

# ***STOCK MATRIX***

## **Advanced Web Programming**

### **Project Report**

**Project Title:** *Stock Matrix – Stock Market Simulation Website*

**Description:**

*Stock Matrix is a React-based stock market simulation website with MongoDB integration. It offers a risk-free trading experience, emulating real market conditions. Users can create accounts, manage portfolios, and utilize a risk assessment tool. Future plans include Alphavantage premium API integration for expanded data access. The platform also features a personalized watchlist, educational resources, and real-time news updates. With a user-friendly interface and responsive design, Stock Matrix aims to empower individuals with practical trading skills. Ongoing developments include advanced charting tools and machine learning-driven insights for an enriched trading experience.*

**Team Members:**

*Dev Vora;*

*Tirth Mehta.*

**Technology Framework:**

*1. Frontend:*

React: A popular JavaScript library for building user interfaces, known for its component-based architecture and efficient rendering.

*2. Backend:*

Node.js: A JavaScript runtime environment that allows for server-side execution of JavaScript code.

Express.js: A web application framework for Node.js, used for building robust and scalable backend applications.

*3. Database:*

MongoDB: A NoSQL database that provides flexibility in handling unstructured data, making it suitable for applications with dynamic data requirements.

*4. Authentication:*

JWT (JSON Web Tokens): A standard for securely transmitting information between parties as a JSON object. It's commonly used for authentication.

*5. HTTP Client:*

Axios: A promise-based HTTP client for making HTTP requests, commonly used for API communication.

#### 6. *Version Control:*

Git: A distributed version control system used for tracking changes and collaborating on code.

#### 7. *Package Management:*

npm: Tools for managing project dependencies and scripts.

#### 8. *Frontend Design Framework:*

Material-UI: Pre-built UI components and styling libraries to streamline frontend development.

#### 9. *API Integration (Future Plan):*

Alphavantage API: A financial data API that provides access to a wide range of market data, including stock quotes, technical indicators, and more.

By leveraging these technologies, the Stock Matrix project has achieved a robust, scalable, and user-friendly stock market simulation platform.

### **Abstract:**

*"Stock Matrix" is an innovative React-based stock market simulation platform with MongoDB integration. This project enables users to engage in risk-free trading, replicating real market conditions. It encompasses essential account features, portfolio management, and a powerful risk assessment tool. Future enhancements involve integrating Alphavantage's premium API for broader data access. The platform also offers a personalized watchlist, educational resources, and real-time news updates. With its user-friendly interface and responsiveness, Stock Matrix seeks to equip users with practical trading expertise. Ongoing development plans include advanced charting tools and machine learning-driven insights, further enhancing the trading experience.*

## 1. Introduction

The Stock Matrix project is a cutting-edge stock market simulation website built using the React framework and backed by MongoDB as its database. It provides users with an immersive experience in the world of trading without the need to invest real money. The platform aims to empower individuals with the knowledge and skills needed to make informed investment decisions.

### **Key Features:**

#### *1. Realistic Stock Market Simulation:*

Users can engage in simulated trading activities with real-time stock market data.

#### *2. Basic Account Features:*

Users can create accounts, log in, and securely manage their profiles.

Password reset and account recovery options are available for user convenience.

#### *3. Risk Management Tool:*

A robust risk management tool assists users in assessing and mitigating potential investment risks.

#### *4. Watchlist:*

Users can curate a personalized watchlist to monitor specific stocks or investment opportunities of interest.

#### *5. Portfolio Management:*

Users can track and manage their virtual portfolios, allowing them to monitor the performance of their simulated investments.

#### *6. Educational Resources:*

The platform provides educational content, including articles, tutorials, and videos, to help users enhance their understanding of the stock market.

#### *7. News Feed Integration:*

Real-time news updates and financial reports are integrated to keep users informed about market trends and events.

#### *8. Historical Data Analysis:*

Users can access historical stock data for in-depth analysis and research purposes.

#### *9. User Notifications:*

Users receive notifications for significant events, such as price alerts or news updates related to their watchlist.

*10. Responsive Design:*

The website is designed to be fully responsive, ensuring a seamless experience across various devices and screen sizes.

*11. Social Sharing Integration (Future Plan):*

Users will have the option to share their simulated trades or investment strategies on social media platforms.

*12. Alphavantage Premium API Integration (Future Plan):*

The platform plans to integrate the Alphavantage premium API for enhanced data retrieval, enabling access to a wide range of financial information and market analytics.

*13. Community Forum (Future Plan):*

A community forum will be implemented to facilitate discussions, knowledge sharing, and networking among users.

**Future Developments:**

The project has plans for continuous improvement and expansion, including the implementation of additional features such as advanced charting tools, machine learning-based investment insights, and integration with popular trading platforms.

By combining an intuitive user interface with comprehensive market data and educational resources, Stock Matrix aims to be a valuable platform for both novice and experienced traders looking to hone their skills and make informed investment decisions in a risk-free environment.

## 2. System Architecture

The system architecture for the Stock Matrix - Stock Market Simulation Website encompasses both frontend and backend components, ensuring scalability, responsiveness, and efficient data handling. Here's a breakdown of the architecture:

### 1. *Frontend:*

- **React Components:** The frontend is built using React, a component-based JavaScript library. Each component manages its state and rendering, facilitating a modular and reusable codebase.
- **HTTP Requests:** Axios, or a similar HTTP client, handles requests to the backend API for data retrieval and updates.
- **Routing:** React Router or a similar library manages client-side routing for navigation within the application.
- **User Interface (UI) Framework:** Material-UI frameworks provide pre-built UI components for a polished and consistent look.

### 2. *Backend:*

- **Node.js and Express.js:** The backend is powered by Node.js, allowing server-side JavaScript execution. Express.js provides a robust framework for building the server and handling HTTP requests.
- **API Endpoints:** The backend defines RESTful API endpoints to handle various requests from the frontend, such as user authentication, portfolio management, watchlist updates, and data retrieval.
- **MongoDB Database:** MongoDB, a NoSQL database, stores user account information, portfolio data, watchlists, and other application data. It provides flexibility in handling dynamic and unstructured data.
- **Authentication Middleware:** JWT (JSON Web Tokens) is implemented for secure user authentication and authorization.
- **Integration with Alphavantage API (Future Plan):** For enhanced market data, the backend will integrate with the Alphavantage premium API to fetch real-time stock information.
- **Error Handling and Validation:** Middleware and error handling mechanisms are in place to ensure proper validation of incoming requests and appropriate error responses.

### 3. *Deployment and Hosting:*

- The application is deployed using a Platform as a Service (PaaS) provider like Heroku, Vercel, or AWS Amplify. These platforms simplify deployment and offer scalability options.

### 4. *Database Management:*

- MongoDB Atlas or a similar cloud-based database service is used to manage the MongoDB database. This ensures data durability, high availability, and automated backups.

### 5. *Authentication and Security:*

- JWT tokens are securely generated and validated to authenticate users. Passwords are hashed using bcrypt for added security.

**6. *Monitoring and Analytics (Future Plan):***

- Tools like New Relic or custom logging solutions are implemented for monitoring application performance, tracking user interactions, and generating insights.

This architecture ensures a responsive, secure, and scalable platform for users to engage in simulated stock market trading. Future plans include integrating additional features and tools to enhance the user experience and provide valuable insights into market trends.

### 3. Code

#### 1. Registration Page

Code:

```
import React, { useEffect, useState } from "react";
import Input from "../../components/Input";
import { Link, useNavigate } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { userRegisterThunk } from "../../features/user/auth";
import Loading from "../../components/Loading";

// Regular expressions for input validation
const EMAIL_REGEX = /^[a-z0-9._]+@[a-z]+\.[a-z]{2,4}$/;
const NAME_REGEX = /^[A-Za-z]+[\sA-Za-z]{5,20}$/;
const PASSWORD_REGEX = /^[A-Za-z0-9!@#%&^*]{2,20}$/;
const PHONE_REGEX = /^[0-9]{10,11}$/;

const Register = ({ showAlert }) => {
  const [fullName, setFullName] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");

  // Validation state variables
  const [checkEmail, setCheckEmail] = useState(false);
  const [checkName, setCheckName] = useState(false);
  const [checkPhone, setCheckPhone] = useState(false);
  const [checkPassword, setCheckPassword] = useState(false);
  const [matchPassword, setMatchPassword] = useState(false);

  // Focus state variables
  const [nameFocus, setNameFocus] = useState(false);
  const [emailFocus, setEmailFocus] = useState(false);
  const [phoneFocus, setPhoneFocus] = useState(false);
  const [passwordFocus, setPasswordFocus] = useState(false);
  const [confirmPasswordFocus, setConfirmPasswordFocus] =
    useState(false);

  const [error, setError] = useState("");

  // Validation checks using useEffect
  useEffect(() => {
    if (NAME_REGEX.test(fullName)) {
      setCheckName(true);
    } else {
      setCheckName(false);
    }
  })
```

```

    }, [fullName]));

    // Similar useEffect blocks for email, phone, password, and password
    match validation

    // Redux state and dispatch
    const { isLoading, isError, isSuccess, id } = useSelector(
      (state) => state.authReducer
    );
    const dispatch = useDispatch();
    const navigate = useNavigate();

    // Handling form submission
    const handleSubmit = (e) => {
      e.preventDefault();

```

Explanation:

1. The code imports necessary React libraries, components, and dependencies.
2. It defines state variables for user input fields (fullName, email, phone, password, and confirmPassword) and validation checks for these fields.
3. The component renders a registration form with input fields for user details. It includes validation checks for each input field and provides feedback to the user in case of invalid input.
4. Upon form submission, the code sends a user registration request to the server using the Redux store and dispatches the userRegisterThunk action. If the registration is successful, the user is redirected to the "/user/info" page.

## 2. Login Page

Code:

```

import React, { useEffect, useState } from "react";
import Input from "../../components/Input";
import { Link, useNavigate } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { clearUserAuthState, userLoginThunk } from
"../../features/user/auth";
import Loading from "../../components/Loading";

const Login = ({ showAlert }) => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  const { isSuccess, isError, isLoading, token } = useSelector(
    (state) => state.authReducer
  );

```



```
const dispatch = useDispatch();
const navigate = useNavigate();

const handleSubmit = (e) => {
  e.preventDefault();

  if (email === "" || password === "") {
    setError("Please Enter above field.");
    return;
  }

  let data = {
    email,
    password,
  };

  dispatch(userLoginThunk(data)).then((data) => {
    if (!data.payload?.success) {
      showAlert({
        show: true,
        type: "warning",
        message: data.payload?.message,
      });
      return;
    }
  });
};

useEffect(() => {
  if (token) {
    showAlert({
      show: true,
      type: "success",
      message: "Login Successfully.",
    });
    localStorage.setItem("token", token);
    dispatch(clearUserAuthState());
    navigate("/dashboard/home");
  }
});

return (
  // JSX for rendering the login form
);
};

export default Login;
```

Explanation:

1. The code imports necessary React libraries, components, and dependencies for the login page, including state management and routing.
2. The component defines state variables for user input fields (email and password) and an error message variable to handle input validation and error feedback.
3. Within the component, a Redux state is used to manage authentication-related data, including whether the login was successful, whether there's an error, and the user's token.
4. The useEffect hook listens for changes in the token state variable, which signifies a successful login. When a token is present, it sets an alert, stores the token in local storage, clears the authentication state, and navigates the user to the dashboard.
5. The component renders a login form with input fields for email and password, handles form submission, and dispatches a Redux action (userLoginThunk) to authenticate the user. It provides error feedback if the login is not successful.
6. The code includes links for registration and handles loading states with a loading spinner during the login process.

### 3. Email Verification

Code:

```
import { encode } from "js-base64";
import User from "../model/User.js";
import transporter from "../utils/emailSend.js";
import template from "../utils/emailTemplate.js";
import config from "../config.js";

export const sendEmailLogin = async (req, res) => {
  let success = false;

  try {
    const { id } = req.params;

    let user = await User.findOne({ _id: id });

    let token = encode(user._id.toString());

    await transporter.sendMail({
      from: "stockmatrix7@gmail.com",
      to: user.basic.email,
      subject: "Verify your account with StockMatrix",
      html: template(
        "Thank you for choosing <b>StockMatrix</b> as your Investment  
Partner. Please Verify the email to login and start investing today.",
        `${config.react_url}/verify/${token}`
      ),
    });

    success = true;

    return res.status(200).send({
```

```
        success,  
        message: "Email Successfully Sent.",  
    });  
    } catch (err) {  
        console.log(err);  
        return res.status(500).send({  
            success,  
            message: "Internal Server Error.",  
        });  
    }  
};
```

Explanation:

This code is an asynchronous function that sends a verification email to a user based on their provided ID. It imports necessary modules for encoding, user data, email transport, email templates, and configuration. The code retrieves a user by ID, encodes their ID into a token, and sends an email with a verification link to the user's email address. If successful, it responds with a 200 status and a success message; if there's an error, it logs the error and responds with a 500 status and an error message.

#### 4. KYC Page

Code:

```
import Kyc from "../model/Kyc.js";  
import User from "../model/User.js";  
import random from "../utils/randomHash.js";  
import multer from "multer";  
import fs from "fs";  
import path from "path";  
  
const storage = multer.diskStorage({  
    destination: async (req, file, cb) => {  
        cb(null, `public/uploads`);  
    },  
    filename: async (req, file, cb) => {  
        const uniqueName =  
            random(5) + new Date().getTime() + "_" + file.originalname;  
        cb(null, `${uniqueName}`);  
    },  
});  
  
const upload = multer({  
    storage,  
    limits: { fileSize: 1048576 * 10 },  
    fileFilter: (req, file, cb) => {  
        if (  
            file.mimetype == "image/png" ||  
            file.mimetype == "image/jpg" ||
```

```

        file.mimetype == "image/jpeg"
      ) {
        cb(null, true);
      } else {
        cb(null, false);
        const err = new Error("Only .png, .jpg and .jpeg format
allowed!");
        err.name = "ExtensionError";
        return cb(err);
      }
    },
  }).fields([
    {
      name: "proof-1",
      maxCount: 2,
    },
    {
      name: "proof-2",
      maxCount: 2,
    },
  ]
]);

```

Explanation:

The essential part of the code is the **addKyc** function, which handles the upload of KYC documents, checks for existing KYC records, and manages their approval status. It utilizes the **multer** middleware for file uploads and MongoDB for data storage, returning success or error responses based on the outcome.

## 5. Portfolio

Code:

```

import axios from "axios";
import Portfolio from "../model/Portfolio.js";
import SoldTicker from "../model/SoldTicker.js";
import Wallet from "../model/Wallet.js";
import config from "../config.js";
import { pow } from "mathjs";

// Stock Buy -----
export const stockBuyTicker = async (req, res) => {
  let success = false;
  try {
    const { name, symbol, buy_price, no_of_shares } = req.body;

    let portfolio = await Portfolio.findOne({ user_id: req.user.id
});

```

```
let wallet = await Wallet.findOne({ user_id: req.user.id });

if (wallet.balance < buy_price * no_of_shares) {
  return res.status(400).send({
    success,
    message: "Not Enough Funds in the Wallet.",
  });
}

wallet = await Wallet.findOneAndUpdate(
  { user_id: req.user.id },
  {
    $inc: {
      balance: -(buy_price * no_of_shares),
    },
  },
);

let date = new Date();

if (!portfolio) {
  portfolio = await Portfolio.create({
    user_id: req.user.id,
    stocks: [
      {
        name,
        symbol,
        no_of_shares,
        buy_price,
        date_of_buy: date,
      },
    ],
    total_investment: buy_price * no_of_shares,
  });
} else {
  portfolio = await Portfolio.findOneAndUpdate(
    { user_id: req.user.id },
    {
      $push: {
        stocks: {
          name,
          symbol,
          no_of_shares,
          buy_price,
          date_of_buy: date,
        },
      },
      $inc: {
        total_investment: buy_price * no_of_shares,
      },
    },
  );
}
```

```

        },
      },
    );
  }

  success = true;

  return res.status(200).send({
    success,
    message: `Stock bought successfully`,
  });
} catch (err) {
  console.log(err)
  return res.status(500).send({
    success,
    message: "Internal Server Error.",
  });
}
};

```

#### Explanation:

This code defines an async function `stockBuyTicker` that handles stock purchasing by updating user portfolios and wallets. It checks if the user has enough funds and either creates a new portfolio entry or updates an existing one with the purchased stock details. If successful, it responds with a "Stock bought successfully" message; otherwise, it returns an "Internal Server Error" in case of any issues.

## 6. Watchlist

Code:

```

import Watchlist from "../model/Watchlist.js";
import axios from "axios";
import config from "../config.js";

// Add to Watchlist
export const addWatchlist = async (req, res) => {
  let success = false;
  try {
    // stocks, mutual_funds, etfs
    const { name, symbol, type } = req.body;

    let watchlist = await Watchlist.findOne({ user_id: req.user.id });

    if (!watchlist) {
      watchlist = await Watchlist.create({
        user_id: req.user.id,
        [type]: [

```

```

        {
            name,
            symbol,
        },
    ],
});
} else {
    watchlist = await Watchlist.findOneAndUpdate(
        { user_id: req.user.id },
        {
            $push: {
                [type]: {
                    name,
                    symbol,
                },
            },
        },
    );
}
}

```

Explanation:

This code defines an async function `addWatchlist` that allows users to add financial instruments (stocks, mutual funds, or ETFs) to their watchlist. It checks if the user already has a watchlist, and either creates a new one or updates the existing watchlist by adding the provided financial instrument's details (name and symbol). If successful, it sets `success` to `true`; otherwise, it handles any errors that might occur during the process.

## 8. Stocks View Page

Code

```

import React, { useEffect, useState } from "react";
import { FaRegThumbsUp, FaRegThumbsDown } from "react-icons/fa";
import                               StockDetailsTables                               from
"./../components/dashboard/StockDetailsTables";
import { useParams } from "react-router-dom";
import                               StockModal                               from
"./../components/dashboard/modals/StockModal";
import { useDispatch, useSelector } from "react-redux";
import {
    clearHistoricalState,
    clearStockDetails,
    getAllStockDetailsThunk,
    getBalanceSheetThunk,
    getCashFlowThunk,
    getFinancialRatiosThunk,
    getRevenueStmtThunk,
    getStockHistoricalDataThunk,
    getStockInfoThunk,

```

```

    getStockSuggestionThunk,
    getStockDetailsCurrentPriceThunk,
  } from "../../features/stocks/stockDetails";
import Chart from "react-apexcharts";
import { buyStockThunk } from
  "../../features/portfolio/stockTransaction";

const StockDetails = ({ showAlert }) => {
  const { id } = useParams();

  const [isModal, setIsModal] = useState(false);
  const [quantity, setQuantity] = useState(0);

  const [period, setPeriod] = useState("ytd");
  const [interval, setInterval] = useState("1d");

  const {
    isSuccess,
    isError,
    isLoading: detailsLoading,
    priceData,
    cashFlow,
    balanceSheet,
    revenueStmt,
    suggestion,
    historicalData,
    financial,
    info,
  } = useSelector((state) => state.stockDetailsReducer);
  const dispatch = useDispatch();

  useEffect(() => {
    if (id) {
      dispatch(clearStockDetails());
      dispatch(getAllStockDetailsThunk(id));
      dispatch(getStockHistoricalDataThunk({ symbol: id, period,
interval }));
    }
  }, [id]);

  const handleChangeInPeriod = (prd) => {
    setPeriod(prd);
    dispatch(clearHistoricalState());
    dispatch(
      getStockHistoricalDataThunk({ symbol: id, period: prd, interval
    })
    );
  };
};

```



```

const handleChangeInterval = (itl) => {
  dispatch(clearHistoricalState());
  dispatch(
    getStockHistoricalDataThunk({ symbol: id, period, interval: itl
  })
  );
};

const options = {
  chart: {
    type: "candlestick",
    height: 250,
    width: "100%",
  },
  title: {
    text: priceData?.name,
    align: "left",
  },
  xaxis: {
    type: "datetime",
  },
  yaxis: {
    tooltip: {
      enabled: true,
    },
  },
};

```

Explanation:

The code defines a React component, "StockDetails," used to display detailed stock information.

It utilizes various hooks for state management, fetches stock data using Redux, and includes an ApexCharts candlestick chart for visualizing historical stock prices.

The component offers options to change the time period for historical data and allows users to buy stocks through a modal.

## 9. Mutual Fund and ETF Page

Code:

```

import React, { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useParams } from "react-router-dom";
import {
  clearMFDetails,
  clearMFHistory,
  getMFCurentPriceThunk,
  getMFDetailsThunk,

```

```

    getMFHistoryThunk,
  } from "../../features/mutualfunds/mfDetails";
import                                MutualFundModal                                from
  "../../components/dashboard/modals/MutualFundModal";
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Filler,
  Legend,
} from "chart.js";
import { Line } from "react-chartjs-2";
import { buyMFThunk } from "../../features/portfolio/mfTransaction";
const MutualFundDetails = ({ setAlert }) => {
  const { id } = useParams();

  const [isModal, setIsModal] = useState(false);
  const [price, setPrice] = useState(0);
  const [years, setYears] = useState(0);
  const [type, setType] = useState("");
  const [investment, setInvestment] = useState("");

  const [start, setStart] = useState("");
  const [end, setEnd] = useState("");
  const [period, setPeriod] = useState("ytd");
  const [interval, setInterval] = useState("1d");

  const { isSuccess, isLoading, isError, priceData, detailsData,
  historyData } =
    useSelector((state) => state.mfDetailsReducer);

  const dispatch = useDispatch();

  useEffect(() => {
    if (id) {
      dispatch(clearMFDetails());
      dispatch(getMFCurentPriceThunk(id));
      dispatch(getMFDetailsThunk(id));
      let year = new Date().getFullYear();
      dispatch(
        getMFHistoryThunk({
          symbol: id,
          period,
          interval,
        })
      )
    }
  })
}

```

```

    );
  }
}, [id]));

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Filler,
  Legend
);

const options = {
  responsive: true,
  plugins: {
    legend: {
      position: "top",
    },
    title: {
      display: true,
      text: priceData?.name,
    },
  },
};

```

Explanation:

The code defines a React component, "MutualFundDetails," used to display detailed stock information.

It utilizes various hooks for state management, fetches data using Redux, and includes an ApexCharts candlestick chart for visualizing historical stock prices.

The component offers options to change the time period for historical data and allows users to buy through a modal.

## 10. Deposits and Withdraws

Code:

```

import mongoose from "mongoose";
import BankBalance from "../model/BankBalance.js";
import Deposit from "../model/Deposit.js";
import User from "../model/User.js";
import Wallet from "../model/Wallet.js";
import randomHash from "../utils/randomHash.js";

// Add Deposit

```

```
export const addDeposit = async (req, res) => {
  let success = false;

  try {
    const { amount } = req.body;

    let transaction_id = randomHash(14);

    let bankBalance = await BankBalance.findOne({
      user_id: req.user.id,
    });

    let user = await User.findOne({ _id: req.user.id });

    if (bankBalance.balance < amount) {
      return res.status(400).send({
        success,
        message: "Some Error Occured.",
      });
    }

    let deposit = await Deposit.create({
      amount,
      bank_account_number: user?.bank?.accountNumber,
      transaction_id,
      user_id: req.user.id,
    });
  }
}
```

Explanation:

This code defines an "addDeposit" function to handle depositing money into a user's bank account.

It generates a unique transaction ID, checks if the user has sufficient balance in their bank account, and records the deposit in the database. If the balance is insufficient, it returns an error response.

## 11. Dashboard:

Code: Contains multiple components Stock Details,Portfolio,Watchlist,Account Info ,Ticker Search Top loser Top Gainer etc

## 12. Python API (Fast API) to fetch prices symbols

Code:

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from utils.stocks import *
from utils.mutualfunds import *
from utils.etf import *
```

```
import uvicorn
import aiohttp as aiohttp
# import os
# from dotenv import load_dotenv

app = FastAPI()

# Allow requests from both the React app and the backend
origins = ["http://localhost:3000", "http://localhost:8000", "http://localhost:3001"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Get the path to the directory this file is in
# BASEDIR = os.path.abspath(os.path.dirname(__file__))

# # Connect the path with your '.env' file name
# load_dotenv(os.path.join(BASEDIR, 'config.env'))

# test_var = os.getenv("TEST_VAR")

# print(test_var)

@app.get("/")
def read_root():
    return {"message": "Stock Market API"}

# ----- STOCKS -----
# -----
# Get Nifty50 and Sensex
@app.get("/stock/index")
def read_stock_index():
    data = handle_index()
    return data
```

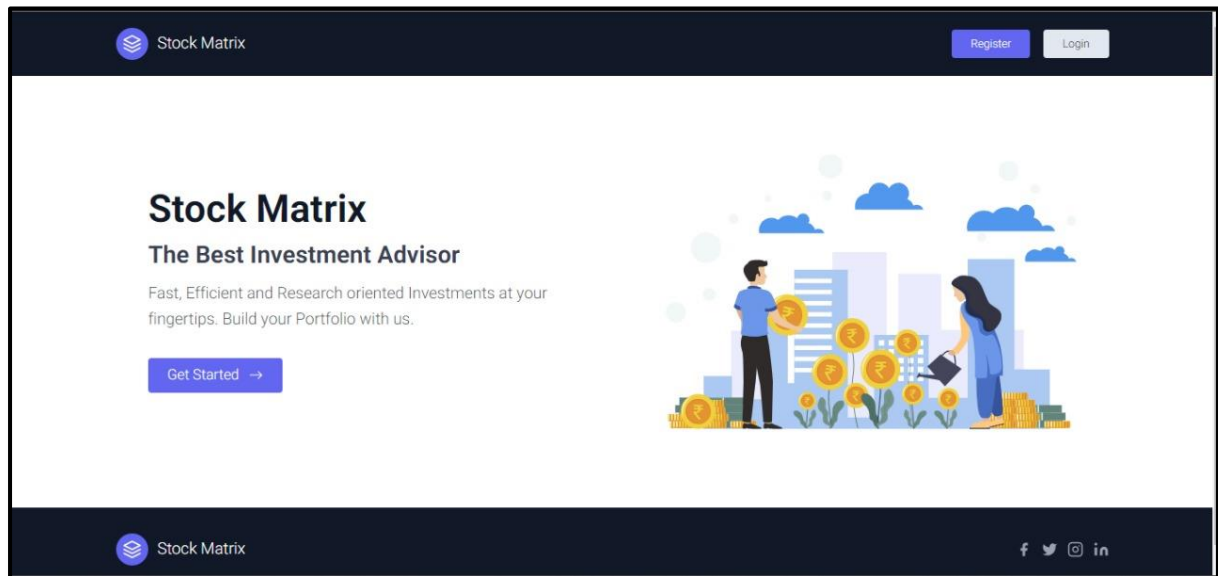
Explanation:

This code configures a FastAPI application, sets up CORS (Cross-Origin Resource Sharing) to allow requests from specified origins, and defines a route to retrieve data related to stock market indices like Nifty50 and Sensex.

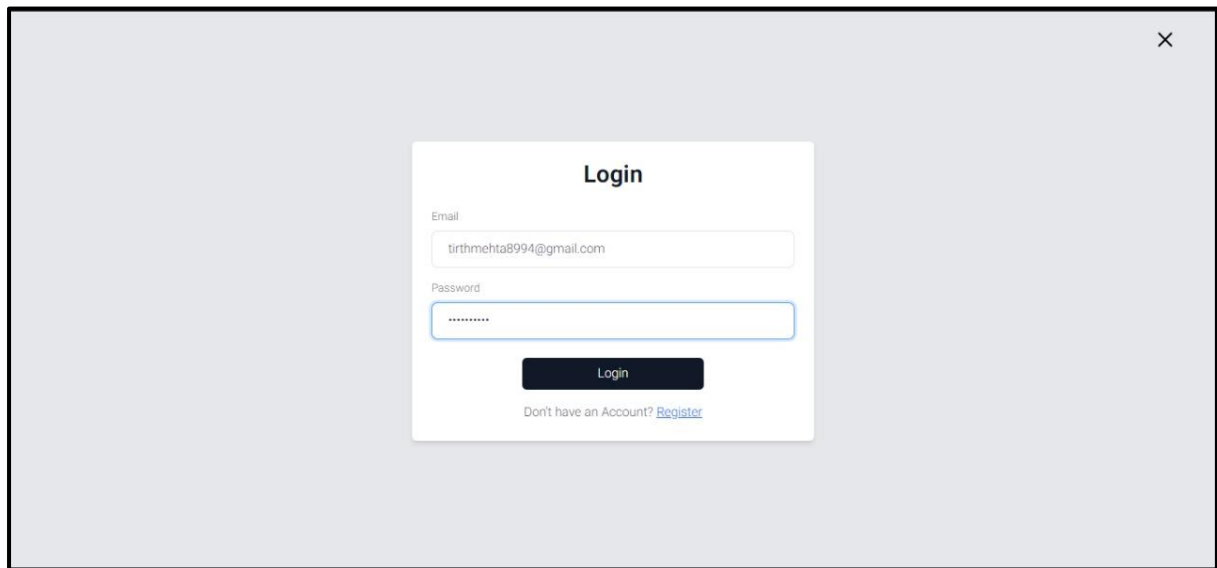
## 4. Output

### Screenshots:

#### 1. Registration Page

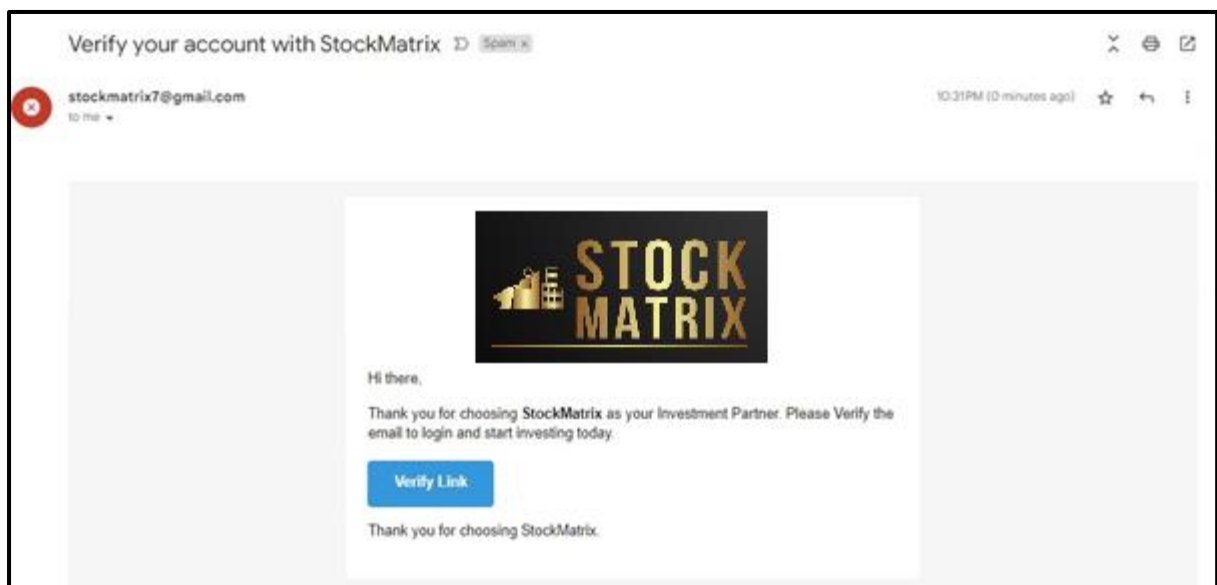
The screenshot shows a registration form titled 'Register' in a white box with a close button (X) in the top right corner. The form contains the following fields: 'Full Name' with the value 'Dev Vora', 'Email' with the value 'devavora5@gmail.com', 'Phone No.' with the value '9567890345', 'Password' with masked characters '\*\*\*\*\*', and 'Confirm Password' with masked characters '\*\*\*\*\*'. Below the 'Confirm Password' field, there is a green checkmark and the text 'Confirm Password.'. At the bottom of the form is a dark blue button labeled 'Register'. Below the button, there is a link that says 'Already have an Account? [Login](#)'.

## 2. Login Page



A screenshot of a web browser displaying a login form. The form is titled "Login" and is centered on a light gray background. It contains two input fields: "Email" with the value "tirthmehta8994@gmail.com" and "Password" with masked characters "\*\*\*\*\*". Below the password field is a dark blue "Login" button. At the bottom of the form, there is a link that says "Don't have an Account? [Register](#)". A close button (X) is visible in the top right corner of the browser window.

## 3. Email Verification



## 4. KYC Page

Great! Please do your KYC.

KYC

Proof of Identity

Choose File No file chosen

File Size upto: 10MB

Proof of Address

Choose File No file chosen

File Size upto: 10MB

Add Documents

## 5. Portfolio

Stock Matrix

Dashboard

Portfolio

Stocks

Mutual Funds

ETFs

Deposits

Withdraws

Watchlist

Tirth Mehta

Search Tickers

Logout

Your Portfolio

Total Investment

₹ 0

Total Profit

₹ 0

Your Stocks

Name	Symbol	Profit Earned	Total Investment	Buy Price	Date
------	--------	---------------	------------------	-----------	------

Your Mutual Funds

Your Lumpsum Investments

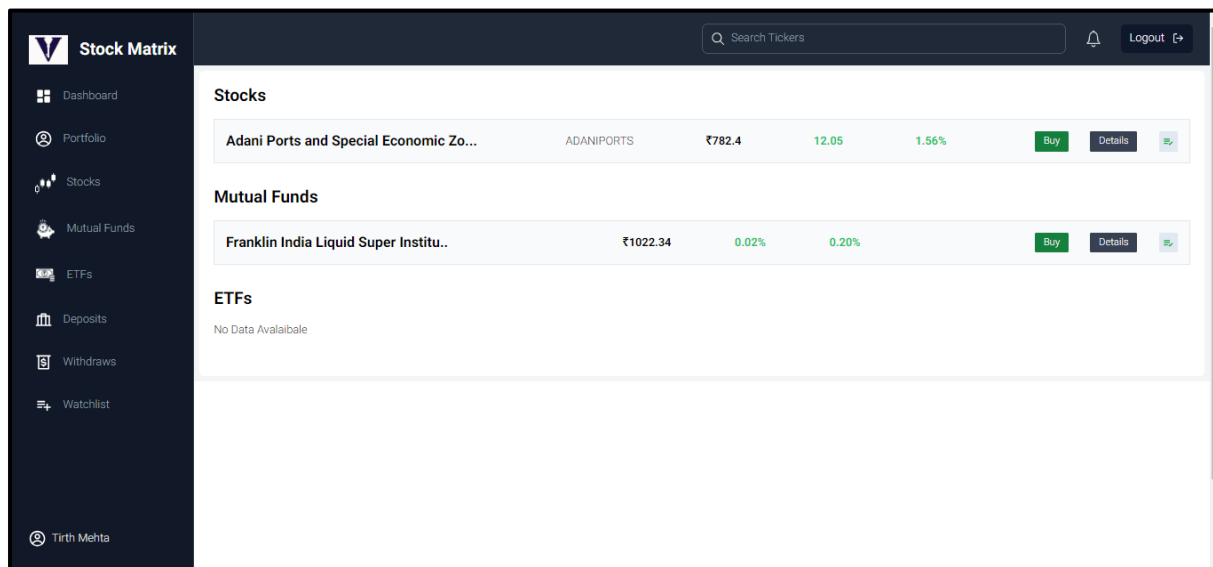
Name	Buy Price (NAV)	Exp. Amount Earned	Total Investment	Exp. Interest Earned	Total years	Date of Buy
------	-----------------	--------------------	------------------	----------------------	-------------	-------------

Your SIP Investments

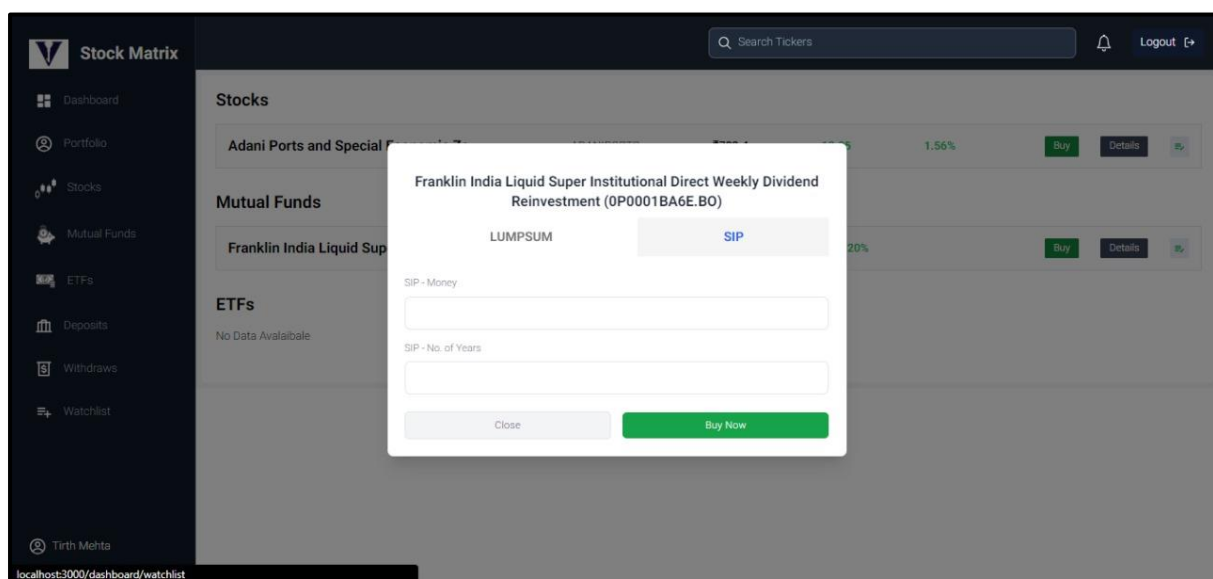
Name	Buy Price (NAV)	Exp. Amount Earned	Exp. Interest Earned	Total years	Date of Buy
------	-----------------	--------------------	----------------------	-------------	-------------

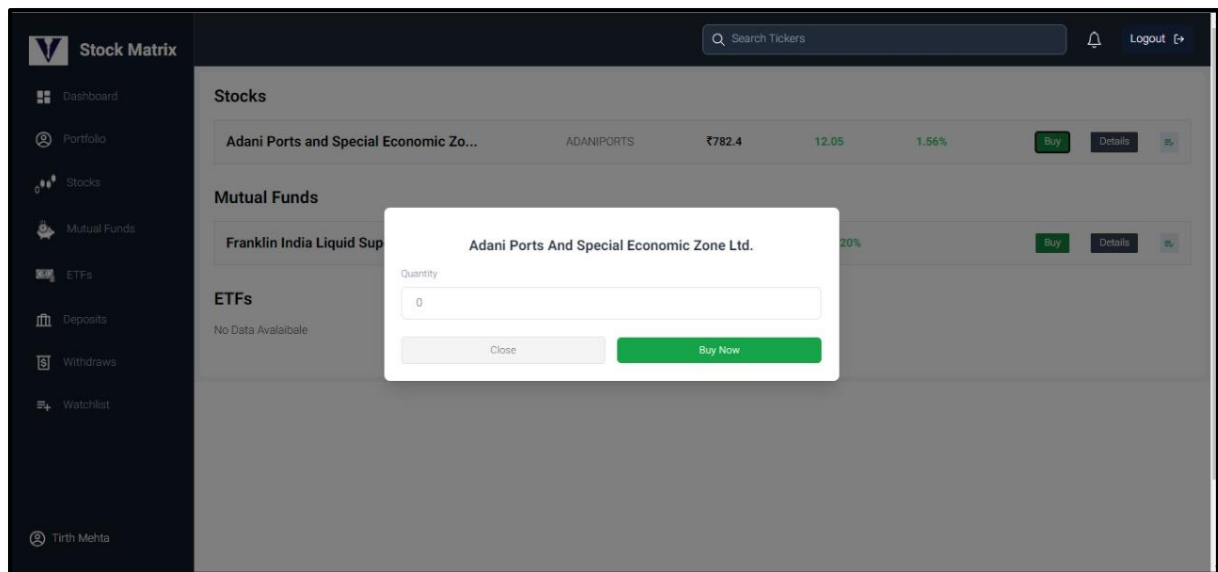


## 6. Watchlist

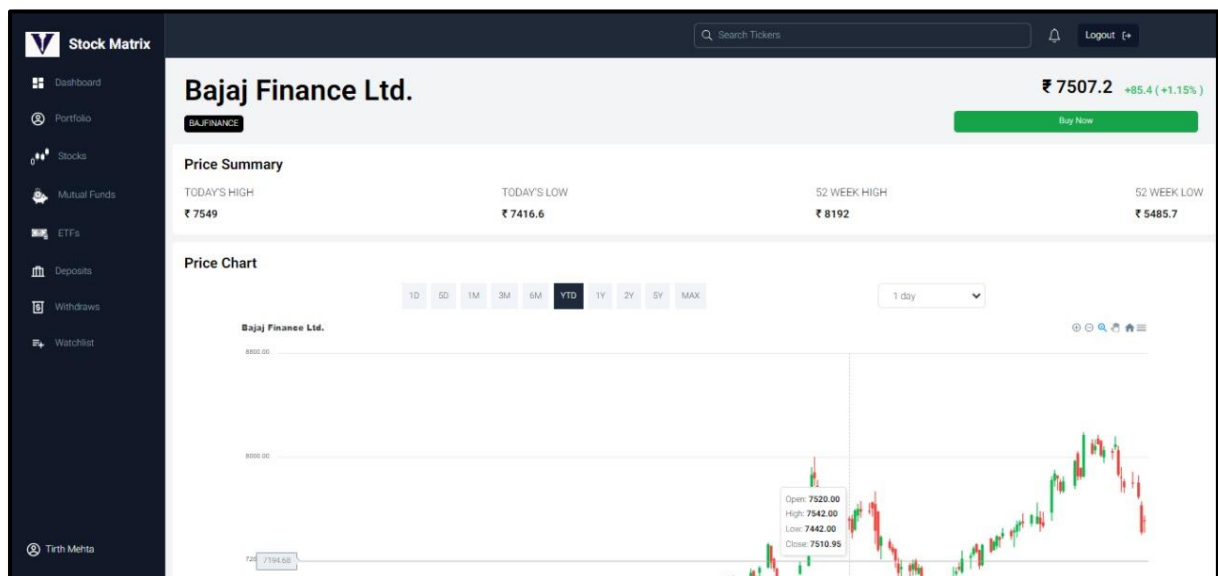


## 7. Buy and Sell





## 8. Stocks View Page



**Stock Matrix**

- Dashboard
- Portfolio
- Stocks**
- Mutual Funds
- ETFs
- Deposits
- Withdraws
- Watchlist

Tirth Mehta

### Stocks

**Nifty 50 (NIFTY50)**  
**₹ 19047.25**  
+190.000 (1.010%)

**S&P Bse Sensex (SENSEX)**  
**₹ 63782.80**  
+634.650 (1.010%)

Adani Enterprises Ltd.	ADANIENT	₹2261.7	58.5	2.66%	Buy	Details	₹
Adani Ports and Special Economic Zo...	ADANIPORTS	₹782.4	12.05	1.56%	Buy	Details	₹
Apollo Hospitals Enterprise Ltd.	APOLLOHOSP	₹4830.2	69.9	1.47%	Buy	Details	₹
Asian Paints Ltd.	ASIANPAINT	₹2955.15	-5.15	-0.17%	Buy	Details	₹
Axis Bank Ltd.	AXISBANK	₹1002.75	30.7	3.16%	Buy	Details	₹
Bajaj Auto Ltd.	BAJAJ-AUTO	₹5373.55	114.2	2.17%	Buy	Details	₹
Bajaj Finance Ltd.	BAJFINANCE	₹7507.2	85.4	1.15%	Buy	Details	₹

### Company Essentials

Market Cap	Enterprise Value	No. of Shares	P/E
₹ 257834.05 Cr.	₹ 259414.44 Cr.	114 Cr.	145.91
P/B	Face Value	Div. Yield	Book Value (TTM)
17.76	₹ 1	0.05 %	₹ 127.38
CASH	DEBT	Promoter Holding	EPS (TTM)
₹ 1339.86 Cr.	₹ 2920.25 Cr.	72.61 %	₹ 15.5
Sales Growth	ROE	ROCE	Profit Growth
150.99%	17.39 %	21.88%	125.16 %

### Strengths

- The company has shown a good profit growth of 32.4176306907109% for the Past 3 years.
- The company has shown a good revenue growth of 60.7459835435925% for the Past 3 years.
- The company has significantly decreased its debt by 1811.58 Cr.
- The company has an efficient Cash Conversion Cycle of -44.9608896390093 days.
- The company has a good cash flow management; CFO/PAT stands at 4.45286876309842.
- The company has a high promoter holding of 72.61%.

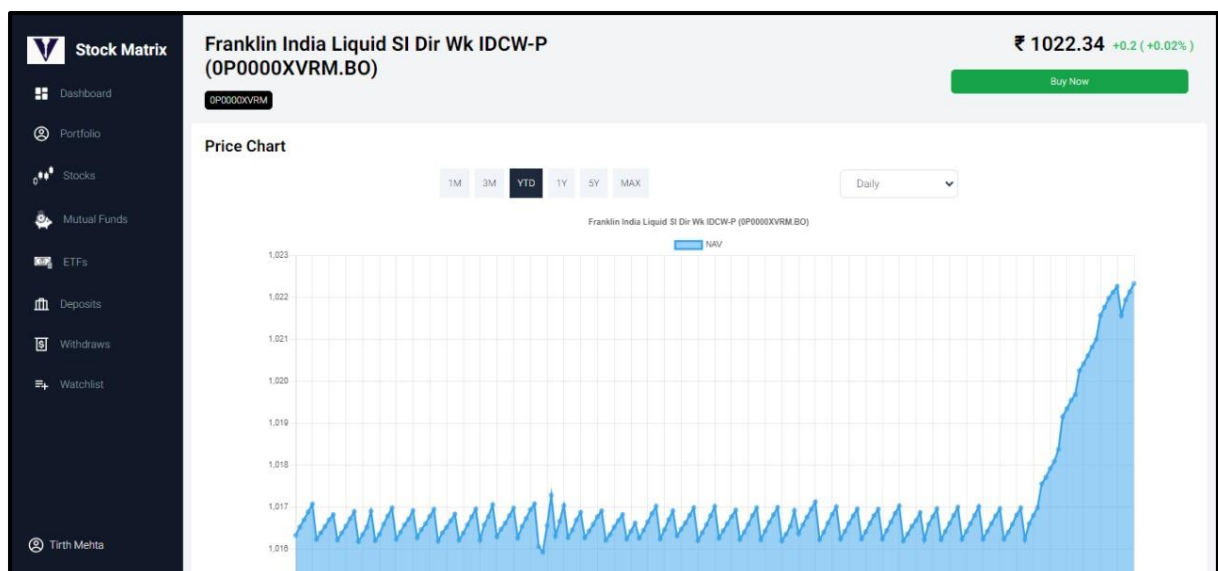
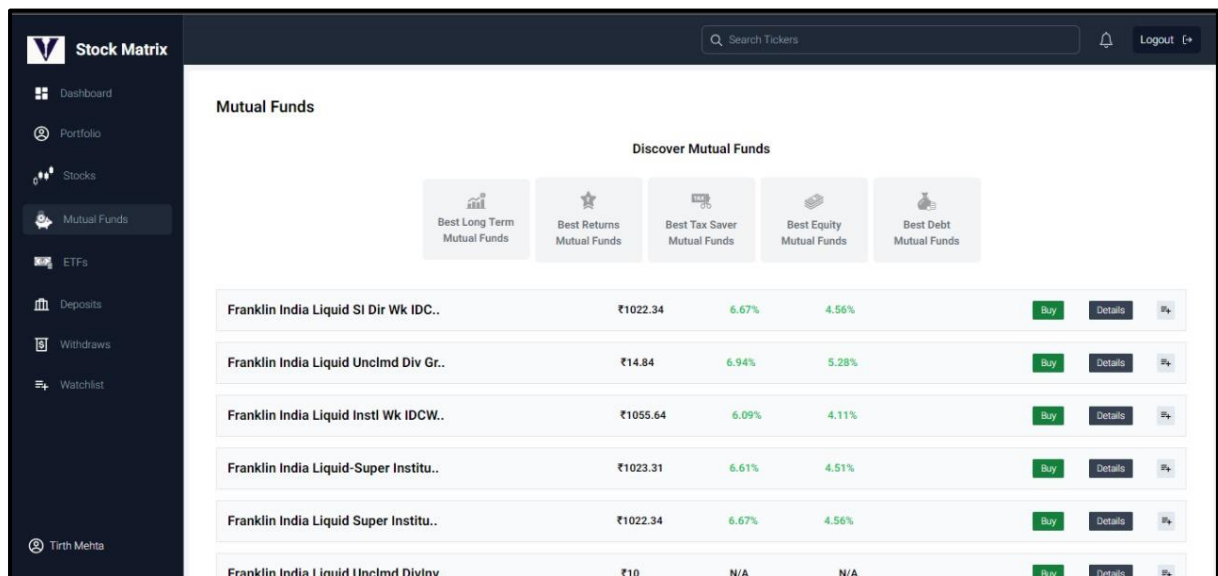
### Financial Ratios

Sales Growth	Profit Growth	ROE%	ROCE %	Debt/Equity
1 year: 150.99%	1 year: 125.16%	1 year: 17.39%	1 year: 21.88%	0.21
3 year: 60.75%	3 year: 32.42%	3 year: 14.34%	3 year: 19.23%	
5 year: 47.82%	5 year: 52.46%	5 year: 15.31%	5 year: 18.63%	
Price to Cash Flow	Interest Cover Ratio	CFO/PAT (5 Yr. Avg.)		
23.05	4.33	4.45		

### Limitations

- The company is trading at a high PE of 145.91.
- The company is trading at a high EV/EBITDA of 80.0758.

## 9. Mutual Fund and ETF Page



Stock Matrix		Portfolio Composition		Equity Holdings	
<div>Dashboard</div> <div>Portfolio</div> <div>Stocks</div> <div>Mutual Funds</div> <div>ETFs</div> <div>Deposits</div> <div>Withdraws</div> <div>Watchlist</div> <div>Tirth Mehta</div>		Cash	100.00%	Price/Earnings	0
		Stocks	0.00%	Price/Book	0
		Bonds	0.00%	Price/Sales	0
		Others	0.00%	Price/Cashflow	0
		Preferred	0.00%	Median Market Cap	0
		Convertible	0.00%	3 Year Earnings Growth	0
		Sector Weightings		Performance	
		Basic Materials	N/A	YTD	5.58%
		Consumer Cyclical	N/A	One Month	0.43%
		Financial Services	N/A	Three Month	1.58%
		Real Estate	N/A	One Year	6.67%
		Consumer Defensive	N/A	Three Year	4.58%
		Healthcare	N/A	Five Year	4.56%
		Utilities	N/A		
		Communication Services	N/A		
		Energy	N/A		
		Industrials	N/A		

## 10. Deposits and Withdraws

Stock Matrix		Deposits	
<div>Dashboard</div> <div>Portfolio</div> <div>Stocks</div> <div>Mutual Funds</div> <div>ETFs</div> <div>Deposits</div> <div>Withdraws</div> <div>Watchlist</div> <div>Tirth Mehta</div>		<div> <div>Wallet Balance</div> <div>₹ 100000</div> </div> <div> <div>Amount</div> <div>Your Amount</div> <div>You will be provided with a fake Bank Balance containing the balance of ₹ 1,00,000</div> <div>Deposit to Wallet</div> </div>	
		Transaction Id	
		Amount	
		Bank Account Number	
		Date	
		#02ea3a2c724a	
		100000	
		123456789	
		10/28/2023, 6:44:26 PM	
		#0da84f1444f1	
		100000	
		123456789	
		10/28/2023, 8:37:24 PM	

**Stock Matrix**

Logout

Dashboard
Portfolio
Stocks
Mutual Funds
ETFs
Deposits
**Withdraws**
Watchlist

**Withdraws**

Wallet Balance  
₹ 100000

Amount  

  
Withdraw from Wallet

Transaction Id	Amount	Date
#a3633636495a	100000	10/28/2023, 6:44:47 PM

Tirth Mehta

## 11. Dashboard:

**Stock Matrix**

Logout

Dashboard
Portfolio
Stocks
Mutual Funds
ETFs
Deposits
Withdraws
Watchlist

**Dashboard**

Wallet Balance  
₹ 100000

Total Investment  
₹ 0

Total Profit  
₹ 0

Chart.js Line Chart - Multi Axis  
Deposit Withdraw

**Stock Indexes**

Nifty 50 (NIFTY50)  
₹ 19047.25 +190.00 (1.01%)

S&P Bse Sensex (SENSEX)  
₹ 63782.80 +634.65 (1.01%)

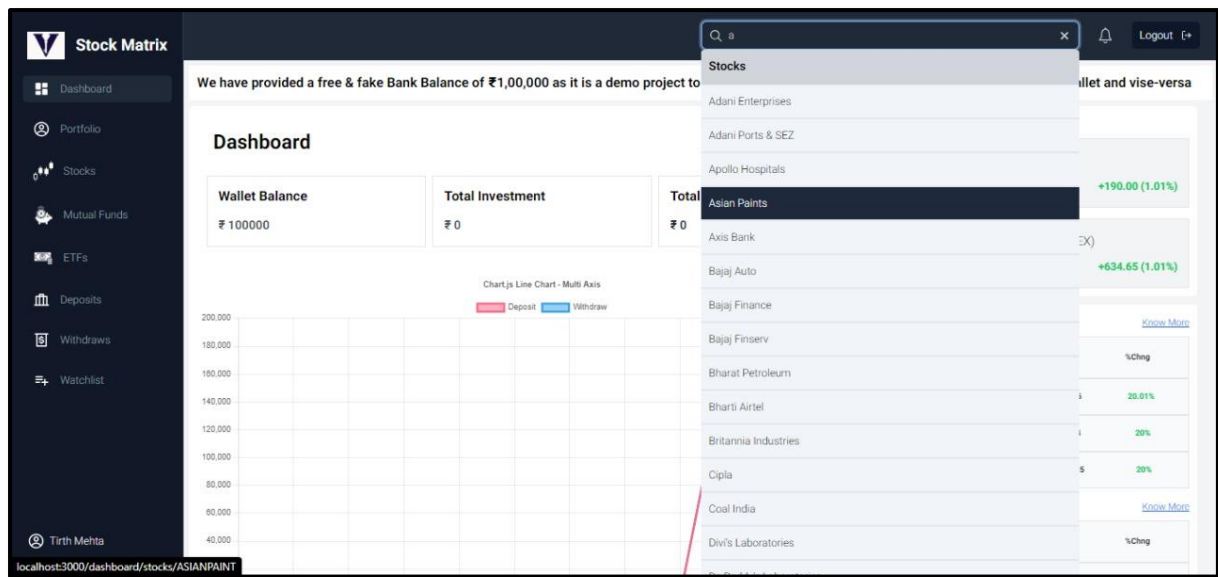
**Top Gainers**
[Know More](#)

Symbol	LTP	%Chng
Talbro's Auto C...	₹251.45	20.01%
Riddhi Steel &...	₹51.84	20%
Kody Technolab	₹280.25	20%

**Top Losers**
[Know More](#)

Symbol	LTP	%Chng
Kesar Terminal...	₹35.45	-9.91%

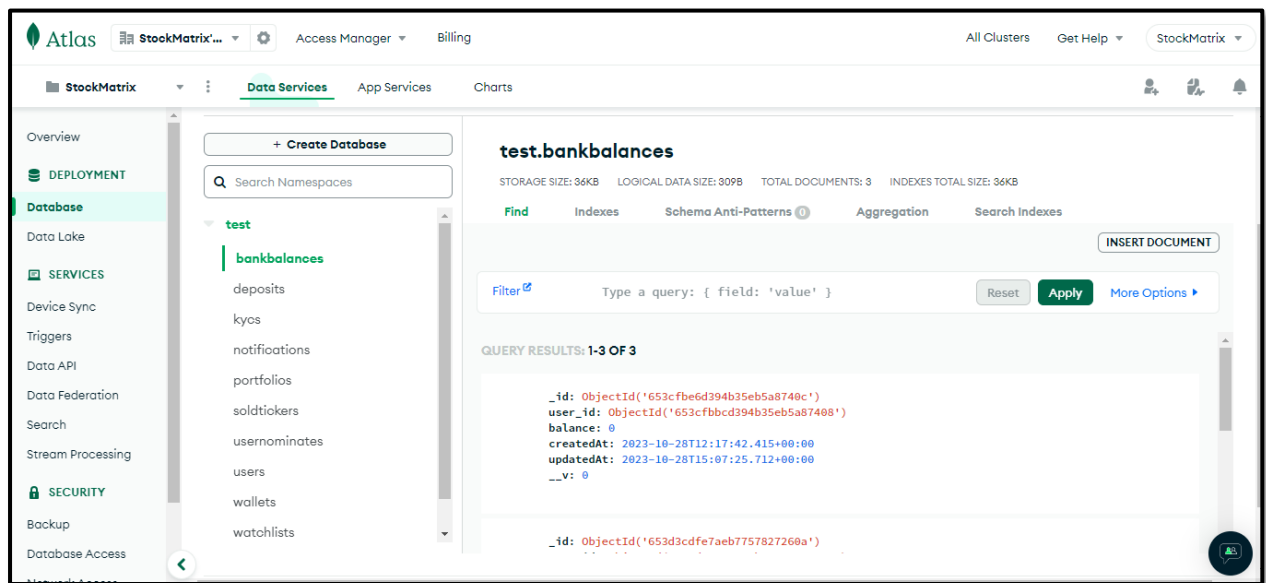
Tirth Mehta



## 12. Python API (Fast API) to fetch prices symbols

```
INFO: 127.0.0.1:54408 - "GET /stock/details/all/STLTECH HTTP/1.1" 200 OK
INFO: 127.0.0.1:54504 - "GET /stock/index HTTP/1.1" 200 OK
INFO: 127.0.0.1:54504 - "GET /stock/index HTTP/1.1" 200 OK
INFO: 127.0.0.1:54505 - "GET /top/stocks?skip=0&limit=50 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54505 - "GET /top/stocks?skip=0&limit=50 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54506 - "GET /all/etfs?skip=0&limit=10 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54506 - "GET /all/etfs?skip=0&limit=10 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54561 - "GET /top/stocks?skip=0&limit=50 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54560 - "GET /stock/index HTTP/1.1" 200 OK
INFO: 127.0.0.1:54506 - "GET /top/stocks?skip=0&limit=50 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54527 - "GET /mutualfund/all?skip=0&limit=10 HTTP/1.1" 200 OK
INFO: 127.0.0.1:54560 - "GET /stock/index HTTP/1.1" 200 OK
SCRIP-269887.NS: No data found, symbol may be delisted
INFO: 127.0.0.1:54506 - "GET /stock/historical/data/SCRIP-269887?period=ytd&interval=1d HTTP/1.1" 200 OK
SCRIP-269887.NS: No data found, symbol may be delisted
INFO: 127.0.0.1:54506 - "GET /stock/historical/data/SCRIP-269887?period=ytd&interval=1d HTTP/1.1" 200 OK
INFO: 127.0.0.1:54527 - "GET /mutualfund/all?skip=0&limit=10 HTTP/1.1" 200 OK
SCRIP-269887.NS: No data found, symbol may be delisted
INFO: 127.0.0.1:54527 - "GET /stock/historical/data/SCRIP-269887?period=2y&interval=1d HTTP/1.1" 200 OK
SCRIP-269887.NS: No data found, symbol may be delisted
INFO: 127.0.0.1:54527 - "GET /stock/historical/data/SCRIP-269887?period=1mo&interval=1d HTTP/1.1" 200 OK
INFO: 127.0.0.1:54596 - "GET /stock/index HTTP/1.1" 200 OK
INFO: 127.0.0.1:54596 - "GET /stock/index HTTP/1.1" 200 OK
INFO: 127.0.0.1:54597 - "GET /top/stocks?skip=0&limit=50 HTTP/1.1" 200 OK
```

### 13. Mongo DB Collections:





## 5. Conclusion

The Stock Matrix - Stock Market Simulation Website is a dynamic platform designed to revolutionize how individuals interact with the world of stock trading. Through the seamless integration of cutting-edge technologies, including React for the frontend and MongoDB for the database, the project offers users an immersive, risk-free trading experience. The incorporation of a risk management tool, watchlist functionality, and portfolio management capabilities empowers users with valuable insights and tools to make informed investment decisions.

The future integration of Alphavantage's premium API promises to elevate the platform's data accessibility, providing users with a comprehensive understanding of market trends and dynamics. The platform's commitment to user education, evidenced by the inclusion of an extensive library of articles, tutorials, and videos, sets it apart as a valuable learning resource for both novice and experienced traders.

With a user-friendly interface, responsive design, and robust backend architecture, Stock Matrix aims to bridge the gap between theoretical knowledge and practical application in the realm of stock trading. The ongoing development plans, including advanced charting tools and machine learning-driven insights, underscore the project's commitment to continuous enhancement.

In summary, Stock Matrix is poised to become a pivotal tool in the financial literacy landscape, offering users a risk-free environment to develop and refine their trading skills. By fostering a community of informed and empowered traders, this platform is positioned to shape the future of stock market education and investment strategies.