TUMKUR UNIVERSITY

A

PROJECT REPORT

ON

"STOCK SENSE - STOCK PREDICTION WEBISTE"

Submited in the partial fulfillment of the requirements for the degree of

BACHELOR OF COMPUTER APPLICATION

Submitted by

Jeevan O N	**U11VI22S0026**
Chandangowda R	**U11VI22S0027**
H R Manoj Gowda	**U11VI22S0028**
Vishwas S R	**U11VI22S0029**
Manikanta K N	**U11VI22S0030**

DEPARTMENT OF BACHELOR OF COMPUTER APPLICATION

VIDYA FIRST GRADE COLLEGE

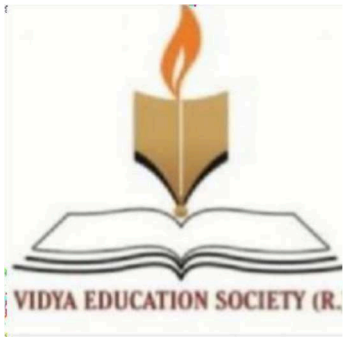
Near Akka-Thangi Park , Devarayapattana New Extension

Tumkur - 572106

``

VIDYA FIRST GRADE COLLEGE

Near Akka-Thangi Park, Devarayanapattana New Extension,
Tumkur-572106



DEPARTMENT OF BACHELOR OF COMPUTER APPLICATIONS

CERTIFICATE

This is to certify that the project work entitled "STOCKSENSE - STOCK-PREDICTION SITE" has been carried out in partial fulfillment of the requirements for the award of the degree of Bachelor of Computer Applications at VFGC, Tumkur, during the academic year 2024–2025. The project has been approved as it meets the academic requirements prescribed for the project work of the Bachelor of Computer Applications, and has been carried out by:

Jeevan O N	**U11VI22S0026**
Chandangowda R	**U11VI22S0027**
H R Manoj Gowda	**U11VI22S0028**
Vishwas S R	**U11VI22S0029**
Manikanta K N	**U11VI22S0030**

Signature of the Guide	Signature of Principal	Signature of Academic Director
Ms tejaswini N Assistant Professor Dept.of BCA, VFGC	Mrs. Shilpa N Principal, VFGC, Tumkuru	Mr. Rajesha K S Academic Director PVFGC, Tumkuru

Name of Examiners

Signature of Examiners

``

ACKNOWLEDGEMENT

First of all, I would like to extend my heartfelt thanks to VFGC, for her constant encouragement and unwavering support completion of this project.

The successful completion of any undertaking is truly complete only when we remember and express our gratitude to the Almighty, our parents, teachers, and all those who directly or indirectly supported and guided us throughout the execution of this work.

I wish to express my deep sense of gratitude and sincere thanks to my internal guide, of Ms. Tejaswini N, Assistant Professor, Department of Bachelor of Computer Applications, VFGC, Tumkuru, for her invaluable guidance and support at every phase of
my project.

I wish to express my sincere thanks to Mr.Rajesha KS, Academic Director ,
VFGC Tumkuru. for his excellent support and guidanceduring my time at the institute.

I am deeply thankful to my parents, classmates, friends, and all the faculty members of the Department of Bachelor of Computer Applications for their continuous encouragement and support throughout the course of this project. Finally,

I would like to express my sincere thanks to all those who directly or indirectly contributed to the successful completion of my project.

Jeevan O N	**U11VI22S0026**
Chandangowda R	**U11VI22S0027**
H R Manoj Gowda	**U11VI22S0028**
Vishwas S R	**U11VI22S0029**
Manikanta K N	**U11VI22S0030**

``

DECLARATION

We hereby declare that the project entitled STOCKSENSE - STOCK-PREDICTION SITE " has been carried out by us under the supervision of Ms. Tejaswini N, Assistant Professor, and submitted in partial fulfillment of the requirements for the Project Work of the VI Semester for the degree of Bachelor of Computer Applications, under Tumkur University, for the academic year 2024–2025.

This report has not been submitted to any other organization or university for the award of any degree or certificate.

(Signatures)

Name1:

Name2:

Name3:

Name4:

ABSTRACT

Stock Sense is an intelligent application aimed at modernizing and simplifying the process of stock market analysis and prediction for investors and financial enthusiasts. Traditional approaches to stock prediction, which rely heavily on manual chart analysis and historical pattern recognition, are often time-consuming, subjective, and prone to human bias. By leveraging machine learning algorithms and data visualization, Stock Sense enhances forecasting accuracy, reduces manual effort, and supports data-driven decision-making.

The system integrates multiple modules that allow users to fetch real-time market data, visualize historical trends, and generate predictive insights through a user-friendly interface. Features like interactive charts, automated trend forecasting, and notification alerts keep users informed about significant market movements. The application combines historical price data, technical indicators, and advanced regression or deep learning models to identify potential future price directions.

The Prediction Module of Stock Sense is designed to automate one of the most critical and challenging aspects of trading—forecasting future price movements. Traditional manual analysis often lacks the capability to process large datasets efficiently and fails to adapt to rapidly changing market conditions. By automating predictions, Stock Sense offers greater consistency, speed, and insight into potential investment opportunities.

Overall, Stock Sense promotes a smarter, data-driven investment environment, enabling users to spend less time on manual analysis and more time making strategic decisions. Through the use of modern technologies such as machine learning, real-time data APIs, and an intuitive interface, Stock Sense empowers investors with actionable insights, fostering confidence and efficiency in navigating the dynamic world of stock trading.

SYNOPSIS

ABOUT THE PROJECT:

The Stock Sense application is designed to help users predict future stock prices by analyzing historical stock data using machine learning techniques, all within a modern web-based interface. The system replaces manual and time-consuming stock analysis with an automated, data-driven solution that increases prediction accuracy and supports smarter investment decisions.
Key features include real-time data collection, dynamic chart visualization, predictive analytics powered by backend models, and interactive dashboards built using EJS templates. The system uses Node.js for server-side logic, MongoDB for flexible and scalable data storage, and a clean EJS frontend to display predictions and historical trends to users in an intuitive way. By automating complex forecasting tasks, Stock Sense reduces human error, speeds up analysis, and brings advanced stock prediction tools to everyday users.

SOFTWARE REQUIREMENTS:

- OPERATING SYSTEM: Windows 10 or above
- FRONTEND: EJS, HTML, CSS, JavaScript
- SERVER SIDE: Node.js with Express.js
- DATABASE: MongoDB (for storing historical stock data, predictions, and user data)
- LIBRARIES / TOOLS: Mongoose (for MongoDB interaction), Chart.js / D3.js (for visualization), Axios / Node-fetch (for API data fetching)

HARDWARE REQUIREMENTS:

- PROCESSOR: Intel i3 processor or above
- RAM: Minimum 4 GB
- HARD DISK: 500 GB or more

CONTENTS

Chapter 1 : Introducttion

Stock Sense is a smart and user-friendly stock market analysis platform designed to simplify the process of tracking, analyzing, and investing in stocks. In today's fast-moving financial world, many investors—especially beginners—face difficulties in understanding stock trends, interpreting market news, and making the right investment decisions at the right time. Stock Sense bridges this gap by providing a one-stop solution for real-time stock data, market analysis, personalized recommendations, and alerts.

Overview

Stock Sense is an intelligent stock market analysis platform that helps users track live stock prices, get personalized recommendations, and receive real-time alerts. It is designed for both beginners and experienced investors, making stock market data simple, fast, and easy to use. With a clean interface and integrated news, Stock Sense makes investing smarter and more efficient.

Objective

The main objective of Stock Sense is to provide a simple, efficient, and intelligent platform for users to track, analyze, and manage stock market information in real-time. It aims to help users make informed investment decisions by offering:

1. Real-time Stock Market Data
2. Personalized Stock Recommendations
3. Instant Alerts & Notifications
4. Integrated Financial News
5. User-Friendly Interface for All Users

Chapter2 :Literature Survey

In the current scenario, most stock market platforms are designed with professional traders in mind, making them complicated and difficult for regular users. They often lack features that help beginners or casual investors make quick, informed decisions. Below is a comparison of the existing system and the proposed system (Stock Sense):

2.1Existing System

- Complex and hard to use for beginners.
- No real-time notifications or alerts.
- No personalized stock recommendations.
- Data spread across multiple platforms.
- Cluttered and confusing user interfaces.

2.2Proposed System (Stock Sense)

- Simple, user-friendly interface for all users.
- Real-time stock updates with live market data.
- Personalized stock recommendations for each user.
- Instant alerts for stock price changes and news.
- All-in-one platform for analysis, news, and updates.

Chapter3 :Requirement Specification

3.1 Hardware Requirements

- Processor: Intel i3 or higher
- RAM: 4 GB minimum (8 GB recommended)
- Storage: 100 MB (for local development)
- Display: Standard HD Display

3.2 Software Requirements

- Operating System: Windows / Linux / macOS
- Frontend: HTML, CSS, JavaScript (or EJS if using Node.js)
- Backend: Node.js with Express
- Database: MongoDB
- APIs: Financial APIs (e.g., Yahoo Finance API, Financial Modeling API)
- Others: Browser (Chrome/Edge), Code Editor (VS Code)

3.3 Functional Requirements

- User Registration and Login
- Real-time Stock Data Fetching
- Display of Stock Charts and Trends
- Notifications and Alerts System
- News Integration Related to Stocks
- Search Functionality for Stocks

3.4 Non-Functional Requirements

- Fast Loading Time
- User-Friendly Interface
- Data Privacy and Security
- Responsive Design (Mobile-Friendly)

Chapter4 :System Analysis

Study of the System

4.1 Graphical User Interface (GUI's)

For better user flexibility and interaction, the interface has been designed with a graphics-based concept supported by a browser-based interface. The GUI is broadly divided into two categories:

4.2 Administrative User Interface

- This interface is focused on managing core organizational data.
- It requires authentication for secure access.
- Functions supported include:
  - Data Insertion
  - Data Deletion
  - Data Updation
  - Advanced Search for information

4.3 Operational / Generic User Interface

- Designed for general users who use the system for their daily operations.
- Helps users to:
  - View and access real-time stock data
  - Get personalized recommendations and alerts
  - Manage their profiles and preferences in a customized way

Chapter5 :Feasibility Report

The preliminary investigation examines the feasibility of the Stock Sense project and its potential usefulness. The primary objective of this feasibility study is to evaluate the Technical, Operational, and Economic aspects of developing Stock Sense to ensure its success and effectiveness.

Even though any project is possible with unlimited resources and time, this feasibility study focuses on practical, real-world limitations.

Types of Feasibility Considered:

1. Technical Feasibility
2. Operational Feasibility
3. Economic Feasibility

5.1 Technical Feasibility

The technical feasibility examines whether the required technology and resources are available to successfully build Stock Sense.

- Does the necessary technology exist? → Yes, modern web technologies like Node.js, MongoDB, and APIs are available.
- Is the system scalable? → Yes, the system can be upgraded easily in the future.
- Is the response time fast? → Yes, with proper API integration, real-time stock updates are ensured.
- Is data secure and reliable? → Yes, security standards are followed, and sensitive data is protected.

Conclusion:

Stock Sense is technically feasible. The system utilizes web-based interfaces and reliable open-source tools, making it cost-effective and easily maintainable.

5.2 Operational Feasibility

Operational feasibility focuses on whether the system will work effectively once implemented and if it fulfills the organization's or users' needs.

- User Acceptance: The system is designed to be simple, ensuring that even non-technical users can operate it.
- Management Support: It is built keeping both administrators and regular users in mind.
- User Resistance: As the system solves major pain points (complexity of existing platforms), user acceptance is expected to be high.

Conclusion:

Stock Sense is operationally feasible. The platform addresses user pain points, offers a better experience than existing platforms, and has a well-structured interface for both admins and users.

5.3 Economic Feasibility

Economic feasibility evaluates whether the benefits of the project justify its development cost.

- Cost of Development: Minimal; developed using existing hardware/software resources.
- Financial Benefits: The application's benefit of providing faster access to stock data and better investment decisions justifies the development cost.
- No Extra Cost: Uses free/open-source technologies where possible.

Conclusion:

Stock Sense is economically feasible. The development cost is low, and the expected benefits are high, making it a cost-effective investment.

Chapter6 :System Design

The Stock Sense system is designed to provide real-time stock information in a simple and user-friendly interface. It uses the Yahoo Finance module to directly fetch stock data without API calls, making the system faster and easier to manage.

Key Points of the Design:

1) User Interface (Frontend):

- Clean, simple, and responsive layout.
- Displays stock prices, changes, charts, and company details.
- Navigation includes: Home | Watchlist | News | Alerts.
- Technologies used: HTML, CSS, JavaScript (or EJS if server-rendered).

2) Backend (Application Layer):

- Uses Node.js with Express.js for server operations.
- No external API calls; uses yahoo-finance2 NPM module to fetch stock data directly.
- Manages routing, rendering, and interaction between frontend and database.

3) Database (Data Layer):

- MongoDB stores:
  - User details (name, email, password)
  - Watchlist (user's selected stocks)
  - Saved preferences.

4) Security Measures:

- Passwords are stored securely using hashing techniques.
- Only authenticated users can access personalized features.

5) Design Goals:

- Direct Yahoo Finance data fetching
- No third-party API dependency
- Fast loading and real-time updates
- Safe and secure for all user data
- Easy to maintain and expand in the future

6.1 Use Case Diagram

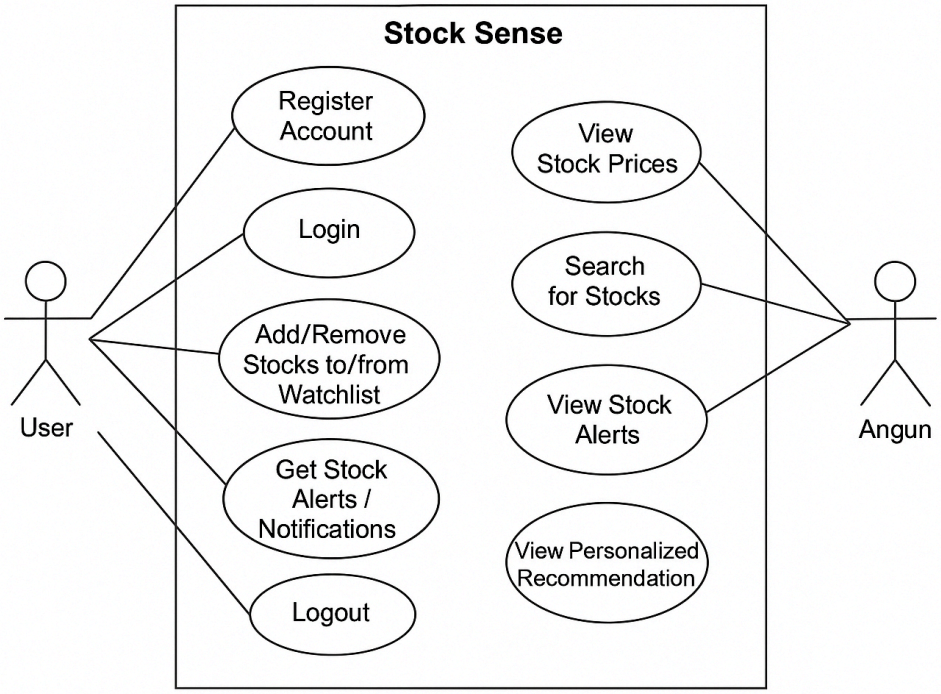


Figure 1.1

Use Case Diagram Description-StockSense

- This Use Case Diagram shows how a User interacts with the Stock Sense system. The user can register, login, and logout securely. Once logged in, the user can search for stocks, view real-time stock prices, and add or remove stocks from their personal watchlist.
- The system also provides personalized recommendations and instant alerts for stock price changes. This ensures that the user stays informed and makes better investment decisions.
- It shows all actions that a typical user can perform in the system.

6.2 ER Diagram

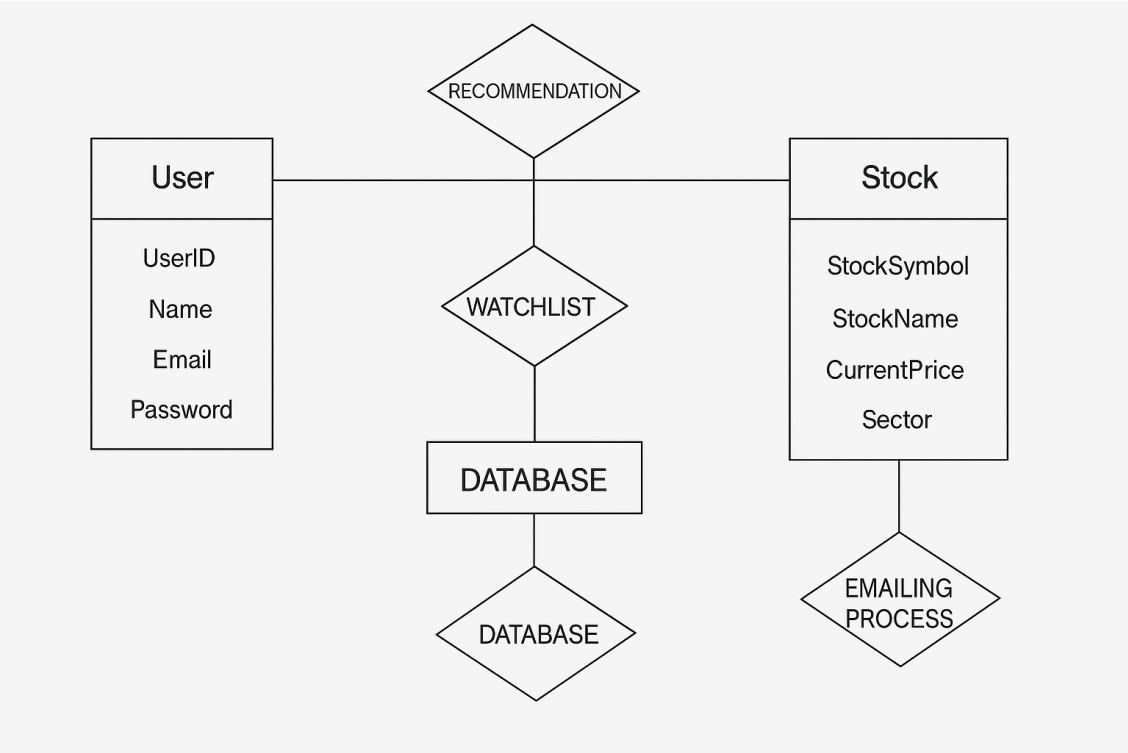


Figure 1.2

ER Diagram Description – Stock Sense

The ER diagram shows how different parts of Stock Sense work together.

- User: Stores user details like UserID, Name, Email, and Password.
- Stock: Contains details of stocks like Symbol, Name, and Current Price.
- Watchlist: Connects Users to their favorite Stocks.
- Database: Saves all user info, stock data, and watchlists.
- Emailing Process: Sends stock updates and recommendations to the user's email.

6.3 Data Flow Diagram

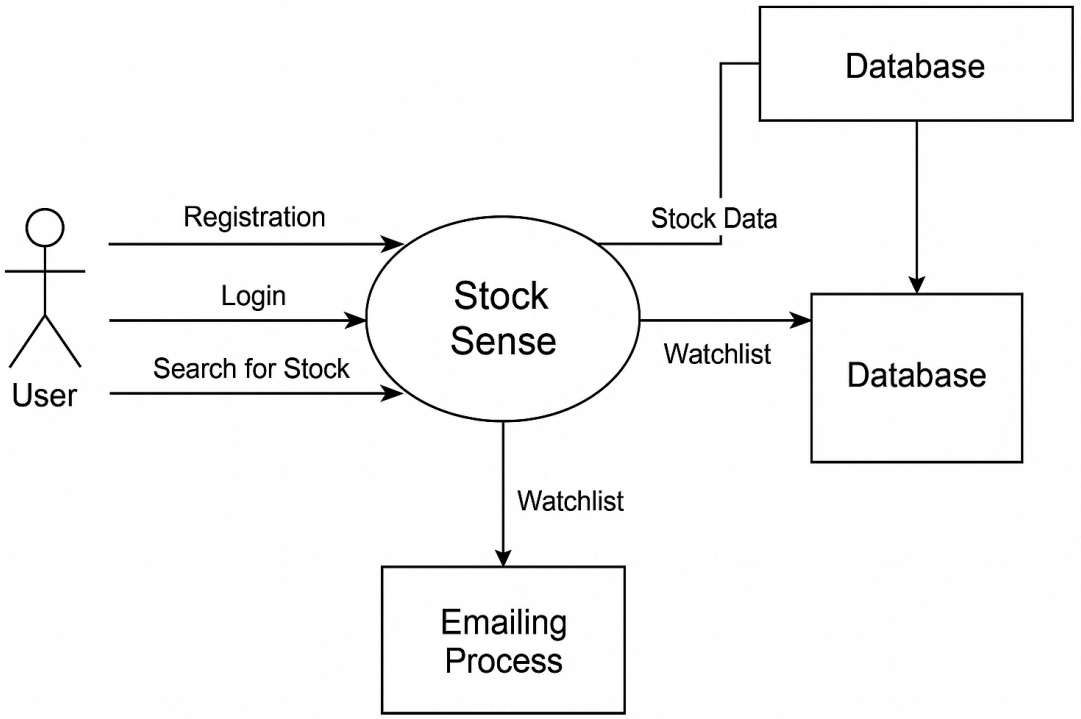


Figure 1.3

Data Flow Diagram Description – Stock Sense

The Data Flow Diagram (DFD) shows how data moves in the Stock Sense system:

- User: Can register, login, and search for stocks.
- Stock Sense System: Processes user actions, handles data, and manages the watchlist.
- Database: Stores user details, stock data, and watchlist information.
- Emailing Process: Uses the watchlist data to send stock updates to the user's email.

6.4 Zero/0 And Second Level DFD

Zero/0 Level DFD :-

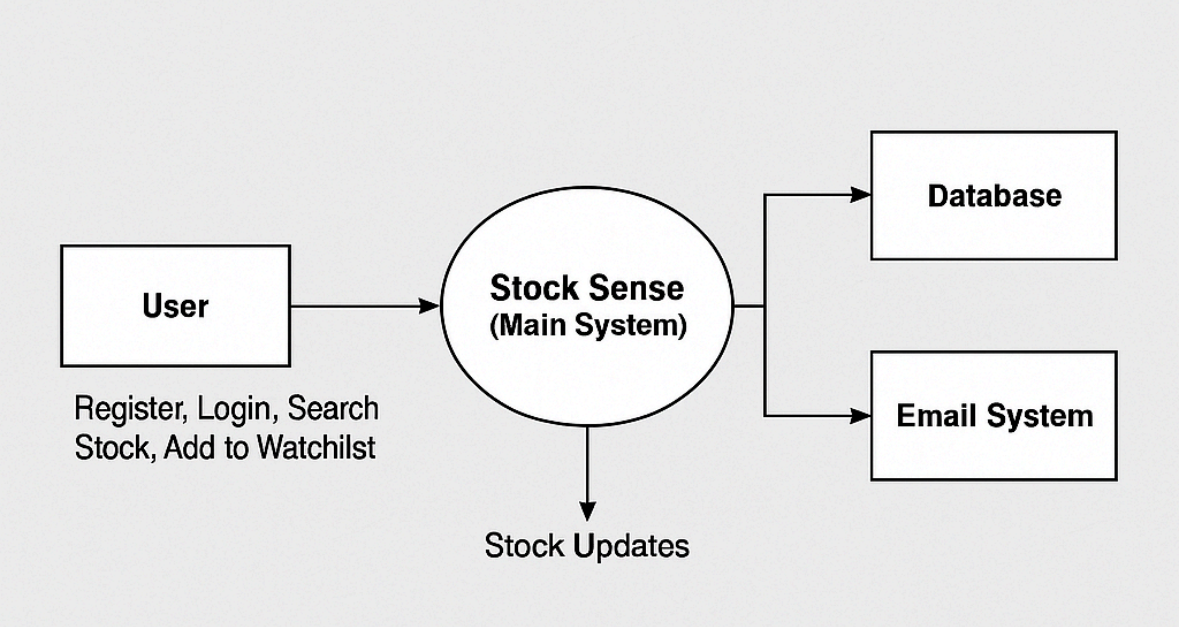


Figure 1.4

Second Level DFD :-

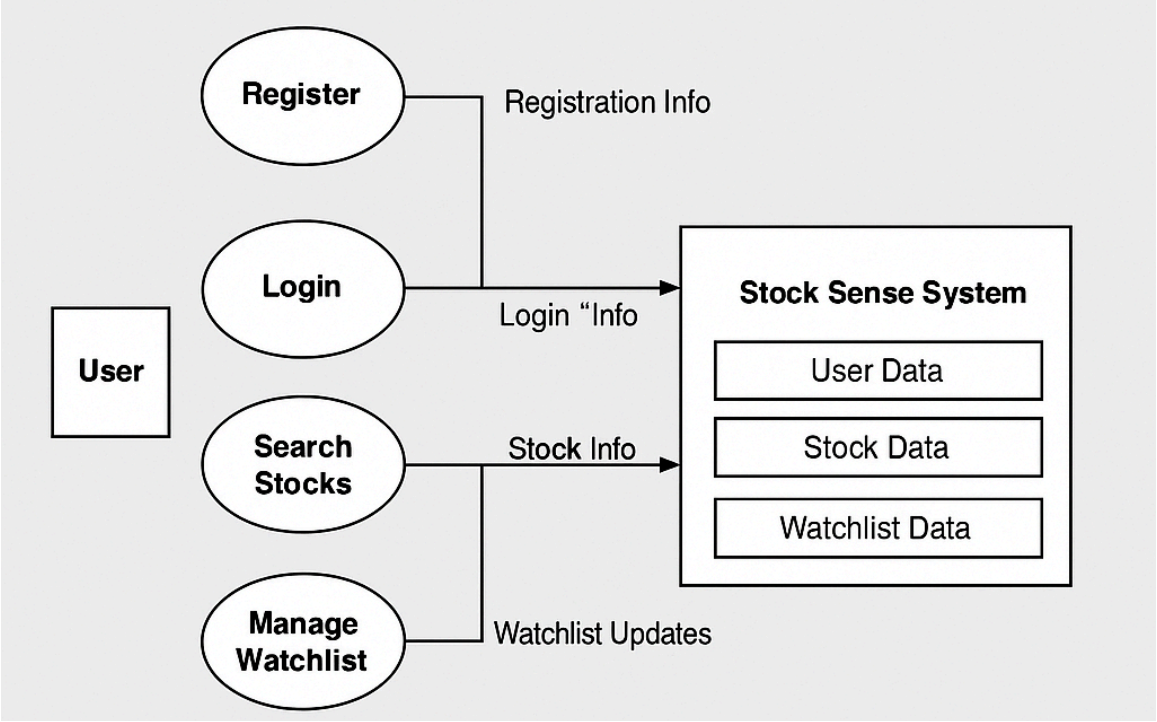


Figure 1.5

Chapter7:DataBase Design

Database Design (MongoDB) – Stock Sense

The database of Stock Sense is designed using MongoDB, a NoSQL database well-suited for handling flexible and scalable data structures. The system contains four main collections: users, stocks, watchlists, and emails.

- The users collection stores all registered user information like name, email, and hashed passwords.
- The stocks collection maintains real-time stock data including stock symbols, company names, and their latest prices.
- The watchlists collection connects users to the stocks they are interested in tracking, forming the core feature of personalized stock monitoring.
- Finally, the emails collection (optional) keeps a record of the notifications and stock alerts sent to users.

The database design of Stock Sense focuses on simplicity and efficiency. The core collections include users, stocks, and watchlists, ensuring that essential user and stock data is organized for quick access and retrieval. The watchlists collection is kept minimal, storing only the email of the user and the stock symbol, making it lightweight and easy to manage. This design reduces complexity and improves performance by avoiding unnecessary relationships. It also aligns well with MongoDB's flexible document model, ensuring scalability for future enhancements.

1. users Collection


```
{
  "_id": {
    "$oid": "685389634c9e09ce14acf869"
  },
  "name": "Mohammed Hussain",
  "email": "sahilsahu7816@gmail.com",
  "password":
"$2b$10$6gUK6AzlqyacXyL2ihjm2eq.1Nl4cnmDUVDxxWltHZa07DEhzCYQq"
,
  "phone": "9019274981",
  "date": "2025-06-19",
  "__v": 0
}
```


2.StockSymbols Collection


```
{
  "_id": {
    "$oid": "6853c71ca2809053f5776ba1"
  },
  "email": "mdsalman7414@gmail.com",
  "symbol": "MARUTI.NS",
  "__v": 0
}
```

Chapter8 :Coding

The Stock Sense platform is developed using Node.js with Express.js for the backend and MongoDB for database management. The Yahoo Finance module is used to fetch real-time stock market data without relying on external API calls. Below is a sample code to fetch stock data and store it in MongoDB:

Coding Section

- Language: Node.js (JavaScript)
- Framework: Express.js
- Database: MongoDB
- Main Modules:
  - express
  - mongoose
  - yahoo-finance2
- Database Collections:
  - users → email, password
  - stocks → symbol, name, price
  - watchlists → email, symbol
- Main Functions:
  - Fetch stock using Yahoo Finance module
  - Save stock data to database
  - Add stock to watchlist (email + symbol)
- No API calls → Only Yahoo Finance Module used internally
- Optional Feature: Email alerts (can use nodemailer)

/#main file to run server # index.js

```
const express = require('express');
const xml2js = require('xml2js');
const yf = require('yahoo-finance2').default;
const transporter = require('./utils/transport')
const path = require('path')
const cookieParser = require('cookie-parser');
const push = require('./controllers/push')
const symbolpush = require('./controllers/symboldb')
const login = require('./controllers/login')
const axios = require('axios')
const db = require('./config/db')
const parser = new xml2js.Parser();
const cheerio = require('cheerio');
const flash = require("connect-flash");
const expressSession = require("express-session");
const usermodel = require('./models/usermodel');
const symboldb = require('./models/symbol')
const {sensexSymbols,bankexSymbols,teckSymbols,midcapSymbols,smallcapSymbols,autoSymbols} =
require('./controllers/BSE_symbol');
const {
nifty50Symbols,niftyBankSymbols,niftyITSymbols,niftyMidcapSymbols,niftySmallcapSymbols,niftyAutoSy
mbols} = require('./controllers/NSE_symbol');
console.log(nifty50Symbols)
const app = express();
const PORT = 4000;
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json())
app.use(expressSession({
 resave: false,
 saveUninitialized: false,
 secret: "dddddd"
})
)
app.use(flash())
app.use(cookieParser());
app.use(express.urlencoded({ extended: true }));
app.set('view engine', 'ejs');
function loginmiddelware(req, res, next) {
 if (!req.cookies.email || req.cookies.email == 0 || req.cookies.email.length == 5) {
  let suc = req.flash("error", "please Log in for further use");
  res.redirect("/register/user/login/user");
 } else {
  next();
 }
}
```

/#main file to run server # index.js

```
app.get('/bse-sensex', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   sensexSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     const summary = await yf.quoteSummary(symbol, { modules: ['financialData', 'defaultKeyStatistics'] });

     const financialData = summary?.financialData || {};
     const keyStats = summary?.defaultKeyStatistics || {};

     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,


      peRatio: keyStats?.forwardPE || financialData?.forwardPE || 0,       // P/E Ratio
      eps: keyStats?.forwardEps || financialData?.epsCurrentYear || 0,     // EPS
      debtToEquity: financialData?.debtToEquity || 0,              // D/E
      opm: financialData?.operatingMargins || 0,               // OPM (Operating Profit Margin)
      roe: financialData?.returnOnEquity || 0,                // ROE (Return on Equity)
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
      peRatio: 0,
      eps: 0,
      debtToEquity: 0,
      opm: 0,
      roe: 0,
     };
    }
   })
  );

  res.render('pagesBse/Sensex', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching stock data:', error);
  res.send('Error fetching stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/stock/:symbol', loginmiddelware, async (req, res) => {
 let suc = req.flash("error");
 const symbol = req.params.symbol.toUpperCase();

 try {
  const quote = await yf.quote(symbol);
  const summary = await yf.quoteSummary(symbol, {
   modules: ['financialData', 'assetProfile']
  });
  const financialData = summary.financialData || {};
  const assetProfile = summary.assetProfile || {};

  const today = new Date();
  const pastDate400 = new Date();
  pastDate400.setDate(today.getDate() - 400); // Get enough data for 1 year return

  const history = await yf.historical(symbol, {
   period1: pastDate400,
   period2: today,
   interval: '1d'
  });

  const closingPrices = history.map(item => item.close);
  const currentPrice = quote.regularMarketPrice;

  const calculateSMA = (data, period) => {
   if (data.length < period) return 'N/A';
   const sum = data.slice(-period).reduce((acc, val) => acc + val, 0);
   return (sum / period).toFixed(2);
  };

  // ✅ Calculate % returns
  const calcReturn = (pastPrice, current) => {
   if (!pastPrice || pastPrice === 0) return 'N/A';
   return (((current - pastPrice) / pastPrice) * 100).toFixed(2) + '%';
  };

  // Find prices from 7 days ago and ~365 days ago
  const price1WeekAgo = history.length >= 7 ? history[history.length - 7].close : null;
  const price1YearAgo = history.length >= 250 ? history[history.length - 250].close : null;

  const returns = {
   oneWeek: calcReturn(price1WeekAgo, currentPrice),
   oneYear: calcReturn(price1YearAgo, currentPrice)
  };
```

/#main file to run server # index.js

```
app.get('/stock/:symbol', loginmiddelware, async (req, res) => {
 let suc = req.flash("error");
 const symbol = req.params.symbol.toUpperCase();

 try {
  const quote = await yf.quote(symbol);
  const summary = await yf.quoteSummary(symbol, {
   modules: ['financialData', 'assetProfile']
  });
  const financialData = summary.financialData || {};
  const assetProfile = summary.assetProfile || {};

  const today = new Date();
  const pastDate400 = new Date();
  pastDate400.setDate(today.getDate() - 400); // Get enough data for 1 year return

  const history = await yf.historical(symbol, {
   period1: pastDate400,
   period2: today,
   interval: '1d'
  });

  const closingPrices = history.map(item => item.close);
  const currentPrice = quote.regularMarketPrice;

  const calculateSMA = (data, period) => {
   if (data.length < period) return 'N/A';
   const sum = data.slice(-period).reduce((acc, val) => acc + val, 0);
   return (sum / period).toFixed(2);
  };

  // ✅ Calculate % returns
  const calcReturn = (pastPrice, current) => {
   if (!pastPrice || pastPrice === 0) return 'N/A';
   return (((current - pastPrice) / pastPrice) * 100).toFixed(2) + '%';
  };

  // Find prices from 7 days ago and ~365 days ago
  const price1WeekAgo = history.length >= 7 ? history[history.length - 7].close : null;
  const price1YearAgo = history.length >= 250 ? history[history.length - 250].close : null;

  const returns = {
   oneWeek: calcReturn(price1WeekAgo, currentPrice),
   oneYear: calcReturn(price1YearAgo, currentPrice)
  };
```

/#main file to run server # index.js

```
const sma30 = calculateSMA(closingPrices, 30);
const sma100 = calculateSMA(closingPrices, 100);

const percentDiff = (current, ma) => {
 if (ma === 'N/A') return 'N/A';
 return (((current - ma) / ma) * 100).toFixed(2) + '%';
};

const ratios = {
 debtToEquity: financialData.debtToEquity || 'N/A',
 returnOnEquity: financialData.returnOnEquity || 'N/A',
 operatingMargins: financialData.operatingMargins || 'N/A',
 interestCoverage: financialData.interestCoverage || 'N/A',
 roce: 'N/A',
 salesGrowth3Years: 'N/A'
};

const companyDetails = {
 sector: assetProfile.sector || 'N/A',
 industry: assetProfile.industry || 'N/A',
 ceo: assetProfile.companyOfficers && assetProfile.companyOfficers[0]
  ? assetProfile.companyOfficers[0].name
  : 'N/A',
 website: assetProfile.website || 'N/A',
 description: assetProfile.longBusinessSummary || 'N/A'
};

const technicals = {
 sma30,
 sma100,
 sma30Percent: percentDiff(currentPrice, parseFloat(sma30)),
 sma100Percent: percentDiff(currentPrice, parseFloat(sma100))
};

const upcomingProbability = () => {
 if (sma100 === 'N/A' || sma30 === 'N/A') return 'Insufficient data for prediction';
 if (currentPrice > sma100) return 'Probability of Uptrend (Long-Term): High';
 if (currentPrice < sma100) return 'Probability of Downtrend (Long-Term): High';
 return 'Stable/Sideways Market Expected';
};
```

/#main file to run server # index.js

```
 const trendData = {
   upcoming: upcomingProbability(),
   shortTerm: (currentPrice > sma30) ? 'Short-term Momentum: Positive' : 'Short-term Momentum: Weak'
  };

  res.render('stock-details', {
   quote,
   ratios,
   history,
   technicals,
   trendData,
   returns,    //
   companyDetails, //    suc
  });

 } catch (error) {
  console.error(error);
  res.send('Error fetching detailed stock data');
 }
});




app.get('/bse-bankex', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   bankexSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesBse/Bankex', { stocks: quotes }); // Make sure Bankex.ejs exists
 } catch (error) {
  console.error('Error fetching Bankex data:', error);
  res.send('Error fetching Bankex data');
 }
});
```

/#main file to run server # index.js

```
app.get('/bse-teck', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   teckSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesBse/BSETeck', { stocks: quotes }); // Make sure Teck.ejs exists
 } catch (error) {
  console.error('Error fetching Teck data:', error);
  res.send('Error fetching Teck data');
 }
});
```

/#main file to run server # index.js

```
app.get('/bse-midcap', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   midcapSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesBSE/BSEmidcap', { stocks: quotes }); // Make sure you create Midcap.ejs
 } catch (error) {
  console.error('Error fetching MidCap data:', error);
  res.send('Error fetching MidCap data');
 }
});
```

/#main file to run server # index.js

```
app.get('/bse-smallcap', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   smallcapSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesBse/BseSmall', { stocks: quotes }); // Ensure Smallcap.ejs exists
 } catch (error) {
  console.error('Error fetching SmallCap data:', error);
  res.send('Error fetching SmallCap data');
 }
});
```

/#main file to run server # index.js

```
app.get('/bse-auto', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   autoSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesBse/BseAuto', { stocks: quotes }); // Make sure Auto.ejs exists
 } catch (error) {
  console.error('Error fetching Auto data:', error);
  res.send('Error fetching Auto data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-nifty', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   nifty50Symbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/Nifty', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching stock data:', error);
  res.send('Error fetching stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-banknifty', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   niftyBankSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/NiftyBank', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching Bank Nifty stock data:', error);
  res.send('Error fetching Bank Nifty stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-it', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   niftyITSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/NiftyIT', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching IT stock data:', error);
  res.send('Error fetching IT stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-midcap', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   niftyMidcapSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/NiftyMidcap', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching Midcap stock data:', error);
  res.send('Error fetching Midcap stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-smallcap', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   niftySmallcapSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/NiftySmallcap', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching Smallcap stock data:', error);
  res.send('Error fetching Smallcap stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/nse-auto', loginmiddelware, async (req, res) => {
 try {
  const quotes = await Promise.all(
   niftyAutoSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesNse/NiftyAuto', { stocks: quotes });
 } catch (error) {
  console.error('Error fetching Auto stock data:', error);
  res.send('Error fetching Auto stock data');
 }
});
```

/#main file to run server # index.js

```
app.get('/search/stock', loginmiddelware, async (req, res) => {
 const searchQuery = req.query.query;
 if (!searchQuery) {
  return res.render('search', { query: '', results: [] });
 }

 try {
  // Step 1: Search for symbols matching the query
  const searchResults = await yf.search(searchQuery);

  if (!searchResults.quotes || searchResults.quotes.length === 0) {
   return res.render('search', { query: searchQuery, results: [] });
  }

  // Step 2: Get top 5 symbols from search results
  const topSymbols = searchResults.quotes.slice(0, 5).map(q => q.symbol);

  // Step 3: Define modules for detailed info
  const modules = [
   'price',
   'summaryProfile',
   'financialData',
   'defaultKeyStatistics',
   'calendarEvents'
  ];

  // Step 4: Fetch detailed data for all symbols in parallel
  const detailedResults = await Promise.all(
   topSymbols.map(symbol =>
    yf.quoteSummary(symbol, { modules }).catch(err => {
     console.error(`Error fetching details for ${symbol}`, err);
     return null; // Skip any failed fetches gracefully
    })
   )
  );

  // Filter out null responses from failed fetches
  const validResults = detailedResults.filter(item => item !== null);

  // Step 5: Render the EJS view with detailed results
  res.render('search', {
   query: searchQuery,
   results: validResults
  });

 } catch (error) {
  console.error('Error in stock search:', error);
  res.render('search', { query: searchQuery, results: [] });
 }
});
```

/#main file to run server # index.js

```
let cachedNews = [];
app.get('/news', loginmiddelware, async (req, res) => {
 try {
  const rssUrl = 'https://feeds.finance.yahoo.com/rss/2.0/headline?s=%5EDJI&region=US&lang=en-
US';
  const response = await axios.get(rssUrl);
  const result = await parser.parseStringPromise(response.data);

  cachedNews = result.rss.channel[0].item;

  res.render('news', { newsList: cachedNews });
 } catch (error) {
  console.error(error);
  res.status(500).send('Error fetching news');
 }
});
// Route: Show Full News Info by ID
app.get('/news/:id', loginmiddelware, async (req, res) => {
 const newsItem = cachedNews[req.params.id];
 if (!newsItem) return res.status(404).send('News not found');

 try {
  const articleUrl = newsItem.link[0];
  const response = await axios.get(articleUrl);
  const $ = cheerio.load(response.data);

  const articleText = $('article').text().trim().slice(0, 3000); // Trim to avoid overflow
  const image = $('article img').first().attr('src') || null;

  res.render('newsDetail', {
   news: newsItem,
   articleText,
   image,
  });
 } catch (err) {
  console.error("Error scraping full article:", err);
  res.render('newsDetail', {
   news: newsItem,
   articleText: 'Full content could not be loaded.',
   image: null,
  });
 }
});
```

/#main file to run server # index.js

```
app.get("/", function (req, res) {
 res.cookie("email", "" || 0)
 res.render("index")
})

app.get("/register/user/login/user", function (req, res) {
 let suc = req.flash("error")
 res.render("registerlogin", { suc })
})

app.post("/register/data", function (req, res) {
 let { name, email, password, phone } = req.body
 push(name, email, password, phone, req, res)
})

app.post("/login/data", function (req, res) {
 let { email, password } = req.body
 login(email, password, req, res)

})
```

/#main file to run server # index.js

```
const indices = [
 { name: 'BSE SENSEX', symbol: '^BSESN', route: "/bse-sensex" },
 { name: 'BSE Bankex', symbol: 'BSE-BANK.BO', route: "/bse-bankex" },
 { name: 'BSE Teck', symbol: ' BSE-TECK.BO', route: "/bse-teck" },
 { name: 'BSE Midcap', symbol: 'BSE-MIDCAP.BO', route: "/bse-midcap" },
 { name: 'BSE Smallcap', symbol: 'BSE-SMLCAP.BO', route: "/bse-smallcap" },
 { name: 'BSE Auto', symbol: ' BSE-AUTO.BO', route: "/bse-auto" }
];
app.get('/home.BSE', async (req, res) => {
 try {
  const data = await Promise.all(indices.map(async (index) => {
   try {
    const quote = await yf.quote(index.symbol);
    if (!quote || !quote.regularMarketPrice) throw new Error('No data');
    return {
     name: index.name,
     route: index.route,
     symbol: index.symbol,
     price: quote.regularMarketPrice,
     change: quote.regularMarketChange,
     changePercent: quote.regularMarketChangePercent
    };
   } catch (err) {
    console.error(`Error fetching ${index.name}:`, err.message);
    return {
     name: index.name,
     symbol: index.symbol,
     price: null,
     change: null,
     changePercent: null
    };
   }
  }));

  res.render('pages/BSEind', { indices: data });
 } catch (err) {
  console.error(err);
  res.redirect("/error/404/no/data/to/proceed/")
 }
});
```

/#main file to run server # index.js

```
const nseIndices = [
 { name: 'Nifty 50', symbol: '^NSEI', route: "/nse-nifty" },
 { name: 'Nifty Bank', symbol: '^NSEBANK', route: "/nse-banknifty" },
 { name: 'Nifty IT', symbol: '^CNXIT', route: "/nse-it" },
 { name: 'Nifty Midcap 100', symbol: 'NIFTY_MIDCAP_100.NS', route: "/nse-midcap" },
 { name: 'Nifty Smallcap 100', symbol: '^CNXSC', route: "/nse-smallcap" },
 { name: 'Nifty Auto', symbol: '^CNXAUTO', route: "/nse-auto" }
];

app.get('/home.NSE', async (req, res) => {
 try {
  const data = await Promise.all(nseIndices.map(async (index) => {
   try {
    const quote = await yf.quote(index.symbol);
    if (!quote || !quote.regularMarketPrice) throw new Error('No data');
    return {
     name: index.name,
     route: index.route,
     symbol: index.symbol,
     price: quote.regularMarketPrice,
     change: quote.regularMarketChange,
     changePercent: quote.regularMarketChangePercent
    };
   } catch (err) {
    console.error(`Error fetching ${index.name}:`, err.message);
    return {
     name: index.name,
     symbol: index.symbol,
     price: null,
     change: null,
     changePercent: null
    };
   }
  }));

  res.render('pages/NSEind', { indices: data });
 } catch (err) {
  console.error(err);
  res.redirect("/error/404/no/data/to/proceed/")
 }
});
```

/#main file to run server # index.js

```
app.get('/market/:symbol', async (req, res) => {
 const rawSymbol = req.params.symbol;
 // Add ^ prefix if not present (common for indices)
 const symbol = rawSymbol.startsWith('^') ? rawSymbol : `^${rawSymbol}`;

 try {
  // Fetch summary details
  const summary = await yf.quoteSummary(symbol, { modules: ['price', 'summaryDetail'] });

  // Prepare details for rendering
  const details = {
   symbol,
   name: summary?.price?.longName || symbol,
   currency: summary?.price?.currency || 'N/A',
   exchange: summary?.price?.exchangeName || 'N/A',
   price: summary?.price?.regularMarketPrice || 'N/A',
   open: summary?.summaryDetail?.open || 'N/A',
   previousClose: summary?.summaryDetail?.previousClose || 'N/A',
   dayHigh: summary?.summaryDetail?.dayHigh || 'N/A',
   dayLow: summary?.summaryDetail?.dayLow || 'N/A',
   marketCap: summary?.summaryDetail?.marketCap || 'N/A',
   volume: summary?.summaryDetail?.volume || 'N/A'
  };

  // Fetch chart data for last 30 days
  const chartData = await yf.chart(symbol, {
   period1: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000), // 30 days ago
   period2: new Date(),
   interval: '1d'
  });

  // Safe check for chart data
  const result = chartData?.chart?.result?.[0];
  if (!result || !result.indicators?.quote?.[0]?.close || !result.timestamp) {
   // Render without chart data
   return res.render('pagesMarket/MarketDetail', {
    details,
    graphData: [],
    noChart: true
   });
  }

  // Prepare graph data for chart
  const prices = result.indicators.quote[0].close;
  const timestamps = result.timestamp;

  const graphData = timestamps.map((timestamp, i) => ({
   date: new Date(timestamp * 1000).toLocaleDateString(),
   close: prices[i]
  }));

  // Render the page with details and graph data
  res.render('pagesBse/MarketDetail', {
   details,
   graphData,
   noChart: false
  });

 } catch (err) {
  console.error('Error fetching market detail:', err.message);
  res.redirect("/error/404/no/data/to/proceed/")
 }
});
```

/#main file to run server # index.js

```
app.get('/market', async (req, res) => {
 try {
  const quotes = await Promise.all(
   marketSymbols.map(async (symbol) => {
    try {
     const result = await yf.quote(symbol);
     return {
      symbol,
      shortName: result?.shortName || symbol,
      price: result?.regularMarketPrice || 0,
      change: result?.regularMarketChange || 0,
      changePercent: result?.regularMarketChangePercent || 0,
     };
    } catch (err) {
     console.error(`Error fetching data for ${symbol}:`, err.message);
     return {
      symbol,
      shortName: symbol,
      price: 0,
      change: 0,
      changePercent: 0,
     };
    }
   })
  );

  res.render('pagesMarket/Market', { markets: quotes });
 } catch (error) {
  console.error('Error fetching market data:', error);
  res.redirect("/error/404/no/data/to/proceed/")
 }

});

app.get("/watch/list/", function (req, res) {
 console.log(req.query.symbol)
 symbolpush(req.cookies.email, req.query.symbol, req, res)
})

app.get("/error/404/no/data/to/proceed/", function (req, res) {
 res.render("error.ejs")
})
```

/#main file to run server # index.js

```
app.get("/Profile", async function (req, res) {
 const user = await usermodel.findOne({ email: req.cookies.email });
 const symbols = await symboldb.find({ email: req.cookies.email }) // or use req.user
from your login middleware
 res.render('profile', { user, symbols });
})

app.get('/watchlist', loginmiddelware, async (req, res) => {
 const userEmail = req.cookies.email;
 const symbols = await symboldb.find({ email: userEmail });
 res.render('watchlist', { symbols });
});

app.get('/watch/remove', loginmiddelware, async (req, res) => {
 const userEmail = req.cookies.email;
 const { symbol } = req.query;

 if (!symbol) return res.send('No symbol provided');

 const userWatchlist = await symboldb.findOne({ email: userEmail });

 if (userWatchlist) {
  userWatchlist.symbols = userWatchlist.symbols.filter(s => s !== symbol);
  await userWatchlist.save();
 }

 res.redirect('/watchlist');
});
```

/#main file to run server # index.js

```
app.get('/watch/remove/:symbol', async (req, res) => {
 const email = req.cookies.email;
 const symbol = req.params.symbol;

 try {
  await symboldb.deleteOne({ email, symbol });
  res.redirect('/watchlist');
 } catch (error) {
  console.error(error);
  res.send('Error removing symbol from watchlist.');
 }
});
app.listen(PORT, () => {
 console.log(`Server running on http://localhost:${PORT}`);
});
```

#db connectivity #config/db.js

```
const mongoose = require('mongoose')
mongoose.connect('mongodb://localhost:27017/StockData')
.then(()=>{
  console.log("connected")
})
.catch((err)=>{
console.log(err)
})

module.exports=mongoose.connection;
```

controllers/push.js

```
const mongoose = require('mongoose')
const usermodel = require('../models/usermodel')
const cookieParser = require('cookie-parser');
const bcrypt = require('bcrypt')
const transporter = require('../utils/transport')

async function push(name, email, password, phone, req, res) {
 console.log(name, email, password, phone)
 let existing = await usermodel.findOne({ email: email })
 if (existing) {
  req.flash("error", "Already Registered Please Login.");
  res.redirect("/register/user/login/user")
 }
 else {
  const hashedPassword = bcrypt.hashSync(password, 10);
  console.log(hashedPassword)
  res.cookie("email", email, { maxAge: 1000 * 60 * 60, })
  let user = await usermodel.create({
   name: name,
   email: email,
   password: hashedPassword,
   phone: phone
  })
  if (user) {

   async function main() {
    // send mail with defined transport object
    const info = await transporter.sendMail({
     from: '"StockSense Official" <thestocksenseofficial@gmail.com>', // sender address
     to: user.email, // list of receivers
     subject: "Welcome to StockSense", // Subject line
     html:
```

controllers/push.js

```
<p>Dear ${user.name},</p>

<p>Welcome to StockSense!</p>

<p>We're excited to have you join our growing community of smart investors and market enthusiasts. With StockSense, you get access to trusted insights, real-time updates, and powerful tools to guide your financial journey.</p>

<p>Stay connected, stay informed, and grow with confidence.</p>

<p>Thank you for choosing StockSense.</p>

<p>Warm regards,<br>
Team StockSense</p>`
  });

  console.log("Message sent: %s", info.messageId);
  // Message sent: <d786aa62-4e0a-070a-47ed-0b0666549519@ethereal.email>
 }

 main().catch(console.error);


 res.redirect("/home")
 }
}



}
module.exports = push;
```

controllers/login.js

```
const mongoose = require('mongoose')
const usermodel = require('../models/usermodel')
const cookieParser = require('cookie-parser');
const bcrypt = require('bcrypt')

async function login ( email , password , req, res ){
  let user = await usermodel.findOne({email:email})
  if(!user){
   req.flash("error", "No account found. Please create an account.");
   return res.redirect("/register/user/login/user#");

  }
  else{
     const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
       req.flash("error", "Invaid Credentials");
      return res.redirect("/register/user/login/user");
    }

  res.cookie('email', user.email, {
   maxAge: 1000 * 60 * 60 * 24, // 24 hours validity

  });
  res.redirect("/home")
  }

}

module.exports=login;
```

Models/usermodel.js

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

const userSchema = mongoose.Schema({
 name: {
  type: String,
  minlength: 3, // Correct spelling is minlength, not minLenght
 },
 email: {
  type: String,
  required: true,
  unique: true,
 },
 password: {
  type: String,
  required: true,
 },
 phone: {
  type: String,
 },
 date: {
  type: String,
  default: () => new Date().toISOString().split('T')[0]
 }
});

module.exports = mongoose.model("Account", userSchema);
```

#Models/symbol.js

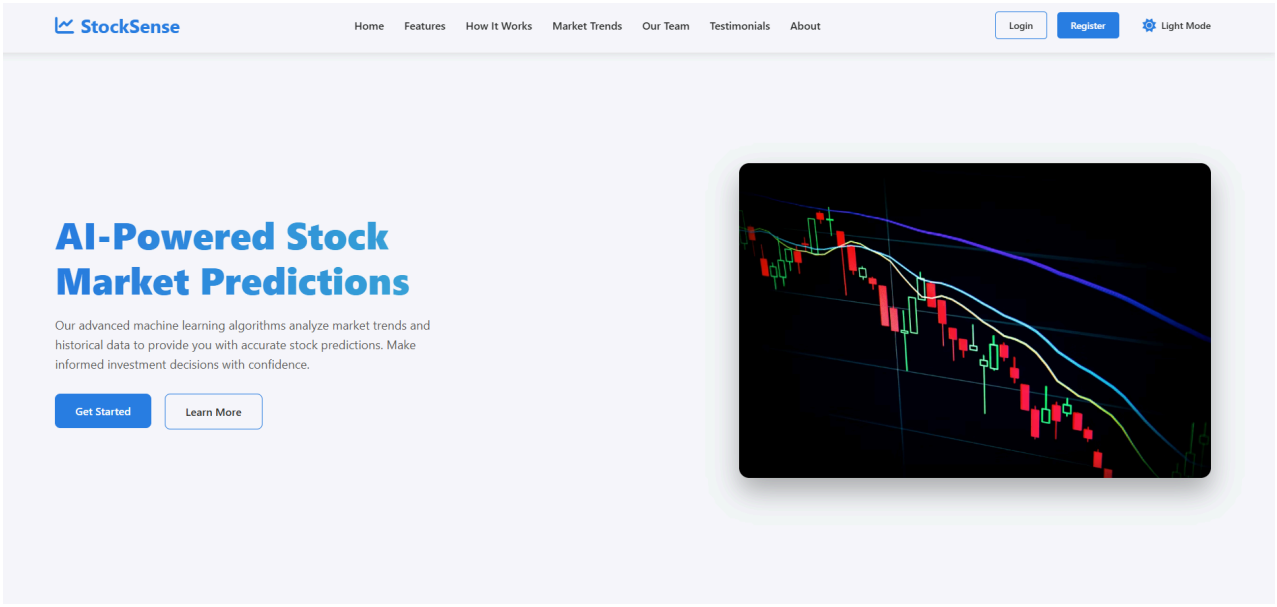
```
const mongoose = require('mongoose')


const symbolSchema =mongoose.Schema( {

  email:{
    type:String,

  },
  symbol:{
    type:String,
  }




});

module.exports=mongoose.model("symbolsdb",symbolSchema)
```

 # Models/symbol.js

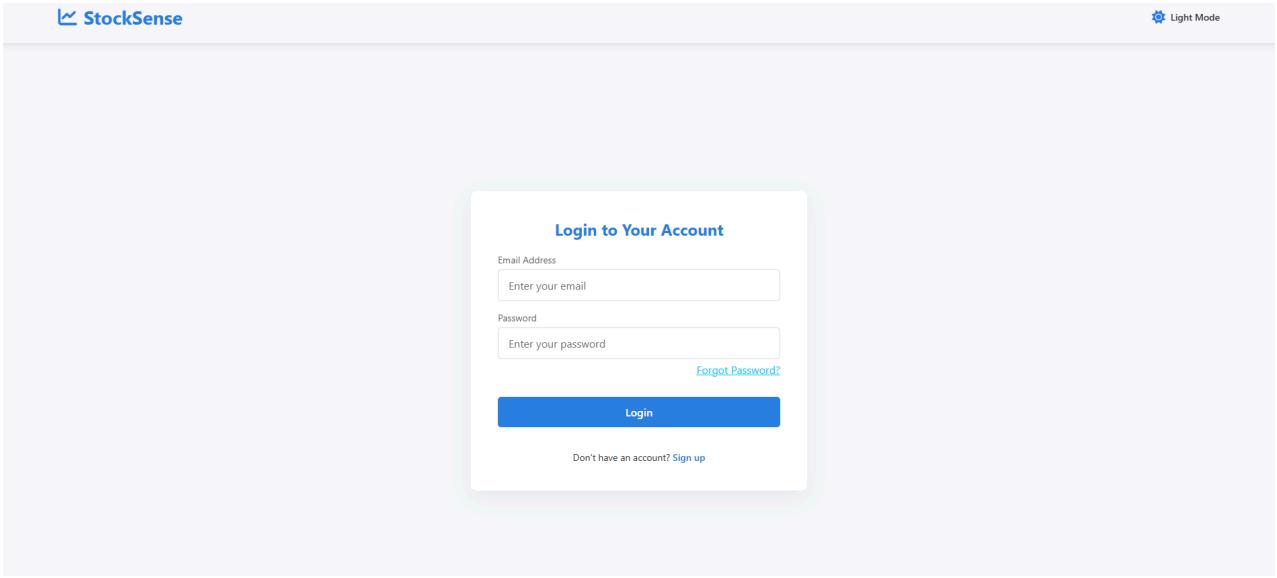
```
const mongoose = require('mongoose')


const symbolSchema =mongoose.Schema( {

  email:{
    type:String,

  },
  symbol:{
    type:String,
  }




});

module.exports=mongoose.model("symbolsdb",symbolSchema)
```

Chapter9 :Output ScreenShots

Indexpage :-

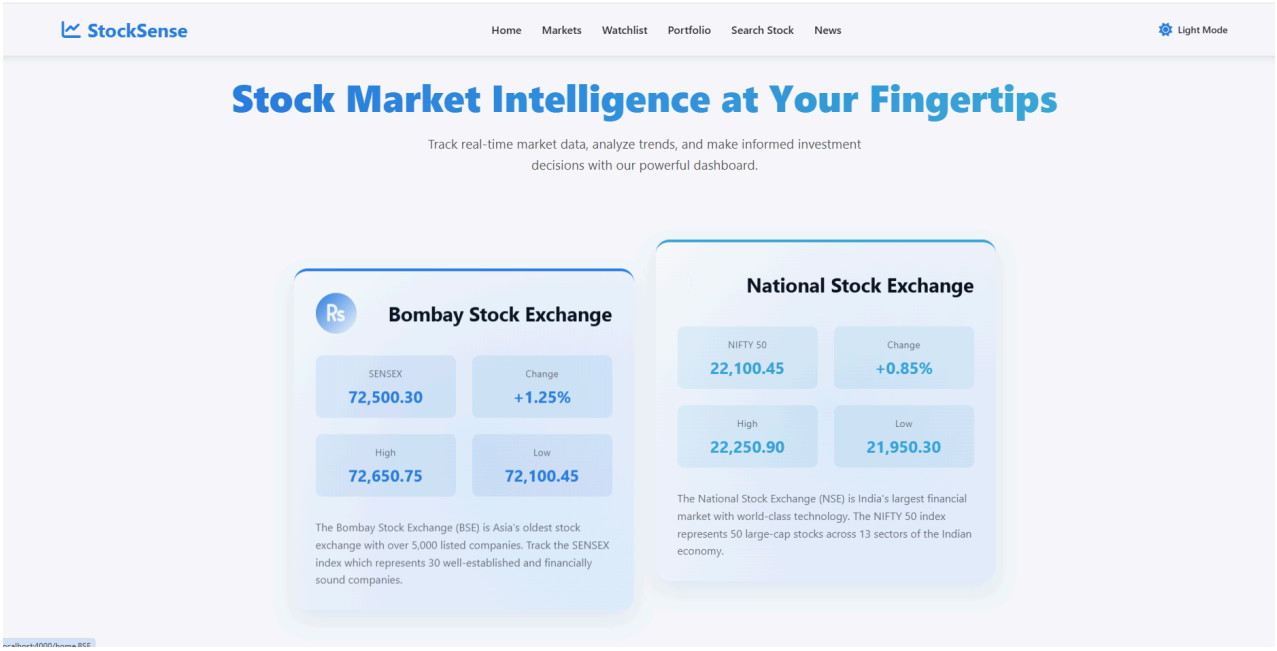


Login Page :-

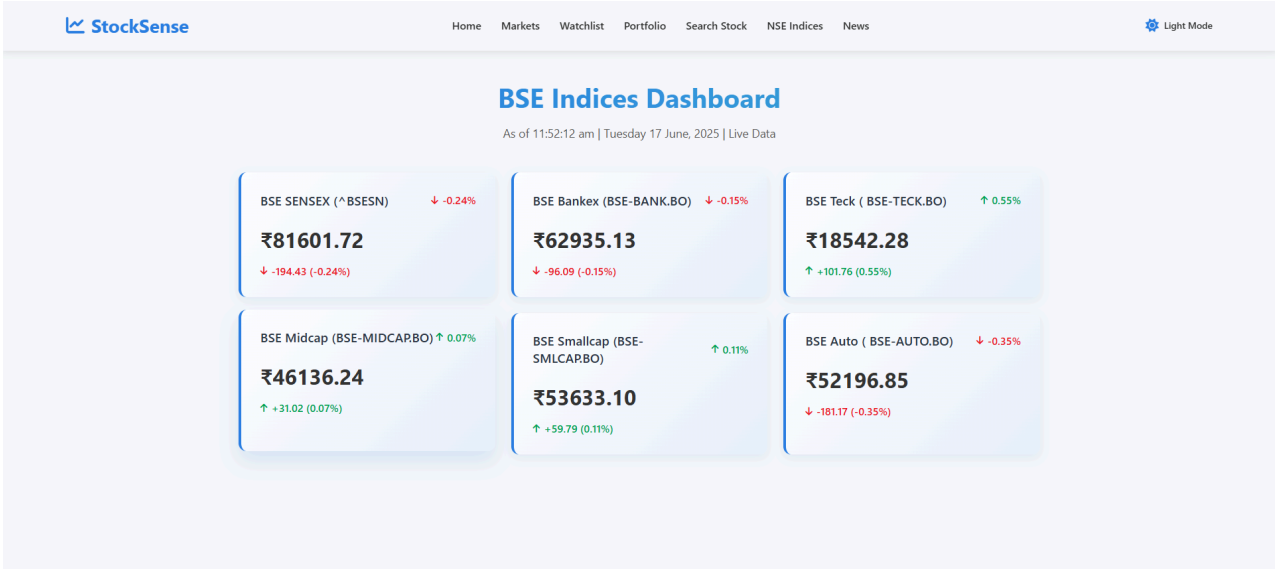


Output ScreenShots

Homepage :-

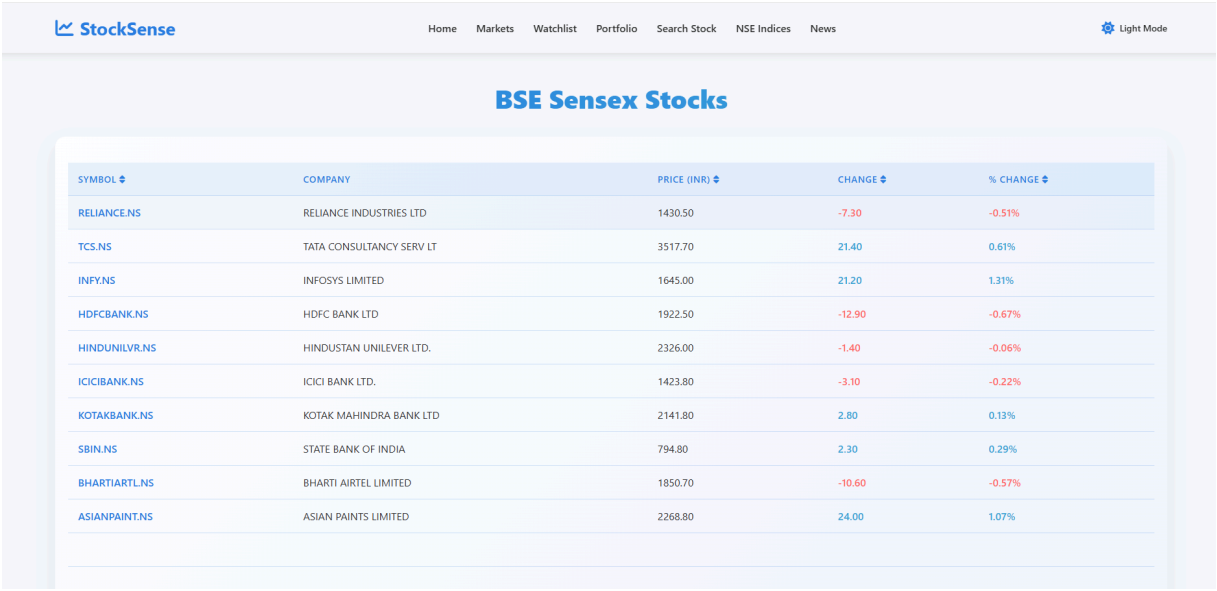


Stock Indices :-



Output ScreenShots

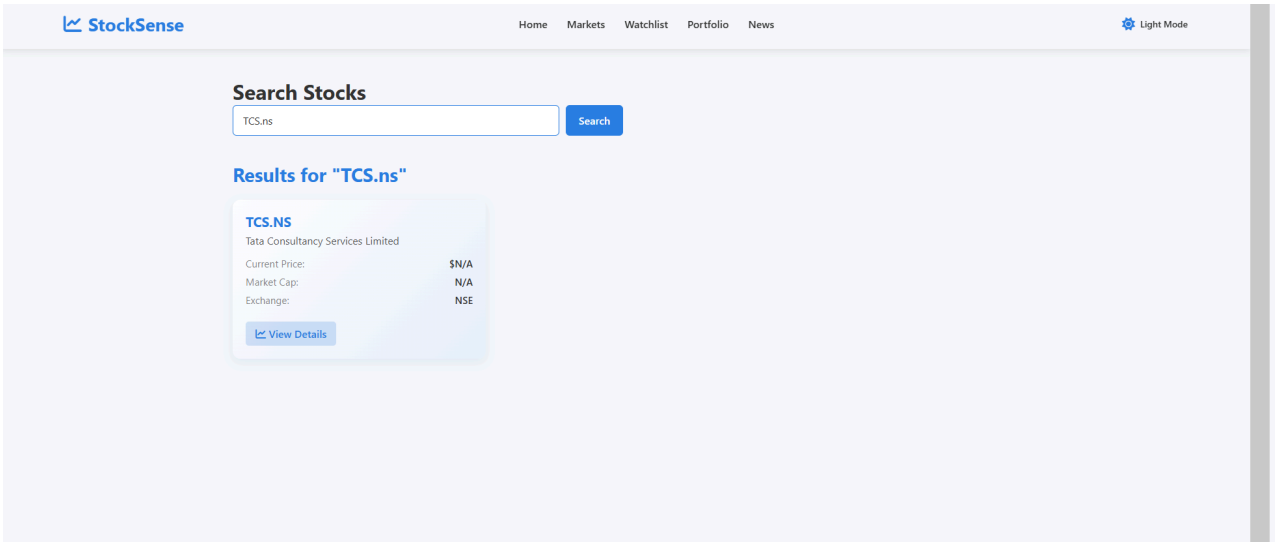
Stocks :-



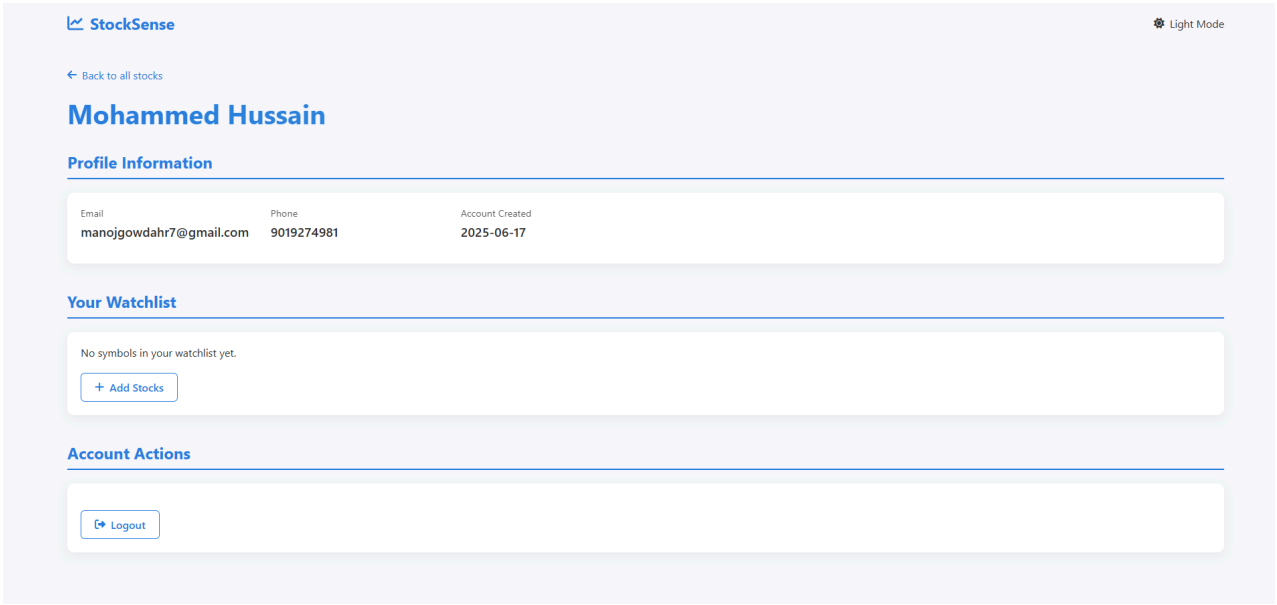
Stock Details :-

Output ScreenShots

Search Stock:-

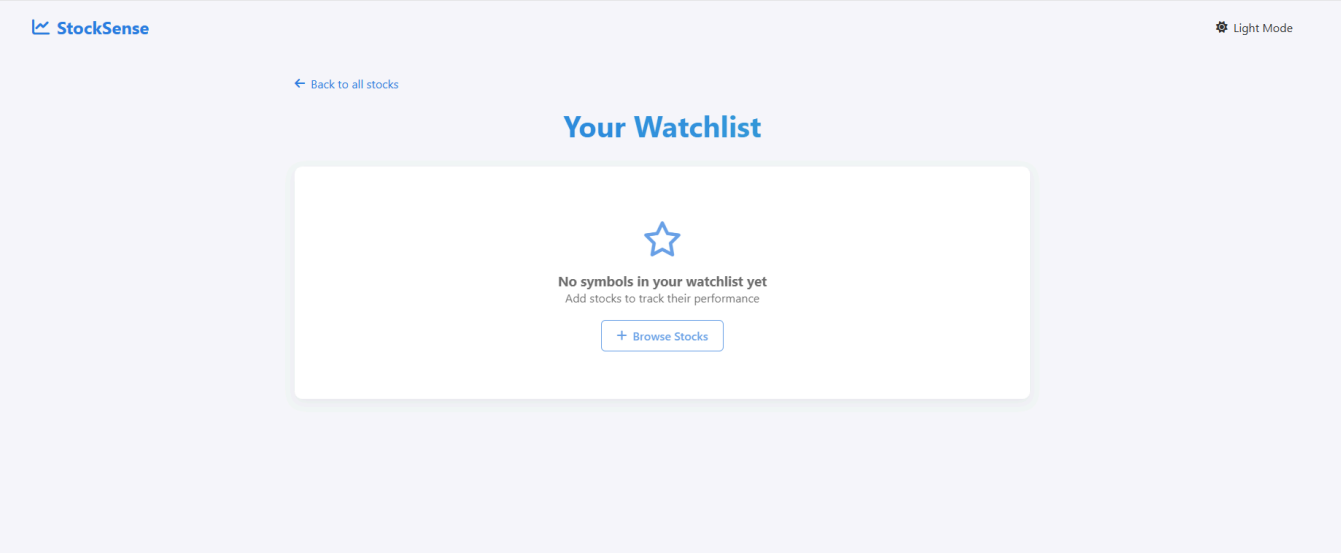


Profile :-



Output ScreenShots

Watchlist :-



Chapter10 :Testing

Testing ensures that the Stock Sense system works correctly and meets its requirements. Various types of testing were performed to check each part of the system.

1. Unit Testing

Each function was tested separately:

- Fetching stock data using Yahoo Finance module
- Saving stock details in MongoDB
- Adding stocks to user watchlists

2. Integration Testing

- Checked how well the modules worked together (database + backend + stock fetch).
- Ensured smooth data flow between fetching, saving, and displaying stock data.

3. Database Testing

- Verified that email and stock symbols were stored properly in MongoDB collections.
- Checked for duplicate entries and data consistency.

4. Functional Testing

- Tested main system features like:
- Real-time stock fetching
- Adding to watchlist
- Viewing watchlist

5. Manual Testing

- Performed by manually running the application, checking console outputs, database entries, and functionality.

6. Error Handling Tests

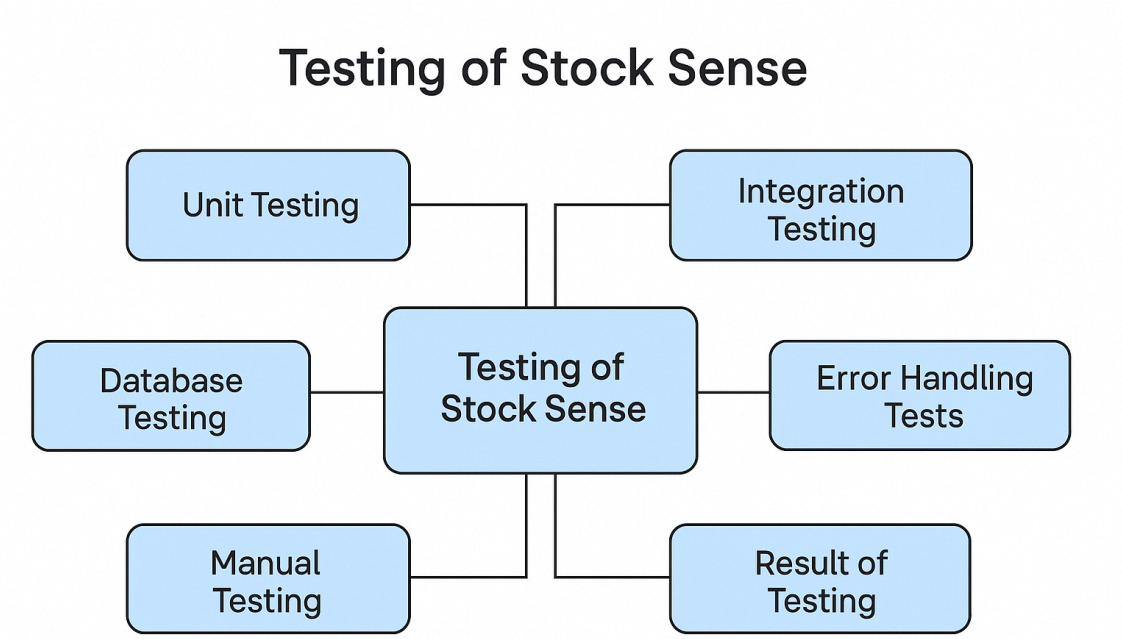
- Tested how the system handled:
- Invalid stock symbols
- Missing or incorrect inputs

7. Result of Testing

- All modules worked properly.
- No major bugs found.
- Ready for future improvements like user authentication or email alerts.

Testing

Diagram :-



Chapter11:Conclusion and Future Implementation

Conclusion

The Stock Sense project is designed to provide users with an easy and effective way to track stock market data in real time. It successfully integrates MongoDB for data storage and uses the Yahoo Finance module for fetching accurate stock information. The system focuses on simplicity, making stock tracking accessible even for beginners.

Benefits & Advantages:

- Real-Time Updates: Users get up-to-date stock prices.
- User-Friendly Interface: Easy for beginners and regular users.
- Personalized Watchlist: Allows users to track stocks of their interest.
- Secure Data Handling: User data like email and stock selections are safely stored.
- No External API Dependency: Uses Yahoo Finance module internally, reducing failures.

Disadvantages:

- Limited Features in Current Version: No advanced analytics or recommendation features yet.
- No Mobile Version: Currently web-only, mobile app can be added in future.
- No Email Alerts Yet: Email notifications feature can be developed as an enhancement.

Overall Summary:

Stock Sense meets its primary goal of providing a clean and reliable stock tracking system. With a solid structure, it is ready for future expansions, like adding advanced features, improving design, and providing more customization options for users.

Future Implementation

The Stock Sense project has laid a strong foundation for real-time stock tracking, but there is significant scope for future development and enhancement to make the platform more powerful, user-friendly, and feature-rich. Some of the major future implementations are:

1. **Advanced Analytics and Visualization**
   - Integration of advanced data visualization tools such as interactive charts, graphs, and trend lines to help users make better decisions based on historical data and technical analysis.
2. **Machine Learning-Based Predictions**
   - Implementation of predictive models using machine learning algorithms to forecast stock prices, trends, and market movements, providing users with data-driven insights.
3. **Mobile Application Development**
   - Designing and launching a dedicated mobile application (Android & iOS) to provide on-the-go access to stock tracking features, ensuring a wider reach and better accessibility.
4. **Email and Push Notifications**
   - Adding a notification system to alert users about important stock updates, price changes, and personal watchlist triggers via email or mobile notifications.
5. **User Portfolio Management**
   - Introducing a feature that allows users to create and manage their personal stock portfolios, calculate profits or losses, and receive personalized insights based on their investments.
6. **Real-Time News Integration**
   - Incorporating live financial news feeds related to the stock market to provide users with up-to-date market trends and happenings.
7. **Multi-language Support**
   - Adding support for regional and international languages to make the application accessible to a diverse range of users.

Chapter12 :References

References

1. Yahoo Finance Module
2. Used for fetching real-time stock market data for the application.
3. MongoDB Documentation
4. Referred for database setup, schema design, and data handling procedures.
5. Node.js Documentation
6. Helped in understanding backend development, server creation, and API handling.
7. Express.js Documentation
8. Referred for setting up routing and middleware for the web application.
9. JavaScript (ECMAScript) Guide
10. Used for enhancing frontend interactivity and client-side operations.
11. W3Schools
12. For basic web development concepts like HTML, CSS, and JavaScript usage.
13. Stack Overflow
14. Utilized for troubleshooting errors and optimizing certain code snippets.
15. MDN Web Docs
16. Referenced for advanced JavaScript, CSS styling, and frontend development practices.
17. GitHub Repositories (Community Projects)
18. Explored for understanding project structure and stock market-related implementations.