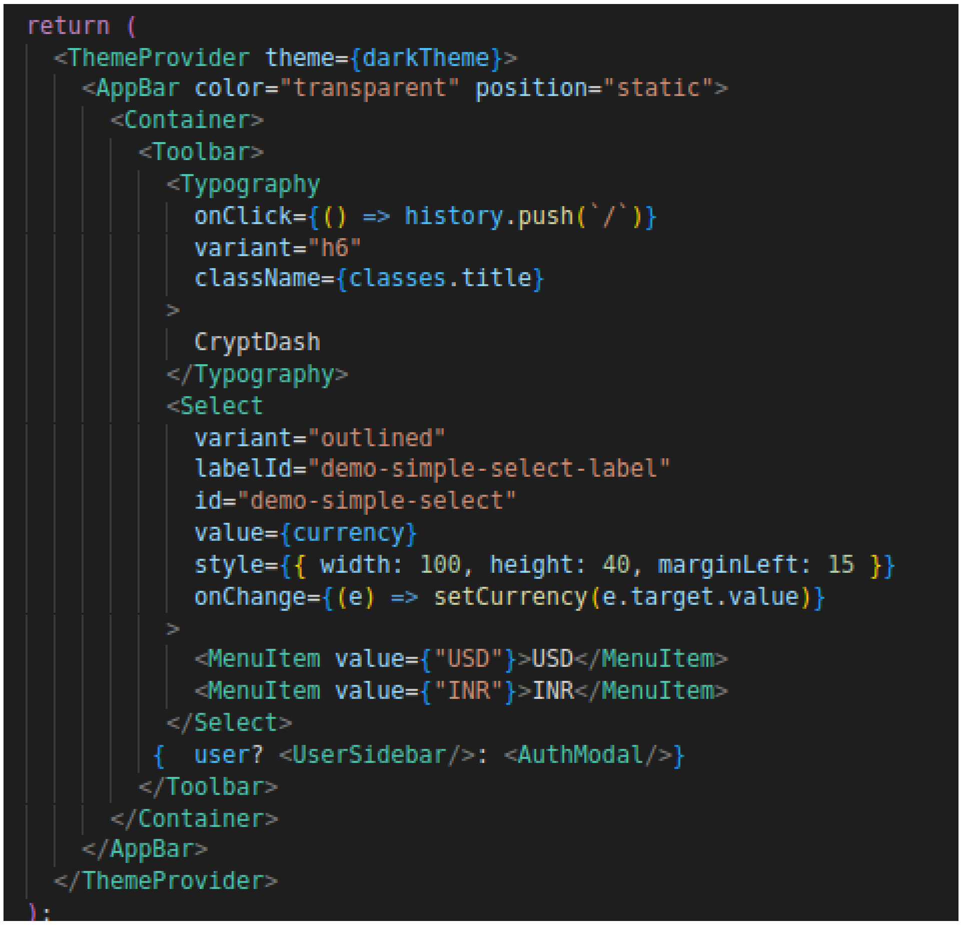
```
useEffect(() => {
  fetchCoins();
  console.log(coins);
}, [currency, myFav]);
```

Effect Hook + React Router to fetch correct watchlist data



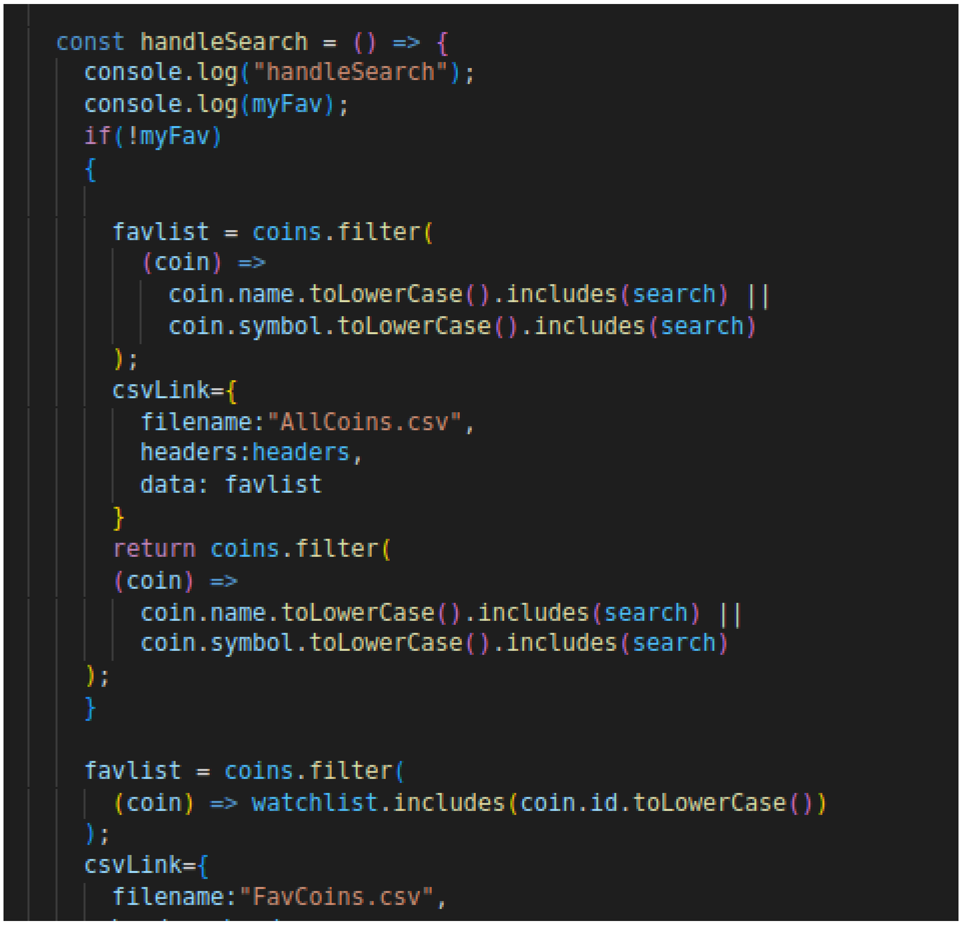
```
fetch("http://localhost:5000/getwatchlist", {
  method: "POST",
  crossDomain: true,
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    "Access-Control-Allow-Origin": "*",
  },
  body: JSON.stringify({
    email:user,
  }),
})
  .then((res) => res.json())
  .then((data) => {
    if(data.watchlist){
    setWatchlist(data.watchlist);
    }
    else
    {
      console.log("nothing in watchlist");
    }
  })
```

JSX that the *App* component returns



```
return (
  <ThemeProvider theme={darkTheme}>
    <AppBar color="transparent" position="static">
      <Container>
        <Toolbar>
          <Typography
            onClick={() => history.push(`/`)}
            variant="h6"
            className={classes.title}
          >
            CryptDash
          </Typography>
          <Select
            variant="outlined"
            labelId="demo-simple-select-label"
            id="demo-simple-select"
            value={currency}
            style={{ width: 100, height: 40, marginLeft: 15 }}
            onChange={(e) => setCurrency(e.target.value)}
          >
            <MenuItem value={"USD"}>USD</MenuItem>
            <MenuItem value={"INR"}>INR</MenuItem>
          </Select>
         {  user? <UserSidebar/>: <AuthModal/>}
        </Toolbar>
      </Container>
    </AppBar>
  </ThemeProvider>
);
```

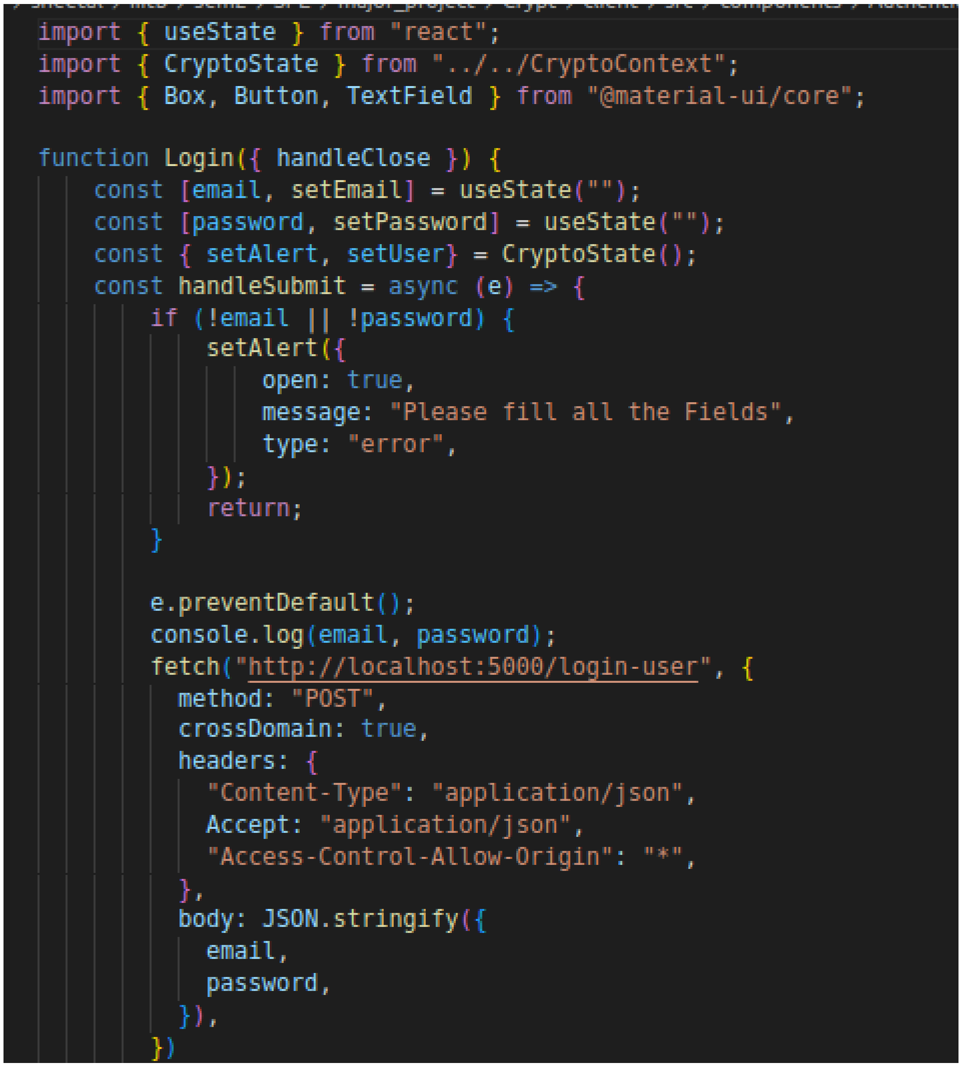
Search functionality



```
const handleSearch = () => {
  console.log("handleSearch");
  console.log(myFav);
  if(!myFav)
  {

    favlist = coins.filter(
      (coin) =>
        coin.name.toLowerCase().includes(search) ||
        coin.symbol.toLowerCase().includes(search)
    );
    csvLink={
      filename:"AllCoins.csv",
      headers:headers,
      data: favlist
    }
    return coins.filter(
    (coin) =>
      coin.name.toLowerCase().includes(search) ||
      coin.symbol.toLowerCase().includes(search)
  );
  }

  favlist = coins.filter(
    (coin) => watchlist.includes(coin.id.toLowerCase())
  );
  csvLink={
    filename:"FavCoins.csv",
```

Services

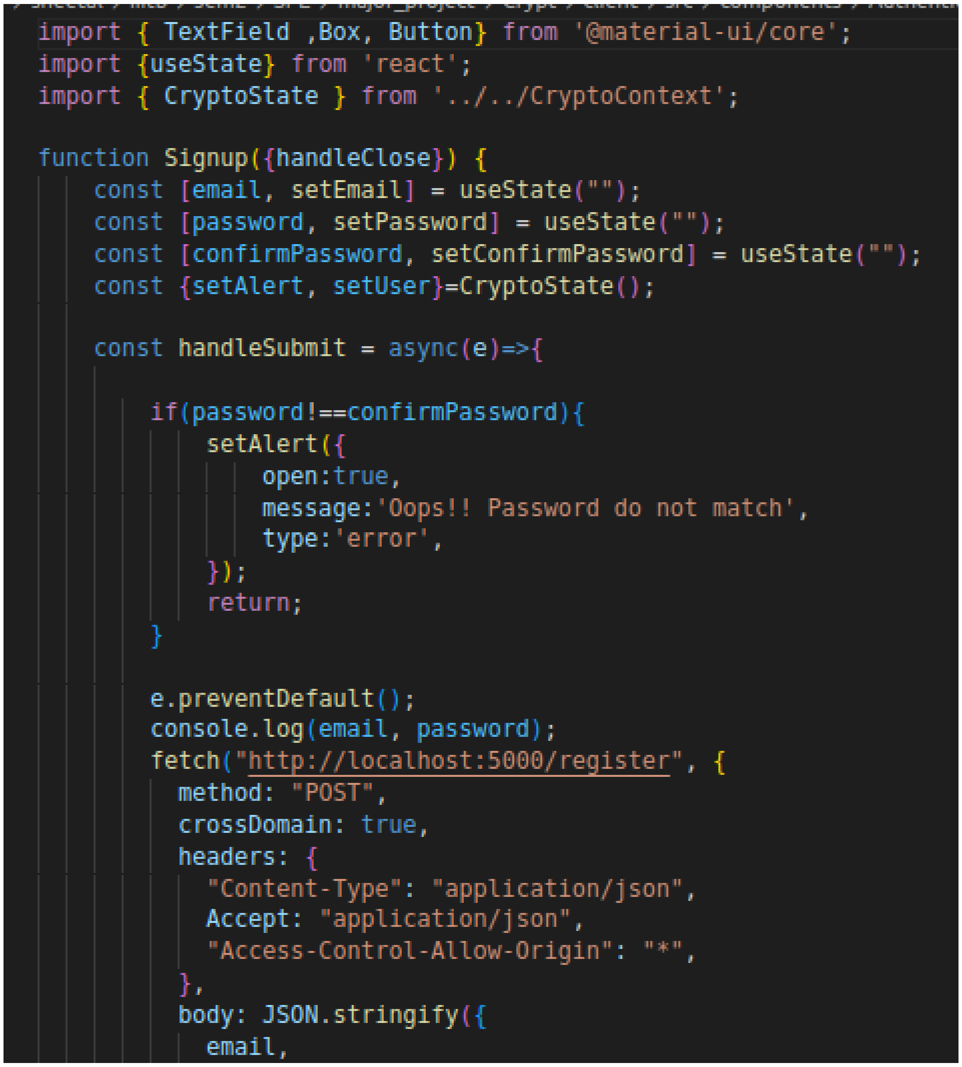
Login & Signup Service



```
import { useState } from "react";
import { CryptoState } from "../../CryptoContext";
import { Box, Button, TextField } from "@material-ui/core";

function Login({ handleClose }) {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const { setAlert, setUser} = CryptoState();
    const handleSubmit = async (e) => {
        if (!email || !password) {
            setAlert({
                open: true,
                message: "Please fill all the Fields",
                type: "error",
            });
            return;
        }

        e.preventDefault();
        console.log(email, password);
        fetch("http://localhost:5000/login-user", {
          method: "POST",
          crossDomain: true,
          headers: {
            "Content-Type": "application/json",
            Accept: "application/json",
            "Access-Control-Allow-Origin": "*",
          },
          body: JSON.stringify({
            email,
            password,
          }),
        })
```

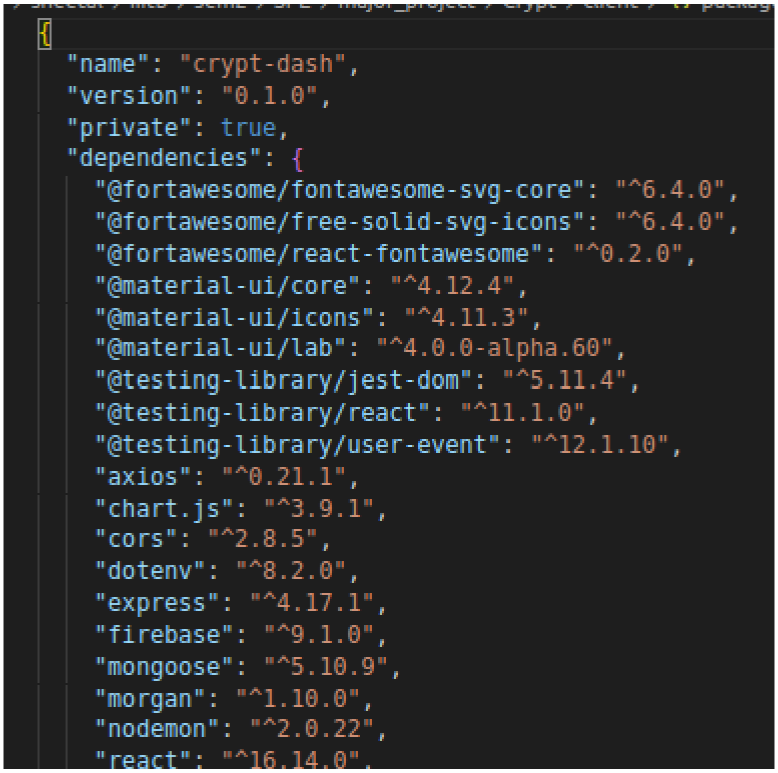
Sign up

Similar structure is then followed for all other required services.



```
import { TextField ,Box, Button} from '@material-ui/core';
import {useState} from 'react';
import { CryptoState } from '../../CryptoContext';

function Signup({handleClose}) {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [confirmPassword, setConfirmPassword] = useState("");
    const {setAlert, setUser}=CryptoState();

    const handleSubmit = async(e)=>{

        if(password!==confirmPassword){
            setAlert({
                open:true,
                message:'Oops!! Password do not match',
                type:'error',
            });
            return;
        }

        e.preventDefault();
        console.log(email, password);
        fetch("http://localhost:5000/register", {
          method: "POST",
          crossDomain: true,
          headers: {
            "Content-Type": "application/json",
            Accept: "application/json",
            "Access-Control-Allow-Origin": "*",
          },
          body: JSON.stringify({
            email,
```

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

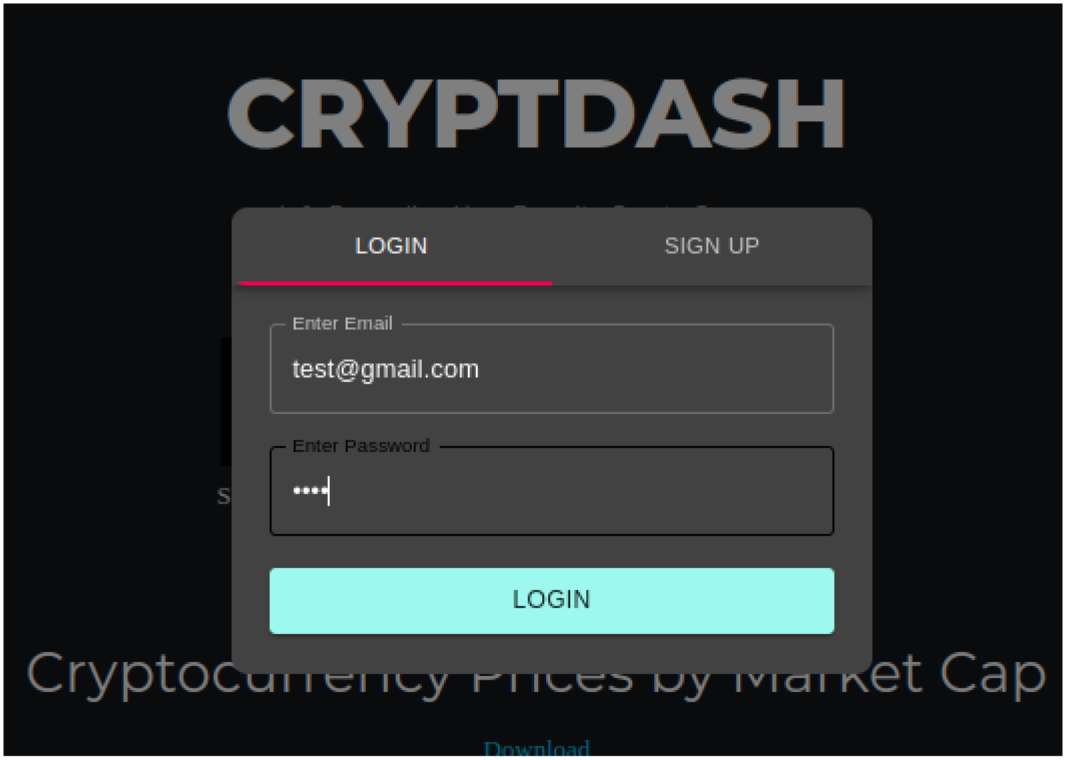


```
{
  "name": "crypt-dash",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^6.4.0",
    "@fortawesome/free-solid-svg-icons": "^6.4.0",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@material-ui/core": "^4.12.4",
    "@material-ui/icons": "^4.11.3",
    "@material-ui/lab": "^4.0.0-alpha.60",
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "axios": "^0.21.1",
    "chart.js": "^3.9.1",
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "firebase": "^9.1.0",
    "mongoose": "^5.10.9",
    "morgan": "^1.10.0",
    "nodemon": "^2.0.22",
    "react": "^16.14.0",
```

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.

5. Result and Discussion

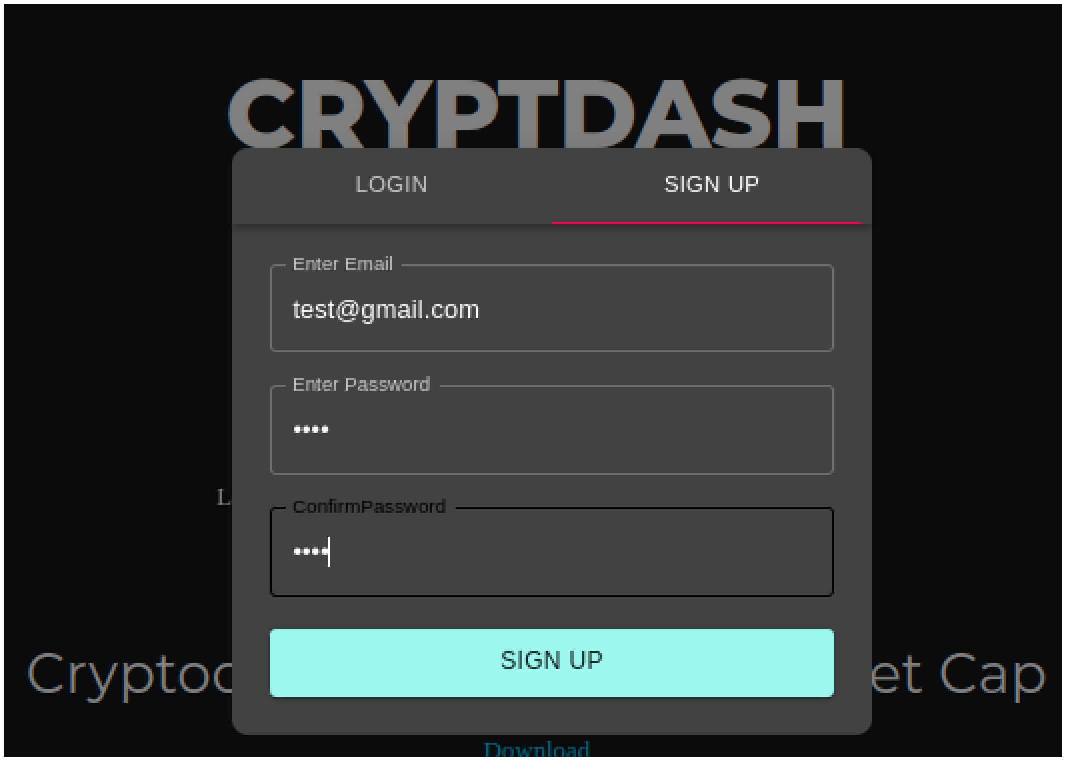
5.1. Login Page



Enter Email: test@gmail.com

Enter Password: ••••

5.2. Signup Page

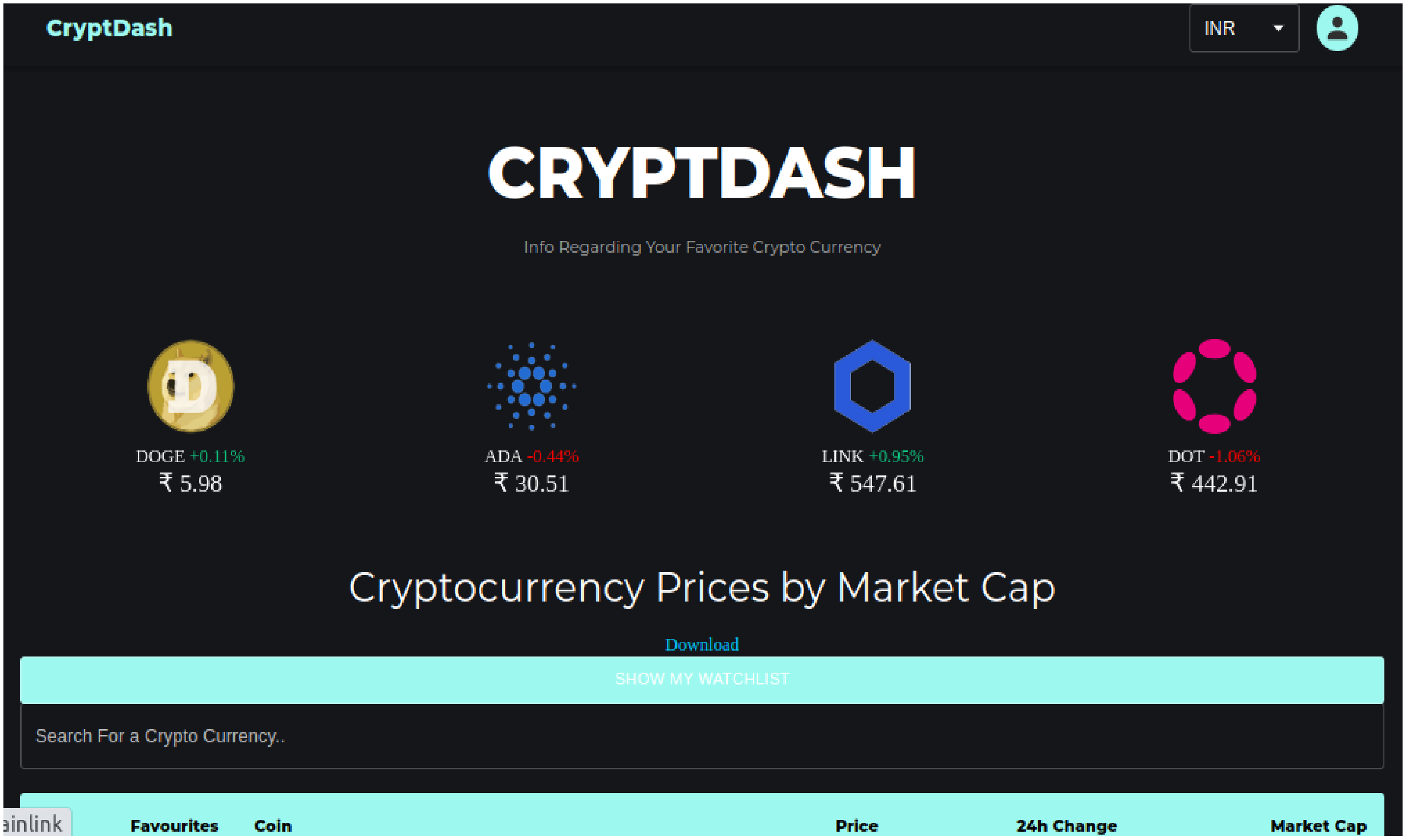


Enter Email: test@gmail.com

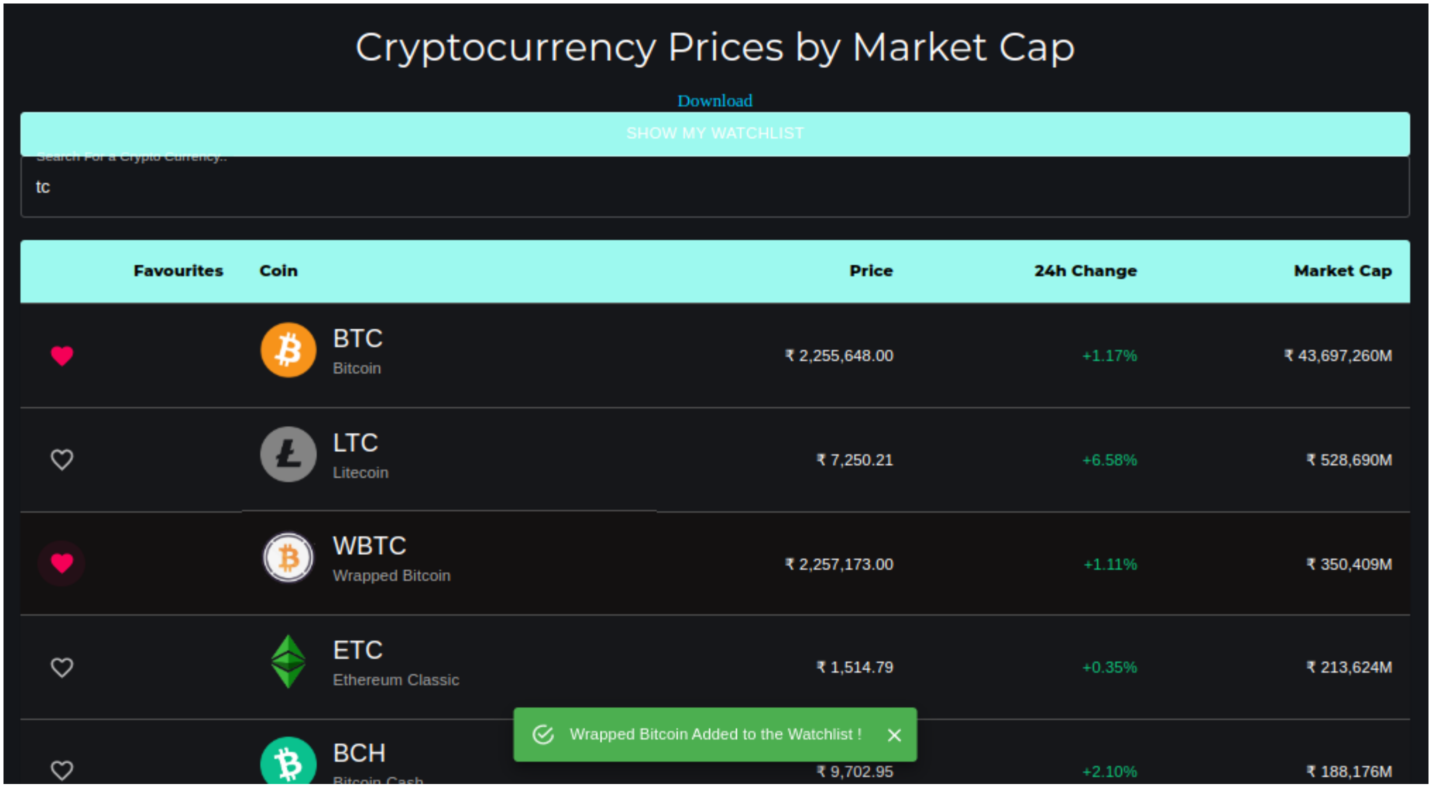
Enter Password: ••••

ConfirmPassword: ••••

5.3. Home Page



5.4. Table to show all cryptocurrency

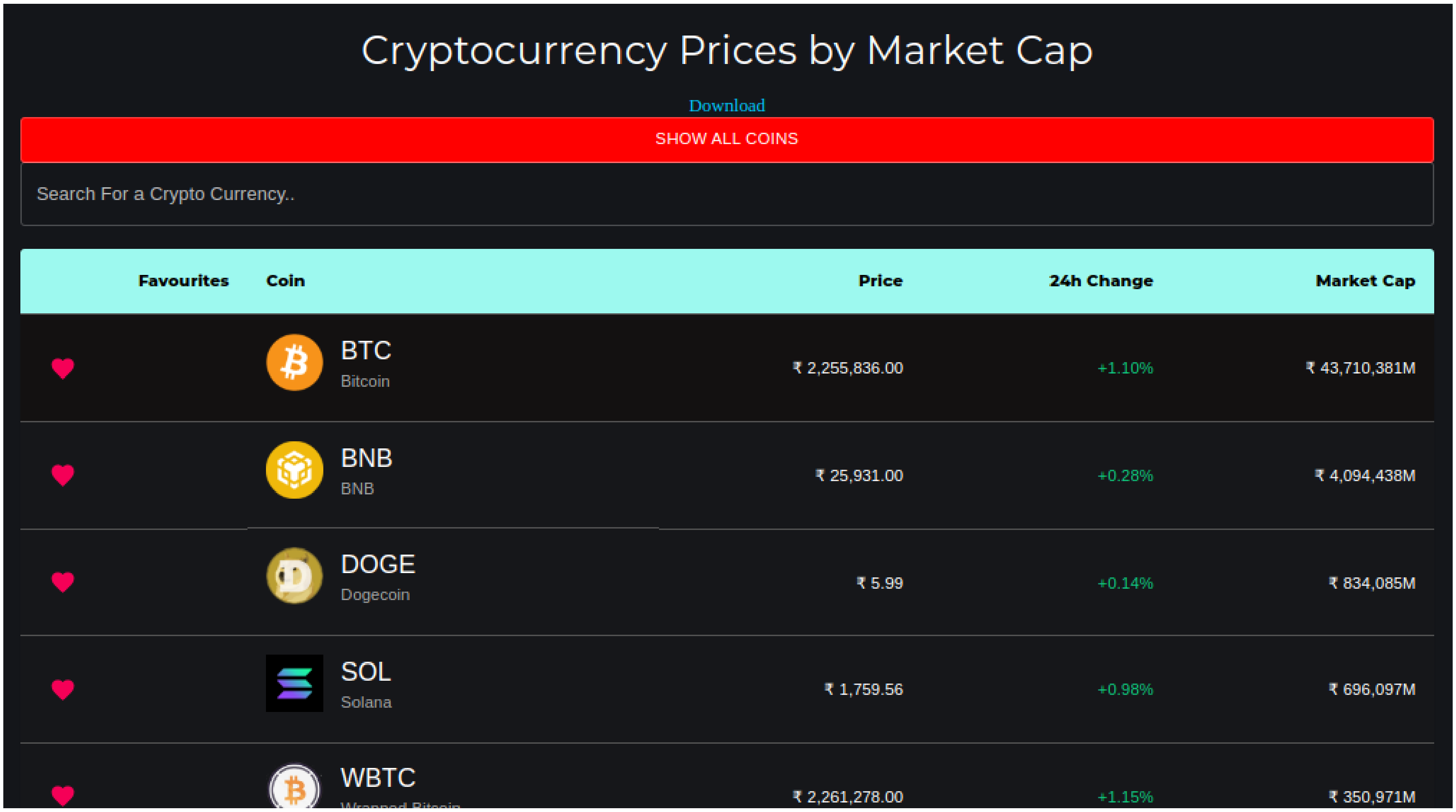


Cryptocurrency Prices by Market Cap
Download
SHOW MY WATCHLIST
Search For a Crypto Currency..
tc

Favourites	Coin	Price	24h Change	Market Cap
♥	BTC Bitcoin	₹ 2,255,648.00	+1.17%	₹ 43,697,260M
♡	LTC Litecoin	₹ 7,250.21	+6.58%	₹ 528,690M
♥	WBTC Wrapped Bitcoin	₹ 2,257,173.00	+1.11%	₹ 350,409M
♡	ETC Ethereum Classic	₹ 1,514.79	+0.35%	₹ 213,624M
♡	BCH Bitcoin Cash	₹ 9,702.95	+2.10%	₹ 188,176M

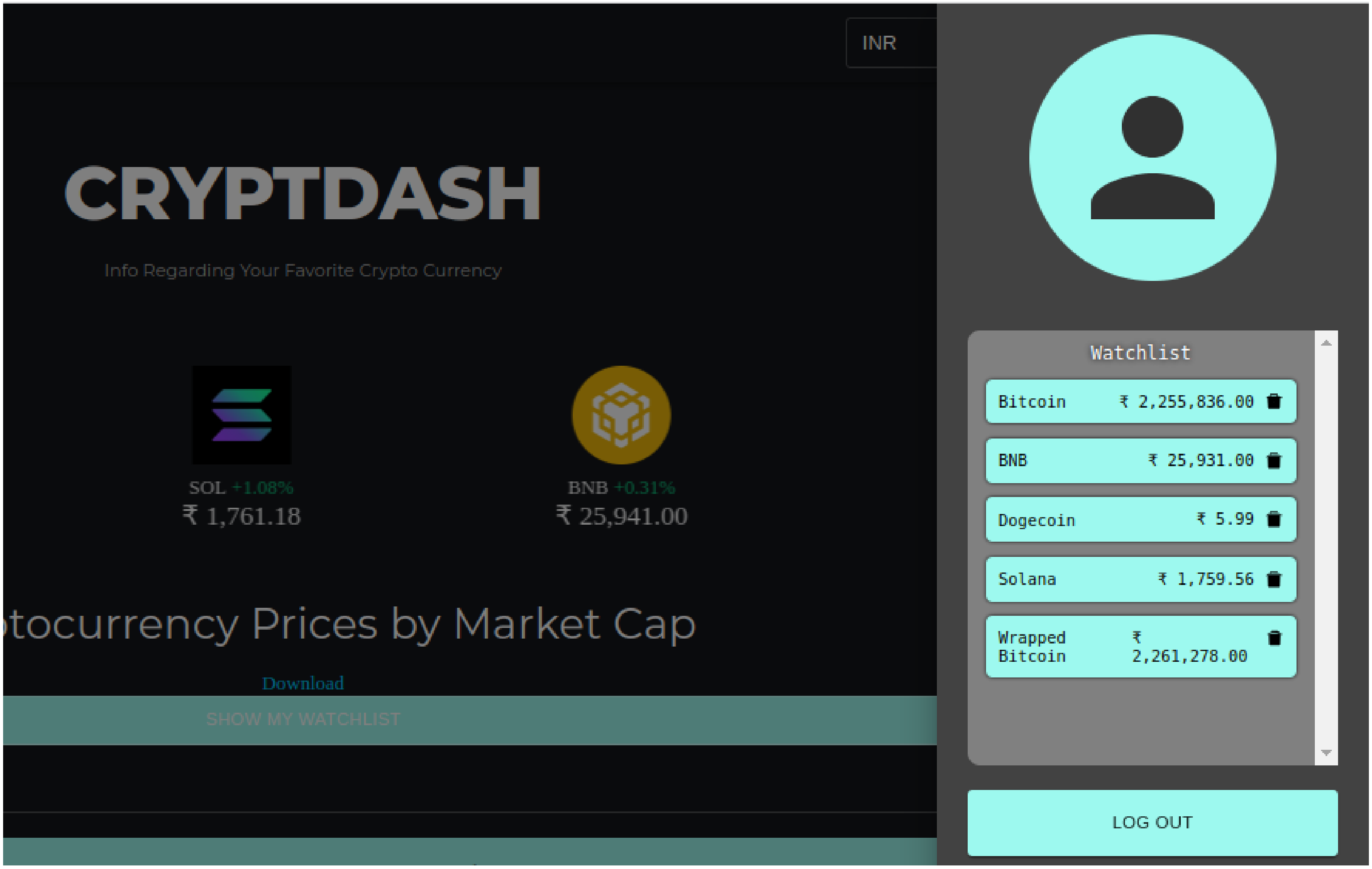
Wrapped Bitcoin Added to the Watchlist !

5.5. Table to show favourite cryptocurrency



Cryptocurrency Prices by Market Cap

Favourites	Coin	Price	24h Change	Market Cap
♥	BTC Bitcoin	₹ 2,255,836.00	+1.10%	₹ 43,710,381M
♥	BNB BNB	₹ 25,931.00	+0.28%	₹ 4,094,438M
♥	DOGE Dogecoin	₹ 5.99	+0.14%	₹ 834,085M
♥	SOL Solana	₹ 1,759.56	+0.98%	₹ 696,097M
♥	WBTC Wrapped Bitcoin	₹ 2,261,278.00	+1.15%	₹ 350,971M



CRYPTDASH

Info Regarding Your Favorite Crypto Currency

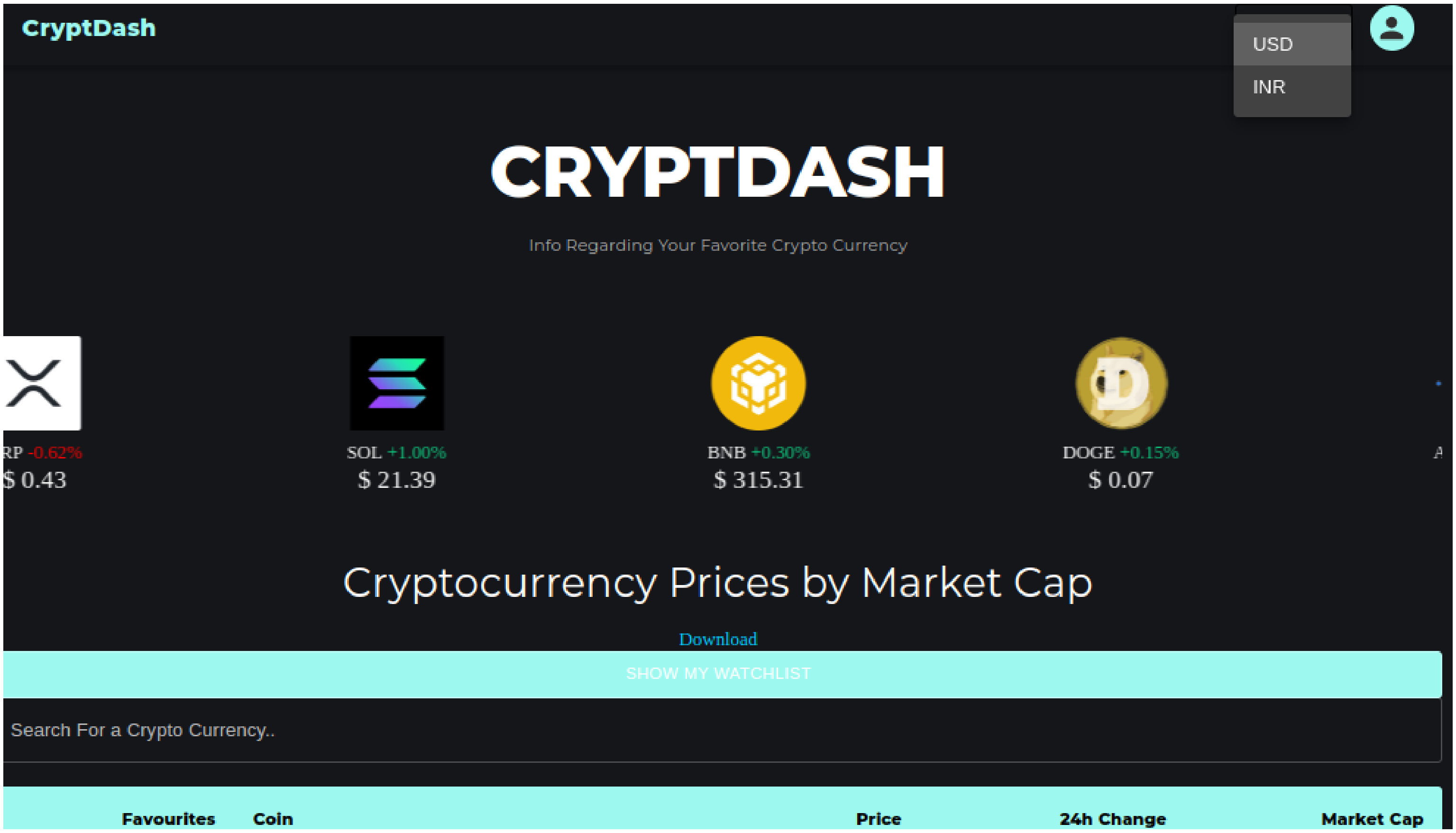
SOL +1.08%
₹ 1,761.18

BNB +0.31%
₹ 25,941.00

tocurrency Prices by Market Cap

Watchlist

Bitcoin	₹ 2,255,836.00
BNB	₹ 25,931.00
Dogecoin	₹ 5.99
Solana	₹ 1,759.56
Wrapped Bitcoin	₹ 2,261,278.00



CRYPTDASH

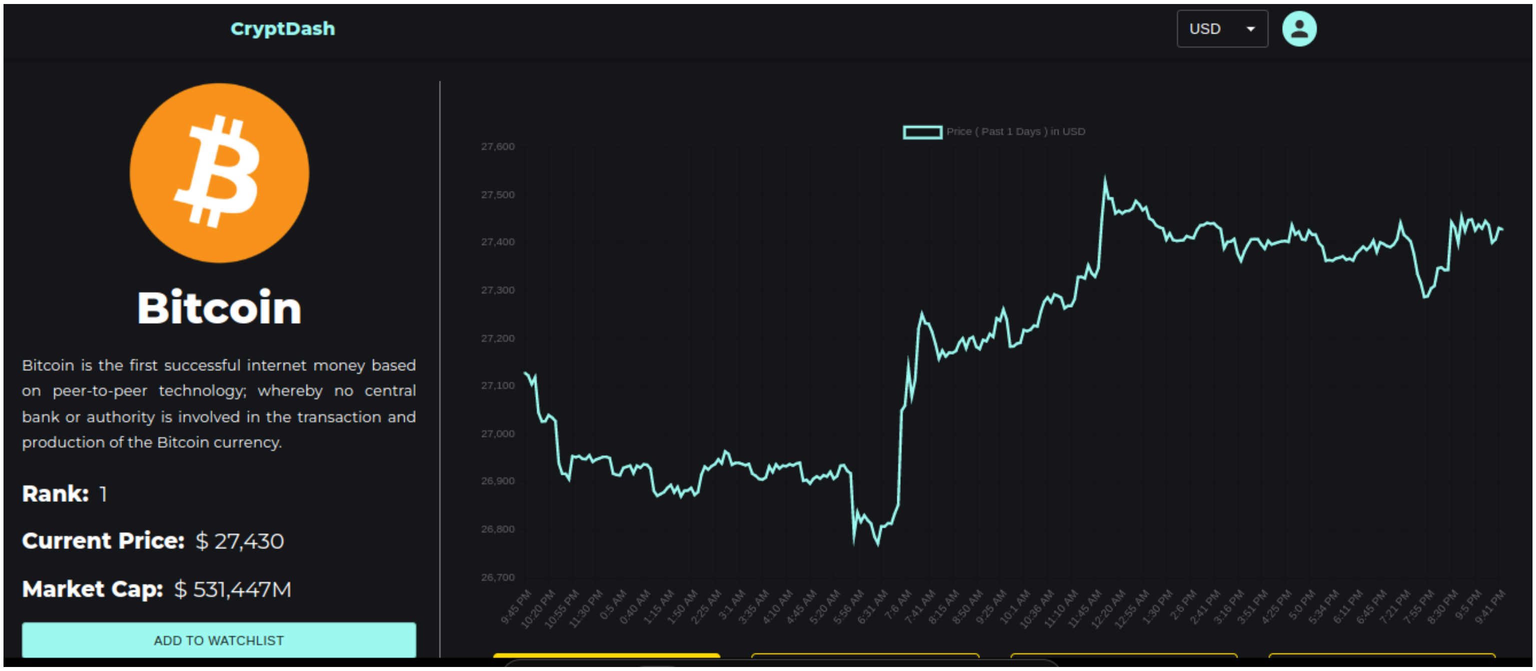
Info Regarding Your Favorite Crypto Currency

RP -0.62% $ 0.43	SOL +1.00% $ 21.39	BNB +0.30% $ 315.31	DOGE +0.15% $ 0.07

Cryptocurrency Prices by Market Cap

Download

Favourites	Coin	Price	24h Change	Market Cap



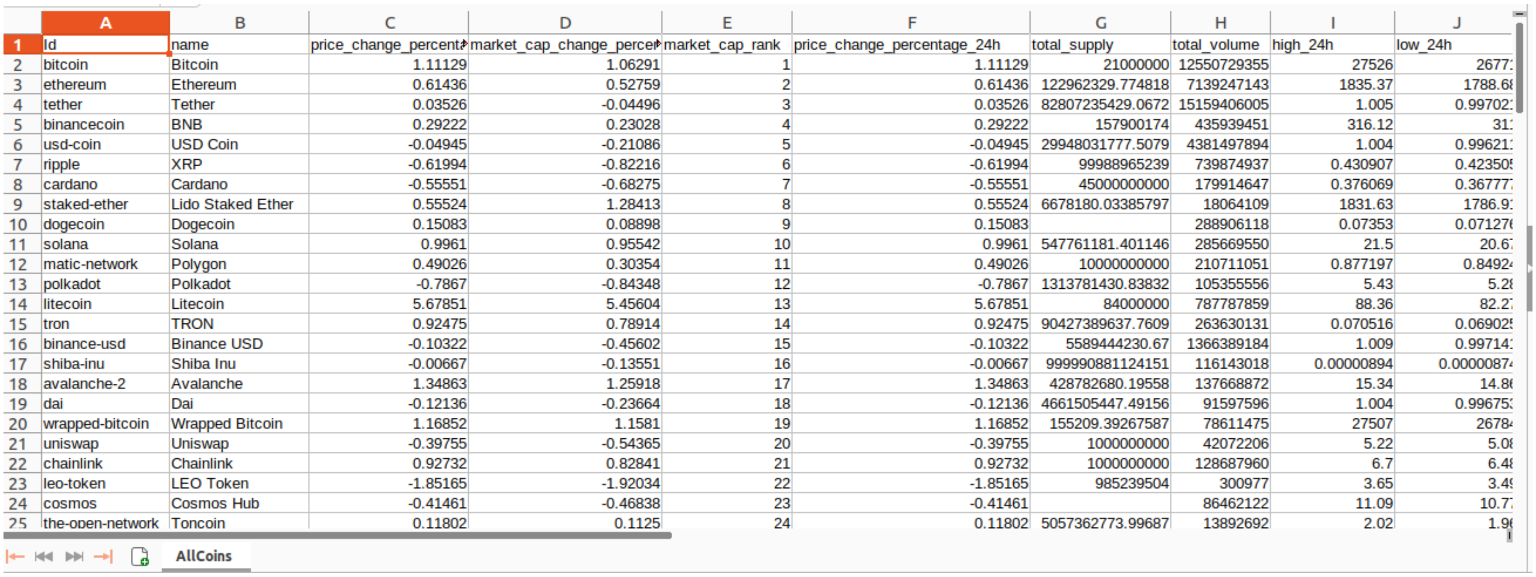
Bitcoin

Bitcoin is the first successful internet money based on peer-to-peer technology; whereby no central bank or authority is involved in the transaction and production of the Bitcoin currency.

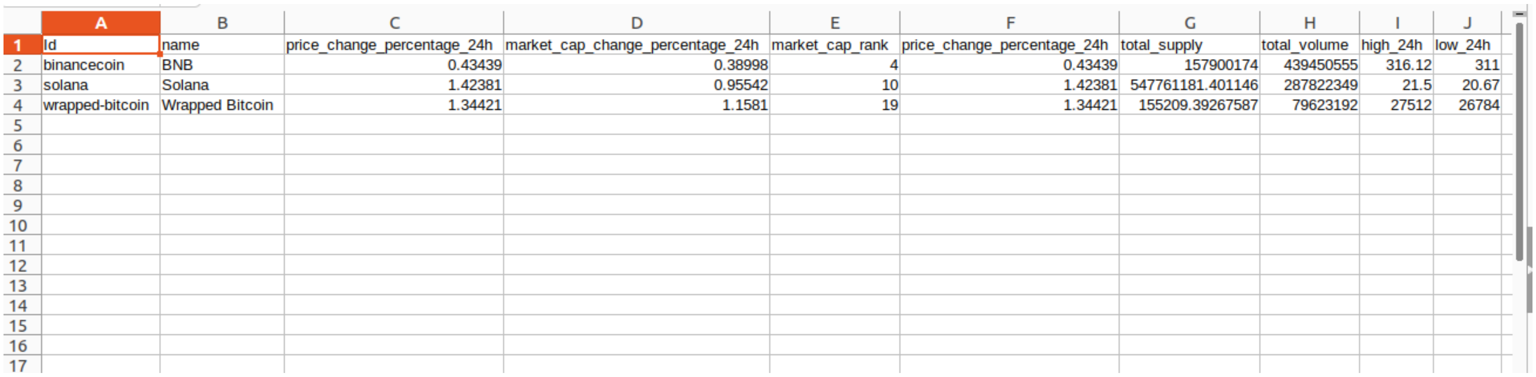
Rank: 1

Current Price: $ 27,430

Market Cap: $ 531,447M



	A	B	C	D	E	F	G	H	I	J
1	Id	name	price_change_percenta	market_cap_change_percer	market_cap_rank	price_change_percentage_24h	total_supply	total_volume	high_24h	low_24h
2	bitcoin	Bitcoin	1.11129	1.06291	1	1.11129	21000000	12550729355	27526	2677
3	ethereum	Ethereum	0.61436	0.52759	2	0.61436	122962329.774818	7139247143	1835.37	1788.6
4	tether	Tether	0.03526	-0.04496	3	0.03526	82807235429.0672	15159406005	1.005	0.99702
5	binancecoin	BNB	0.29222	0.23028	4	0.29222	157900174	435939451	316.12	31
6	usd-coin	USD Coin	-0.04945	-0.21086	5	-0.04945	29948031777.5079	4381497894	1.004	0.99621
7	ripple	XRP	-0.61994	-0.82216	6	-0.61994	99988965239	739874937	0.430907	0.42350
8	cardano	Cardano	-0.55551	-0.68275	7	-0.55551	45000000000	179914647	0.376069	0.36777
9	staked-ether	Lido Staked Ether	0.55524	1.28413	8	0.55524	6678180.03385797	18064109	1831.63	1786.9
10	dogecoin	Dogecoin	0.15083	0.08898	9	0.15083		288906118	0.07353	0.07127
11	solana	Solana	0.9961	0.95542	10	0.9961	547761181.401146	285669550	21.5	20.6
12	matic-network	Polygon	0.49026	0.30354	11	0.49026	10000000000	210711051	0.877197	0.8492
13	polkadot	Polkadot	-0.7867	-0.84348	12	-0.7867	1313781430.83832	105355556	5.43	5.2
14	litecoin	Litecoin	5.67851	5.45604	13	5.67851	84000000	787787859	88.36	82.2
15	tron	TRON	0.92475	0.78914	14	0.92475	90427389637.7609	263630131	0.070516	0.06902
16	binance-usd	Binance USD	-0.10322	-0.45602	15	-0.10322	5589444230.67	1366389184	1.009	0.99714
17	shiba-inu	Shiba Inu	-0.00667	-0.13551	16	-0.00667	999990881124151	116143018	0.00000894	0.0000087
18	avalanche-2	Avalanche	1.34863	1.25918	17	1.34863	428782680.19558	137668872	15.34	14.8
19	dai	Dai	-0.12136	-0.23664	18	-0.12136	4661505447.49156	91597596	1.004	0.99675
20	wrapped-bitcoin	Wrapped Bitcoin	1.16852	1.1581	19	1.16852	155209.39267587	78611475	27507	2678
21	uniswap	Uniswap	-0.39755	-0.54365	20	-0.39755	1000000000	42072206	5.22	5.0
22	chainlink	Chainlink	0.92732	0.82841	21	0.92732	1000000000	128687960	6.7	6.4
23	leo-token	LEO Token	-1.85165	-1.92034	22	-1.85165	985239504	300977	3.65	3.4
24	cosmos	Cosmos Hub	-0.41461	-0.46838	23	-0.41461		86462122	11.09	10.7
25	the-open-network	Toncoin	0.11802	0.1125	24	0.11802	5057362773.99687	13892692	2.02	1.9



	A	B	C	D	E	F	G	H	I	J
1	Id	name	price_change_percentage_24h	market_cap_change_percentage_24h	market_cap_rank	price_change_percentage_24h	total_supply	total_volume	high_24h	low_24h
2	binancecoin	BNB	0.43439	0.38998	4	0.43439	157900174	439450555	316.12	311
3	solana	Solana	1.42381	0.95542	10	1.42381	547761181.401146	287822349	21.5	20.67
4	wrapped-bitcoin	Wrapped Bitcoin	1.34421	1.1581	19	1.34421	155209.39267587	79623192	27512	26784
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										

6. Scope for Future Work

Rename a watchlist
Users can rename their already existing watchlists.

Rearrangement of coin in watchlist
Users can rearrange the coin in their watchlists as per their priority.

Delete User and its watchlist from DB

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.

7. Conclusion

We have successfully created the web-app *cryptdash* that will be helpful for users to make better investment decisions.

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.

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.

8. 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/