Stock Market Portfolio Application

Frontend Description

The frontend of the application is built using React and provides a user interface for interacting with stock data and managing a watchlist. The user can view a list of stocks, add stocks to their watchlist, and navigate between different views.

1. **App.js**

```
// src/App.js

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 stock data from the backend
    fetch("http://localhost:4000/api/stocks")
      .then((res) => res.json())
      .then((data) => setStocks(data))
      .catch((error) => console.error("Error fetching stocks:", error));
  }, []);
  console.log(setStocks, "Stocksdata");

  const getRandomColor = () => {
    const colors = ["#FF0000", "#00FF00"]; // Red and Green
    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"]; // Red and Green
          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) => {
          // Add stock to watchlist
          fetch("http://localhost:4000/api/watchlist", {
            method: "POST",
            headers: {
```

```
            "Content-Type": "application/json",
          },
          body: JSON.stringify(stock),
        })
          .then((res) => res.json())
          .then((data) => {
            // Show an alert with the message received from the server
            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;
```

- **Purpose**: Acts as the main component that sets up routing and handles state management for the watchlist.
- **Components**:
    - **Stocks**: Displays a list of all stocks fetched from the backend.
    - **Watchlist**: Shows stocks that the user has added to their watchlist.

Process and Output:

1. **Routing**:
    - **<Router>**: Uses React Router to handle navigation.
    - **<Routes>** and **<Route>**: Define routes for /stocks and /watchlist paths.
2. **State Management**:
    - **useState**: Manages the watchlist state (watchlist).
    - **addToWatchlist Function**: Adds a stock to the watchlist by sending a POST request to the backend.
3. **Components**:
    - **Stocks Component**:
        - Fetches stock data from http://localhost:5000/api/stocks using fetch and useEffect.
        - Displays stocks with their company names, symbols, and initial prices.
        - Uses getRandomColor to dynamically color the price based on a random selection between red and green.
        - Provides an "Add to My Watchlist" button to add stocks to the watchlist.
    - **Watchlist Component**:
        - Displays stocks added to the watchlist.
        - Also uses getRandomColor to color the prices.

Expected Output:

- **Stocks Page**:
    - Displays a list of stocks with company names, symbols, and colored prices.
    - Each stock has a button to add it to the watchlist.
- **Watchlist Page**:
    - Shows the stocks that have been added to the user's watchlist with colored prices.

2. App.css

```
/* src/App.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;
}
```

```
/* Navigation bar styles */
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;
}
```

- **Purpose**: Provides styling for the application to ensure a cohesive and visually appealing user interface.

Styling Details:

- **General Styles**:
    - Sets font, background color, and margin for the body.
    - Styles .App to center text and add padding.
- **Text and Lists**:
    - Styles headers (h1 and h2) with colors and margins.
    - Styles the list (ul and li) with background colors, padding, and alignment.
- **Buttons and Navigation**:
    - Styles buttons with color, padding, and hover effects.
    - Styles navigation bar and links with colors and hover effects.

Backend Description

The backend of the application is built using Node.js with Express and MongoDB. It manages the stock data and handles requests to fetch stock information and update the watchlist.

1. **server.js**

```
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}`);
});
```

- **Purpose**: Sets up the Express server, connects to MongoDB, and defines API endpoints for stock data and watchlist management.

Process and Output:

1. **Server Setup**:
    - Initializes Express and configures middleware (cors and bodyParser).
    - Connects to MongoDB using Mongoose with connection settings.
2. **Schema Definition**:
    - **stockSchema**: Defines the schema for stock data with fields such as company, description, initial_price, price_2002, price_2007, and symbol.
3. **API Endpoints**:
    - **GET /api/stocks**:
        - Fetches all stock records from MongoDB.
        - Returns stock data as a JSON response.
        - **Output**: A JSON array of stock objects, each with fields defined in the schema.
    - **POST /api/watchlist**:
        - Accepts a stock object in the request body.
        - Saves the stock to MongoDB.
        - Returns a success message upon successful addition.

- **Output**: A JSON response with a message confirming the stock was added to the watchlist.

Expected Output:

- **Stock Data**: A list of stock objects in JSON format, available on the /api/stocks endpoint.
- **Watchlist Addition**: A confirmation message in JSON format when a stock is successfully added to the watchlist via the /api/watchlist endpoint.

Interaction Between Frontend and Backend

1. **Fetching Stocks**:
    - The Stocks component in the frontend makes a GET request to the /api/stocks endpoint to retrieve stock data.
    - The backend responds with a JSON array of stock objects.
2. **Adding to Watchlist**:
    - When a user clicks the "Add to My Watchlist" button, the addToWatchlist function sends a POST request to the /api/watchlist endpoint with the stock data.
    - The backend saves this data to MongoDB and responds with a success message.
3. **Displaying Data**:
    - The frontend displays the retrieved stock data and watchlist items, with prices dynamically colored and interactive elements for user actions.