Table of Contents:

BLACKBUCK ENGINEERS

1.Abstract

Title: Stock Market Portfolio App Using MERN Stack

Introduction: The "Stock Market Portfolio App Using MERN Stack" is a web application designed to streamline the management and tracking of stock market investments. Utilizing the MERN stack—MongoDB, Express.js, React, and Node.js—this app enables users to efficiently view stock data, manage portfolios, and maintain a watchlist of favourite stocks.

Problem Statement and Overview: Current portfolio management tools often lack integration, real-time updates, and user-friendly interfaces. This project addresses these gaps by providing a comprehensive platform that consolidates stock information, offers real-time tracking, and allows users to manage a personal watchlist.

Tools and Applications Used:

- **MongoDB** for database management.
- **Express.js** for server-side logic and API management.
- **React** for building a dynamic and responsive user interface.
- **Node.js** for server operations and handling data interactions.

Existing System and Proposed Plan and Architecture: Existing systems may offer limited features and data integration. This project introduces an integrated solution with:

1. **Frontend:** React-based interface with routes for viewing all stocks and managing a watchlist.
2. **Backend:** Node.js and Express.js to handle API requests and data management.
3. **Database:** MongoDB for storing stock and user data.

Design or Flow:

1. **Backend Setup:** Create Express.js server, define schemas, and set up API endpoints.
2. **Frontend Development:** Implement React components for stock displays and watchlist management.
3. **Integration:** Ensure seamless data flow between frontend and backend.
4. **Testing and Deployment:** Test functionalities and deploy the application to a cloud platform.

Conclusion or Expected Output: The project will deliver a fully functional web application for stock portfolio management, featuring real-time data tracking and an intuitive interface. The application aims to enhance user experience in managing investments and making informed decisions.

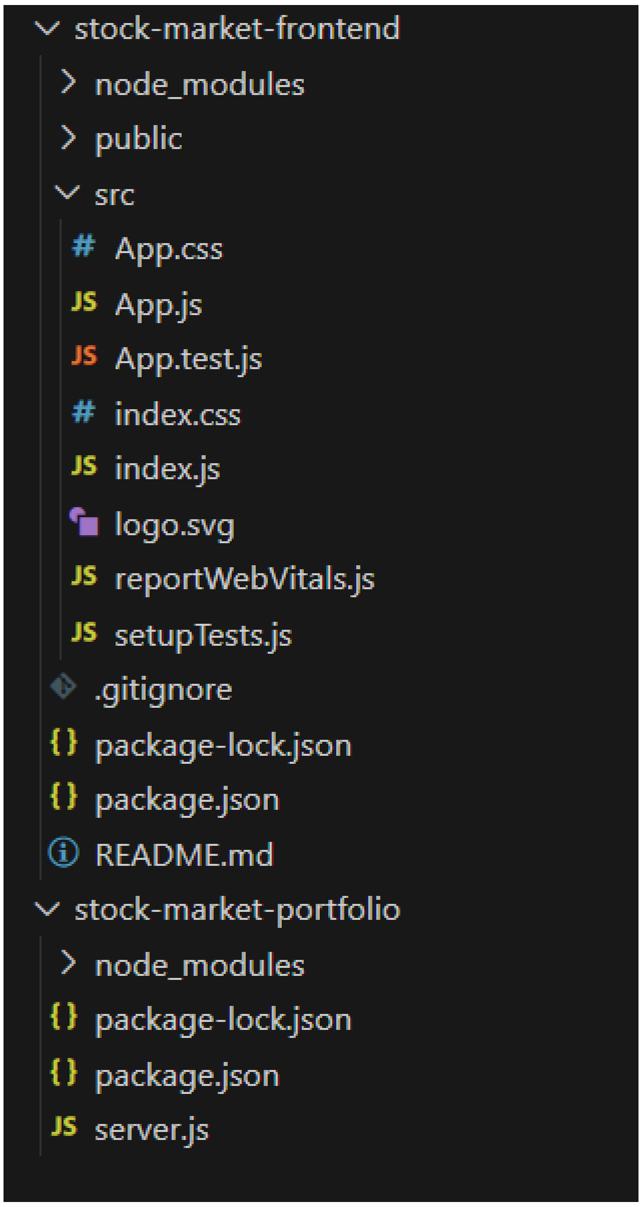
2. Project Overview

The Stock Market Portfolio project is a web application designed to efficiently manage and track stock market investments and portfolios using the MERN stack. The application provides features to view all stocks, add stocks to a watchlist, and color-code stock prices based on their change.

3. Prerequisites

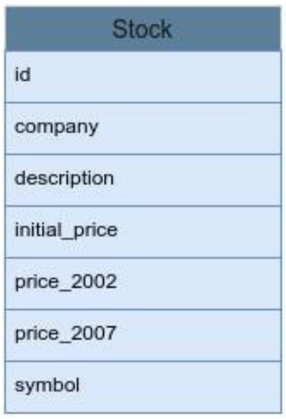
- **Node.js and npm**: Ensure Node.js and npm are installed on your system.
- **MongoDB**: Set up MongoDB, either locally or using MongoDB Atlas.
- **React**: Use React for frontend development.
- **Express**: Use Express for backend development.

4.Folder Structure:



```
stock-market-frontend
  node_modules
  public
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
stock-market-portfolio
  node_modules
  package-lock.json
  package.json
  server.js
```

5.ER Diagram:



1. Collection Overview

Collection Name: Stock

Purpose:

- **Data Retrieval**: This collection stores all stock-related information which is retrieved and displayed to users.
- **Watchlist Management**: It also manages user watchlists by storing stocks that users want to keep track of. There is no separate Watchlist collection; instead, the watchlist functionality is integrated into the Stock collection.

2. Data Fields

- **company**: Represents the name of the company associated with the stock (e.g., "Apple Inc.").
- **description**: Provides a brief description of the company or the stock (e.g., "Technology company specializing in consumer electronics").
- **initial_price**: Records the initial price of the stock when it was first listed (e.g., $50).
- **price_2002**: Contains the price of the stock in the year 2002 (e.g., $25).
- **price_2007**: Contains the price of the stock in the year 2007 (e.g., $40).
- **symbol**: A unique identifier or ticker symbol for the stock (e.g., "AAPL" for Apple Inc.).

3. Functionality

Retrieval of Stock Data:

- **GET Requests**: The application retrieves stock data from the Stock collection using GET requests to display stock information on the frontend.

Adding to Watchlist:

- **POST Requests**: Users can add stocks to their watchlist by making POST requests to the backend. The same Stock collection is used to handle these requests.
  - **Adding New Stocks**: When a new stock is added to the watchlist, it is inserted as a new document into the Stock collection.

- **Updating Existing Stocks**: If the stock already exists, it can be updated with new information or marked as part of the user's watchlist.

4. Design Rationale

Unified Collection:

- **Simplicity**: Using a single collection for both stock data and watchlist management reduces complexity in the schema and simplifies data access and manipulation.
- **Efficiency**: This design eliminates the need for joins or multiple queries to manage and retrieve watchlist items, improving performance and maintainability.

5. Example Document

Here's an example of how a stock document might look in the Stock collection:

json

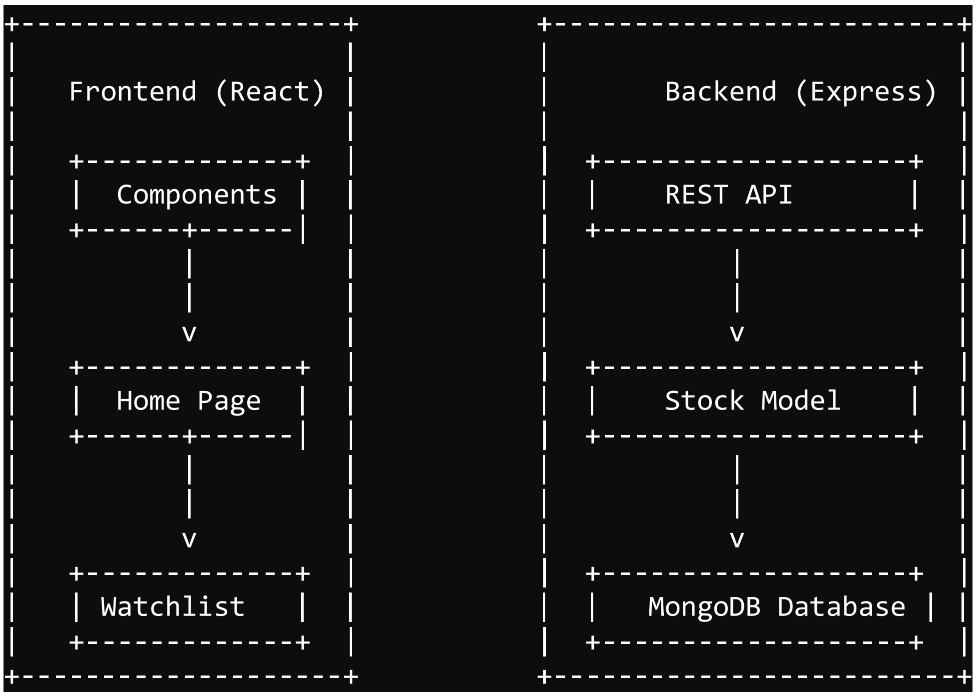
```json
{
  "_id": "ObjectId('...')",
  "company": "Apple Inc.",
  "description": "Technology company specializing in consumer electronics.",
  "initial_price": 50,
  "price_2002": 25,
  "price_2007": 40,
  "symbol": "AAPL"
}
```

6. Benefits

- **Ease of Use**: Users can view and manage their watchlist using the same interface as for viewing all stocks.
- **Streamlined Operations**: Backend operations are streamlined by handling all stock-related requests within a single collection.

6.Architecture Diagram

The architecture diagram provides a visual representation of the Stock Market Portfolio application's components and their interactions. Here’s a detailed breakdown of each element in the diagram:



```
+--------------------+          +-------------------------+
|                    |          |                         |
|   Frontend (React) |          |       Backend (Express) |
|                    |          |                         |
|   +-------------+  |          |  +-------------------+  |
|   |  Components |  |          |  |    REST API       |  |
|   +------+------|  |          |  +-------------------+  |
|          |         |          |           |             |
|          |         |          |           |             |
|          v         |          |           v             |
|   +-------------+  |          |  +-------------------+  |
|   |  Home Page  |  |          |  |    Stock Model    |  |
|   +------+------|  |          |  +-------------------+  |
|          |         |          |           |             |
|          |         |          |           |             |
|          v         |          |           v             |
|   +-------------+  |          |  +-------------------+  |
|   | Watchlist   |  |          |  |   MongoDB Database |  |
|   +-------------+  |          |  +-------------------+  |
+--------------------+          +-------------------------+
```

- **Frontend (React)**: Manages the user interface and communicates with the backend via API requests.
- **Backend (Express)**: Processes API requests, interacts with the database, and sends responses.
- **Database (MongoDB)**: Stores and manages the application's data.

This architecture ensures a clear separation of concerns, where the frontend handles user interactions, the backend manages business logic and data processing, and the database stores persistent data.

1. Frontend (React)

Components:

- **Components**: This represents the various React components that make up the user interface. Components are modular and reusable pieces of the UI, such as buttons, lists, and forms.
  - **Home Page**: A component that likely serves as the main landing page or dashboard of the application.
  - **Watchlist**: A component that displays the stocks a user has added to their watchlist.

Flow:

- The frontend communicates with the backend to fetch data and update the user interface based on user actions.

2. Backend (Express)

REST API:

- **REST API**: Represents the set of endpoints exposed by the backend to handle requests from the frontend. These include:
    - **GET /api/stocks**: Retrieves a list of stocks from the database.
    - **POST /api/watchlist**: Adds a new stock to the watchlist in the database.

Stock Model:

- **Stock Model**: Defines the schema for stock documents in the MongoDB database. It specifies the structure of stock data, including fields like company, description, initial_price, and symbol.

Flow:

- The backend processes requests from the frontend, interacts with the database to perform CRUD operations, and sends responses back to the frontend.

3. Database (MongoDB)

MongoDB Database:

- **MongoDB Database**: Stores the application's data. In this case, it contains:
    - **Stock Collection**: Holds documents with information about each stock, such as company details, prices, and symbols.

Flow:

- The backend interacts with MongoDB to store and retrieve stock data based on API requests. MongoDB handles the data persistence and querying.

Visual Representation of Data Flow

1. **Frontend Interaction**:
    - The **React frontend** requests data from the **Express backend** via API endpoints.
    - The **Stocks component** fetches stock data from the /api/stocks endpoint and displays it to users.
    - When a user adds a stock to their watchlist, the **Watchlist component** makes a **POST** request to /api/watchlist to save the stock data.
2. **Backend Processing**:
    - The **Express backend** receives requests from the frontend and interacts with the **MongoDB database** to fetch or store data.
    - The backend processes **GET** requests to retrieve stock information and **POST** requests to add stocks to the watchlist.
3. **Database Operations**:
    - The **MongoDB database** stores the stock data and watchlist entries.

- The backend uses Mongoose to define the **stock model** and perform database operations, ensuring data is saved and retrieved correctly.

7.Frontend

Step 1: Initialize Frontend

1. Create the React project:

```
npx create-react-app stock-market-frontend
cd stock-market-frontend
```

2. Install dependencies:

```
npm install axios
```

Step 2: Frontend Code

Create or update src/App.js with the following code:

javascript

```
import React, { useState, useEffect } from "react";
import { BrowserRouter as Router, Routes, Route, NavLink } from "react-router-dom";
import "./App.css";
const Stocks = ({ addToWatchlist }) => {
  const [stocks, setStocks] = useState([]);
  useEffect(() => {
    fetch("http://localhost:4000/api/stocks")
      .then((res) => res.json())
      .then((data) => setStocks(data))
      .catch((error) => console.error("Error fetching stocks:", error));
  }, []);
  const getRandomColor = () => {
    const colors = ["#FF0000", "#00FF00"];
    return colors[Math.floor(Math.random() * colors.length)];
  };
  return (
    <div className="App">
      <h1>Stock Market MERN App</h1>
      <h2>Stocks</h2>
```

```
            <ul>
                {stocks.map((stock) => (
                    <li key={stock.symbol}>
                        {stock.company} ({stock.symbol}) -
                        <span style={{ color: getRandomColor() }}>
                            {" "}
                            ${stock.initial_price}
                        </span>
                        <button onClick={() => addToWatchlist(stock)}>
                            Add to My Watchlist
                        </button>
                    </li>
                ))}
            </ul>
        </div>
    );
};
const Watchlist = ({ watchlist }) => {
    const getRandomColor = () => {
        const colors = ["#FF0000", "#00FF00"];
        return colors[Math.floor(Math.random() * colors.length)];
    };
    return (
        <div className="App">
            <h1>Stock Market MERN App</h1>
            <h2>My Watchlist</h2>
            <ul>
                {watchlist.map((stock) => (
                    <li key={stock.symbol}>
                        {stock.company} ({stock.symbol}) -
                        <span style={{ color: getRandomColor() }}>
```

```
              {" "}
              ${stock.initial_price}
            </span>
          </li>
        ))}
      </ul>
    </div>
  );
};
function App() {
  const [watchlist, setWatchlist] = useState([]);
  const addToWatchlist = (stock) => {
    fetch("http://localhost:4000/api/watchlist", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(stock),
    })
      .then((res) => res.json())
      .then((data) => {
        alert(data.message);
        setWatchlist([...watchlist, stock]);
      })
      .catch((error) => console.error("Error adding to watchlist:", error));
  };
  return (
    <Router>
      <nav>
        <NavLink to="/stocks">Stocks</NavLink>
        <NavLink to="/watchlist">Watchlist</NavLink>
      </nav>
      <Routes>
        <Route path="/stocks" element={<Stocks addToWatchlist={addToWatchlist} />} />
        <Route path="/watchlist" element={<Watchlist watchlist={watchlist} />} />
      </Routes>
    </Router>
  );
}
export default App;
```

Create or update src/App.css with the following styles:

CSS

```
body {
  font-family: 'Arial', sans-serif;
  background-color: #d9d7ca;
  margin: 0;
  padding: 0;
}
.App {
  text-align: center;
  padding: 20px;
}
h1 {
  color: #1f454d;
}
h2 {
  color: #3c8d93;
  margin-top: 30px;
}
ul {
  list-style-type: none;
```

```
    padding: 0;
}
li {
    background-color: #3c8d93;
    color: #d9d7ca;
    padding: 10px;
    margin: 10px 0;
    border-radius: 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}
button {
    background-color: #1f454d;
    color: #d9d7ca;
    border: none;
    padding: 8px;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}
button:hover {
    background-color: #3c8d93;
}
nav {
    background-color: #1f454d;
    padding: 15px 0;
}
nav a {
    color: #d9d7ca;
    text-decoration: none;
    margin: 0 20px;
    font-size: 18px;
    transition: color 0.3s ease;
}
nav a:hover {
    color: #3c8d93;
}
```

8.Backend

Step 1: Initialize Backend

1. Create a project folder and initialize it:

```
mkdir stock-market-portfolio
cd stock-market-portfolio
npm init -y
```

2. Install dependencies:

```
npm install express mongoose body-parser cors
```

Step 2: Backend Code

Create server.js and add the following code:

javascript

```javascript
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const bodyParser = require("body-parser");
const app = express();
const PORT = process.env.PORT || 4000;
app.use(cors());
app.use(bodyParser.json());
mongoose.connect("mongodb://localhost:27017/Stock_Market", {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});
const stockSchema = new mongoose.Schema({
    company: String,
    description: String,
    initial_price: Number,
    price_2002: Number,
    price_2007: Number,
    symbol: String,
});
const Stock = mongoose.model("Stock", stockSchema);
app.get("/api/stocks", async (req, res) => {
    try {
        const stocks = await Stock.find();
        res.json(stocks);
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});
app.post("/api/watchlist", async (req, res) => {
    try {
        const {
            company,
            description,
            initial_price,
            price_2002,
            price_2007,
            symbol,
        } = req.body;
        const stock = new Stock({
            company,
            description,
            initial_price,
            price_2002,
            price_2007,
            symbol,
        });
        await stock.save();
        res.json({ message: "Stock added to watchlist successfully" });
    } catch (error) {
        console.error(error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Step 3: Database Setup

1. Open MongoDB Atlas or Compass and insert the provided JSON file into the stocks collection.

MongoDB Queries

Retrieve Data (GET):

- **Endpoint:** GET /api/stocks
- **Description:** Fetches a list of all stocks from the Stock collection.

Postman Request:

  - Method: GET
  - URL: http://localhost:4000/api/stocks
  - Headers: None needed
  - Body: Not applicable for GET requests

Create Data (POST):

- **Endpoint:** POST /api/watchlist
- **Description:** Adds a new stock to the Stock collection, which acts as the watchlist.

Request Body:

json

```
{
  "company": "Apple",
  "description":           "Technology
  company","initial_price": 150.25,
 "price_2002": 120.30,
  "price_2007":
  175.50,  "symbol":
  "AAPL"
}
```

Postman Request:

- Method: POST
- URL: http://localhost:4000/api/watchlist
- Headers: Content-Type: application/json
- Body: Raw JSON data as shown above

Data Storage and Retrieval

Data Stored:

The database stores detailed information about various stocks, which is crucial for users to manage and track their stock investments. Each stock entry in the database includes the following data:

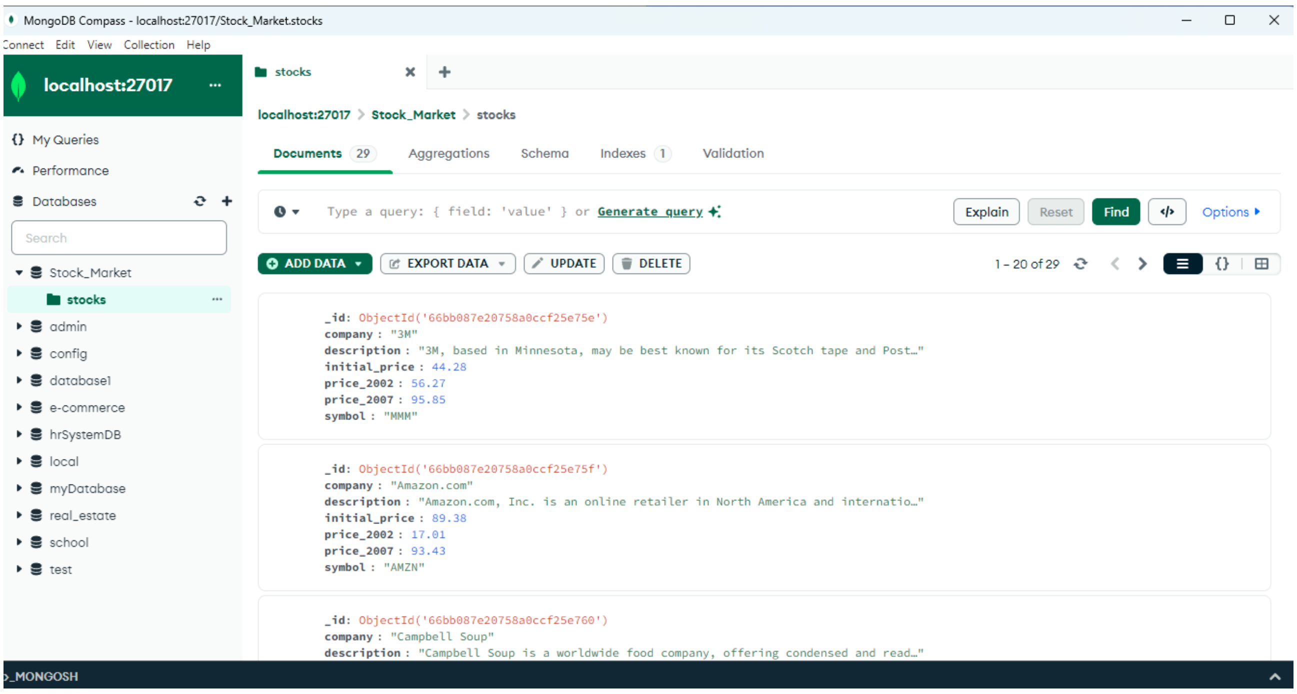
- **_id:** A unique identifier (ObjectId) assigned to each stock entry. This field is automatically generated by MongoDB.
- **company:** The name of the company associated with the stock.
- **description:** A brief description of the company's operations or the stock itself.
- **initial_price:** The price of the stock at the time of its initial listing.
- **price_2002:** The stock price recorded in the year 2002.
- **price_2007:** The stock price recorded in the year 2007.
- **symbol:** The stock's ticker symbol used for trading.

Data Entries:

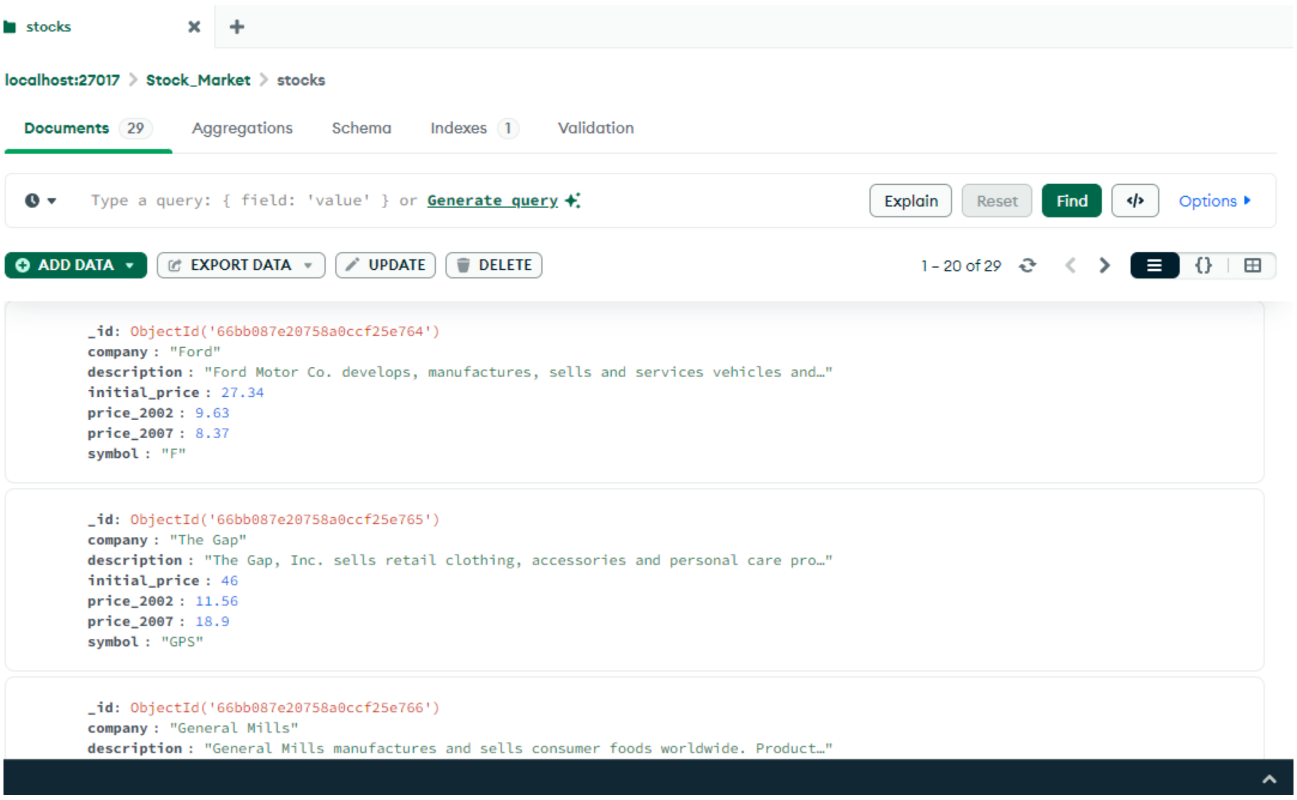
1. **Apple Inc.**
   - **company:** Apple Inc.
   - **description**: Technology company known for its iPhones, iPads, and Mac computers.
   - **initial price:** 150.25
   - **price_2002:** 120.30
   - **price_2007:** 175.50
   - **symbol:** AAPL

MongoDB Compass:

Stocks Collection: The stocks data has stored in the stocks collection.



```
_id: ObjectId('66bb087e20758a0ccf25e75e')
company : "3M"
description : "3M, based in Minnesota, may be best known for its Scotch tape and Post…"
initial_price : 44.28
price_2002 : 56.27
price_2007 : 95.85
symbol : "MMM"

_id: ObjectId('66bb087e20758a0ccf25e75f')
company : "Amazon.com"
description : "Amazon.com, Inc. is an online retailer in North America and internatio…"
initial_price : 89.38
price_2002 : 17.01
price_2007 : 93.43
symbol : "AMZN"

_id: ObjectId('66bb087e20758a0ccf25e760')
company : "Campbell Soup"
description : "Campbell Soup is a worldwide food company, offering condensed and read…"
```



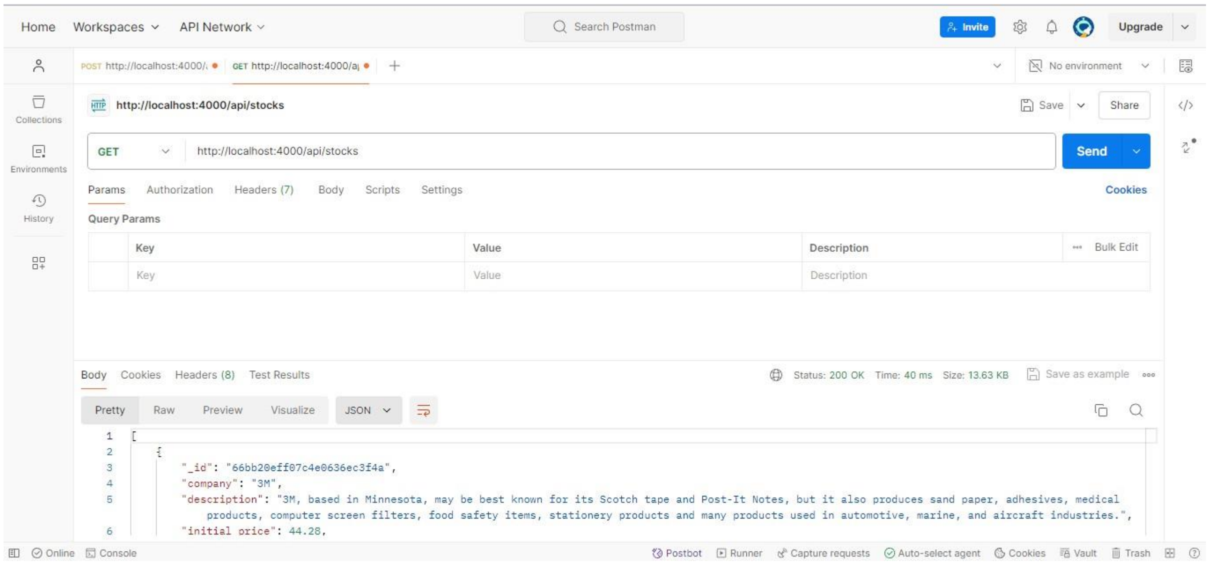
```
_id: ObjectId('66bb087e20758a0ccf25e764')
company : "Ford"
description : "Ford Motor Co. develops, manufactures, sells and services vehicles and…"
initial_price : 27.34
price_2002 : 9.63
price_2007 : 8.37
symbol : "F"

_id: ObjectId('66bb087e20758a0ccf25e765')
company : "The Gap"
description : "The Gap, Inc. sells retail clothing, accessories and personal care pro…"
initial_price : 46
price_2002 : 11.56
price_2007 : 18.9
symbol : "GPS"

_id: ObjectId('66bb087e20758a0ccf25e766')
company : "General Mills"
description : "General Mills manufactures and sells consumer foods worldwide. Product…"
```

Data Retrieval

The GET /api/stocks endpoint retrieves all stock entries from the database and displays them on the Stocks page. Users can then add any stock to their watchlist by clicking the "Add to My Watchlist" button. This action triggers a POST request to the /api/watchlist endpoint, which adds the selected stock to the database.

1. Retrieve Data (GET):

URL: http://localhost:4000/api/stocks


GET http://localhost:4000/api/stocks

Status: 200 OK Time: 40 ms Size: 13.63 KB

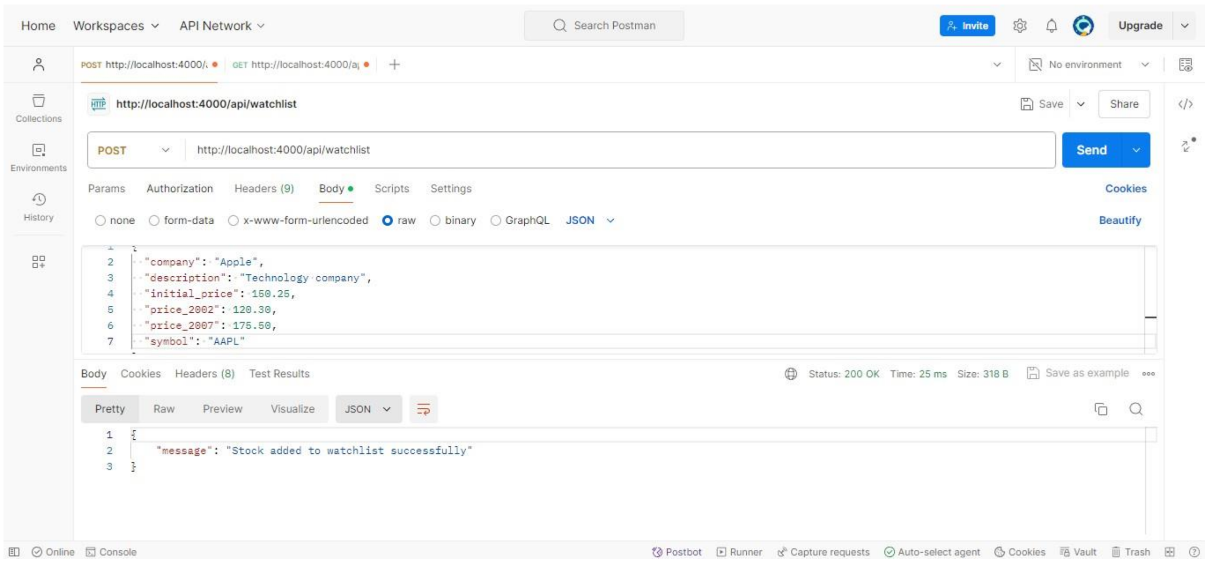
```
[
    {
        "_id": "66bb20eff07c4e0636ec3f4a",
        "company": "3M",
        "description": "3M, based in Minnesota, may be best known for its Scotch tape and Post-It Notes, but it also produces sand paper, adhesives, medical
            products, computer screen filters, food safety items, stationery products and many products used in automotive, marine, and aircraft industries.",
        "initial price": 44.28,
```

1. Create Data (POST)

URL: http://localhost:4000/api/watchlist

Inserting the data using Postman:

```
{
  "company": "Apple",
  "description": "Technology company",
  "initial_price": 150.25,
  "price_2002": 120.30,
  "price_2007": 175.50,
  "symbol": "AAPL"
}
```



POST http://localhost:4000/api/watchlist

Body: raw JSON

```
{
  "company": "Apple",
  "description": "Technology company",
  "initial_price": 150.25,
  "price_2002": 120.30,
  "price_2007": 175.50,
  "symbol": "AAPL"
```

Status: 200 OK Time: 25 ms Size: 318 B

```
{
    "message": "Stock added to watchlist successfully"
}
```

9.Tesing

1. Backend Testing

1.1. Test Stock Data Endpoint

Objective: Verify that the stock data endpoint returns a list of stocks.

1. **URL**: http://localhost:4000/api/stocks
2. **Method**: GET
3. **Expected Response**: A JSON array of stock objects.
4. **Steps**:
    - Open Postman.
    - Enter the URL and select the GET method.
    - Send the request and check that the response contains a list of stocks in JSON format.

1.2. Test Watchlist Endpoint

Objective: Ensure that stocks can be added to the watchlist successfully.

1. **URL**: http://localhost:4000/api/watchlist
2. **Method**: POST
3. **Request Body**:

json

{

  "_id": "66bb087e20758a0ccf25e765",

  "company": "The Gap",

```
  "description": "The Gap, Inc. sells retail clothing, accessories and personal care products.",
  "initial_price": 46,
  "price_2002": 11.56,
  "price_2007": 18.9,
  "symbol": "GPS"
}
```

4. **Expected Response**: A JSON object confirming that the stock has been added to the watchlist with a success message.
5. **Steps**:
   - Open Postman.
   - Enter the URL and select the POST method.
   - Provide the request body as shown above.
   - Send the request and check that the response includes a success message indicating the stock was added.

2. Frontend Testing

2.1. Verify Stock List

Objective: Confirm that the stock list is displayed correctly.

1. **URL**: http://localhost:3000/stocks
2. **Expected Outcome**: The stock data should be fetched from the backend and displayed on the page.
3. **Steps**:
   - Open a web browser.
   - Navigate to the URL.
   - Verify that the stock data is rendered as expected.

2.2. Test Add to Watchlist

Objective: Ensure that stocks can be added to the watchlist from the frontend.

1. **Action**: Click the "Add to My Watchlist" button for a stock.
2. **Expected Outcome**: The stock should be added to the watchlist, and a success message should appear.
3. **Steps**:
   - Open a web browser.

- Navigate to the stock list page.
- Click the "Add to My Watchlist" button for a stock.
- Verify that an alert with a success message appears.

2.3. Verify Watchlist

Objective: Confirm that the watchlist displays the stocks added.

1. **URL**: http://localhost:3000/watchlist
2. **Expected Outcome**: Stocks added to the watchlist should be displayed correctly.
3. **Steps**:
    - Open a web browser.
    - Navigate to the watchlist page.
    - Verify that the stocks are displayed as expected.

3. Integration Testing

3.1. End-to-End Integration

Objective: Ensure the complete workflow from fetching stocks to adding them to the watchlist works seamlessly.

1. **Steps**:
    - Perform the complete user flow:
        1. Fetch stocks from the backend and display them on the frontend.
        2. Add a stock to the watchlist.
        3. View the watchlist and confirm that the added stock is displayed.
    - Verify that there are no issues in communication between the frontend and backend and that the data is handled correctly throughout the process.

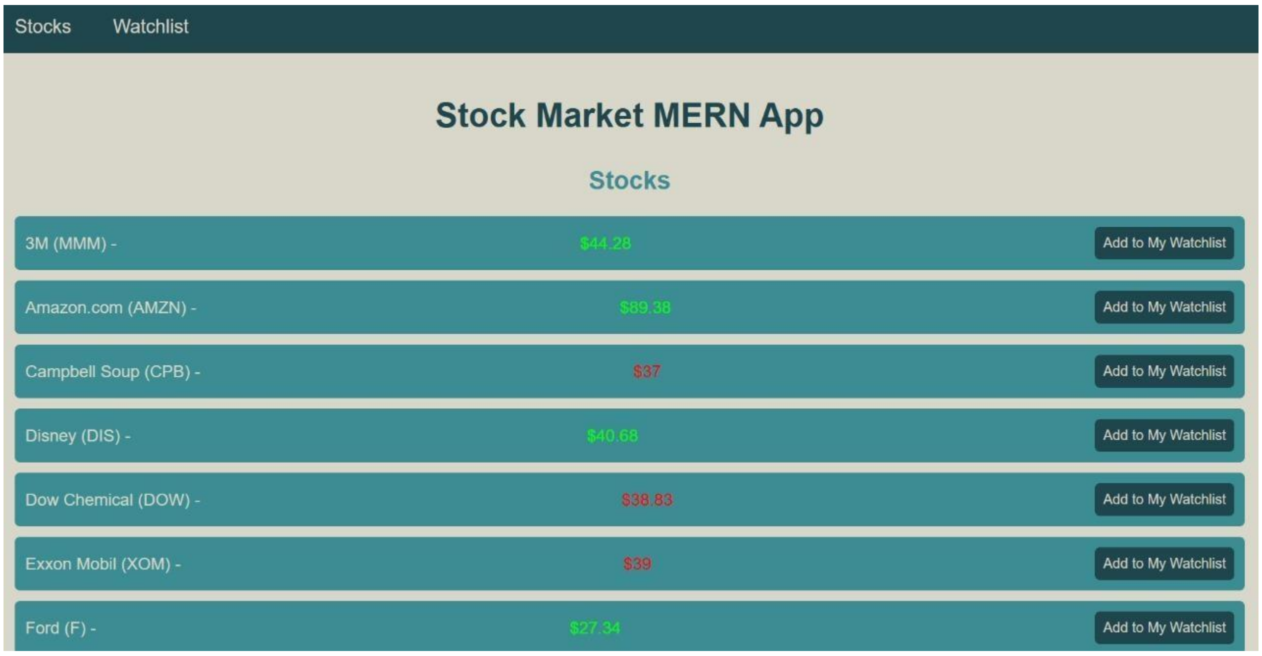
10. Appendix

GeeksforGeeks References

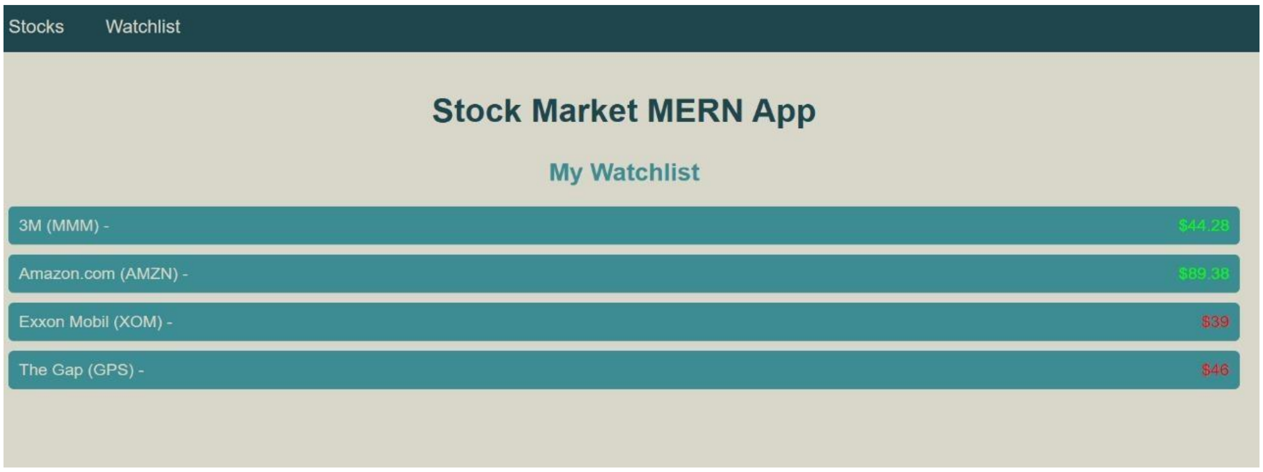
- GeeksforGeeks - Express.js Tutorial
- GeeksforGeeks - MongoDB Tutorial
- GeeksforGeeks - React.js Tutorial
- GeeksforGeeks - Node.js Tutorial

Final Web Page:

The web page integrates with the backend to fetch stock data and update the watchlist. The Stocks page allows users to view available stocks and add them to their watchlist. The Watchlist page displays the stocks that have been added by the user.



3M (MMM) - $44.28
Amazon.com (AMZN) - $89.38
Campbell Soup (CPB) - $37
Disney (DIS) - $40.68
Dow Chemical (DOW) - $38.83
Exxon Mobil (XOM) - $39
Ford (F) - $27.34



3M (MMM) - $44.28
Amazon.com (AMZN) - $89.38
Exxon Mobil (XOM) - $39
The Gap (GPS) - $46

10.Appendix

Reference Materials:

For the development and completion of the Stock Market Portfolio project, the primary resource used for reference and guidance was GeeksforGeeks. The platform provided valuable insights into various technical concepts, coding practices, and methodologies related to the project, including but not limited to:

- **JavaScript and React**: Understanding component structure, state management, and routing.

- **MongoDB and Mongoose**: Querying, filtering, and sorting data using MongoDB with Mongoose in the backend.
- **Express.js**: Setting up routes and handling HTTP requests efficiently.