## Table of Contents:

## 1. Overview:

The Stock Tracker web application allows users to monitor stock prices in real-time, view historical data, and set alerts for stock changes. It provides functionalities like adding stocks to watchlists, tracking the performance of stocks, and displaying key stock metrics.

---

## 2. High-Level Architecture:

- **Frontend (Client-side):** Flutter (Web)
  - A responsive web application built using Flutter, which interacts with the backend via REST APIs.
- **Backend (Server-side):** Spring Boot
  - A RESTful API service built using Spring Boot, which handles business logic, authentication, database interactions, and communicates with external stock data APIs.
- **Database:** MySQL
  - A relational database system to store user information, stock data, watchlist, and stock price history.
- **External APIs (Optional):**
  - **Stock Market API** (e.g., Alpha Vantage, Yahoo Finance, or IEX Cloud) to get real-time stock price data.

---

## 3. Technology Stack:

**Frontend:**

- **Framework:** Flutter Web
- **Languages:** Dart
- **State Management:** Provider / Riverpod / Bloc
- **Routing:** `fluro` or Flutter's native routing system
- **UI Frameworks:** Flutter Material Components
- **API Integration:** HTTP package (`http` or `dio`) to call Spring Boot APIs
- **Authentication:** Firebase Authentication or JWT-based authentication (depending on your choice)

**Backend:**

- **Framework:** Spring Boot
- **Programming Language:** Java
- **API Layer:** Spring Web (REST Controllers)
- **Database Access Layer:** Spring Data JPA or Hibernate
- **Security:** Spring Security (JWT Authentication)
- **Task Scheduling (for real-time updates):** Spring Scheduler / Quartz

- **API Communication:** REST (JSON over HTTP)
- **Stock Data Provider:** Integration with third-party stock market APIs (e.g., Alpha Vantage, IEX Cloud)

**Database:**

- **Database Engine:** MySQL 8.x
- **ORM Framework:** Hibernate / Spring Data JPA
- **Schema Design:**
  - **Users:** Table to store user information.
  - **Stocks:** Table to store stock metadata (symbol, name, etc.).
  - **Watchlist:** Table to store stocks users are watching.
  - **StockPrices:** Historical stock data table (date, stock symbol, opening price, closing price, etc.).
  - **Alerts:** User-created alerts for specific stock price thresholds.

---

## 4. Features and Functionalities:

**Frontend Features:**

1. **User Authentication:**
   - Sign-up and login functionality (via Firebase or JWT-based).
   - User sessions with JWT or token-based management.
2. **Dashboard:**
   - Overview of the current stock market data with real-time updates.
   - List of all stocks being tracked.
3. **Watchlist Management:**
   - Add/remove stocks to/from the watchlist.
   - Display stock prices in the watchlist, with options to sort by name or price.
4. **Real-Time Stock Price Tracking:**
   - Display current stock price (from external APIs).
   - Display stock price in charts (historical data can be shown with a line graph).
5. **Stock Details:**
   - Detailed stock information page with price history, company profile, etc.
6. **Responsive UI:**
   - Adaptive UI to work on both desktop and mobile screens.
7. **Search Functionality:**
   - Search for stocks by symbol or name.

**Backend Features:**

1. **User Authentication:**
   - JWT-based authentication or Firebase Authentication to protect endpoints.
2. **Stock Data API:**
   - Integration with third-party APIs (e.g., Alpha Vantage, IEX Cloud) to fetch real-time stock data.

3. **Stock Tracking:**
    - Store historical stock prices and metadata in the database.
    - Store user-specific watchlist data in the database.
4. **Alert Management:**
    - API to create, update, delete alerts.
    - Periodically check if any stocks have crossed the alert threshold.
5. **API Endpoints (RESTful):**
    - **User APIs**: Registration, login, and account management.
    - **Stock APIs**: Fetch stock data, search stocks, and list watchlist.
    - **Alert APIs**: Create, update, and delete price alerts.
    - **Watchlist APIs**: Manage stocks in the user's watchlist.
6. **Database Management:**
    - Use Spring Data JPA for database management and querying.
    - Automatic database schema generation with Hibernate.
7. **Security:**
    - Use Spring Security to ensure safe endpoints.
    - JWT-based security for session management.

---

## 5. API Documentation :

### Third-Party API's :

**-> Stocks (Twelve Data API ): -**

`https://api.twelvedata.com/stocks` endpoint.

#### 1. Overview

- **Description:** This endpoint retrieves a list of available stocks from Twelve Data.
- **Purpose:** To provide developers with a comprehensive list of stock symbols, names, and other metadata.
- **Base URL:** `https://api.twelvedata.com`
- **Endpoint:** `/stocks`
- **Method:** `GET`
- **Format:** JSON

#### 2. Authentication

- **API Key:** An API key is required for authentication. You must obtain an API key from Twelve Data (https://twelvedata.com/).
- **API Key Usage:** Include your API key as a query parameter named `apikey`.

#### 3. Endpoint Details

- **Endpoint:** `/stocks`

- **Method:** `GET`
- **Query Parameters:**
    - `apikey` (Required): Your Twelve Data API key.
    - `exchange` (Optional): Filter results by exchange code (e.g., `BSE`, `NSE`, `NASDAQ`, `NYSE`).
    - `country` (Optional): Filter results by country code (e.g., `IN`, `US`).
    - `type` (Optional): Filter results by asset type (e.g., `stock`).
    - `symbol` (Optional): Filter results by symbol.
    - `name` (Optional): Filter results by name.
    - `page` (Optional): Page number for pagination. Default is 1.
    - `per_page` (Optional): Number of results per page. Default is 50. Maximum is 500.

**4. Request Examples (URLs):**

- Retrieving all stocks:

```
Unset

https://api.twelvedata.com/stocks?apikey=YOUR_API_KEY
```

- Retrieving stocks from the NSE exchange:

```
Unset
https://api.twelvedata.com/stocks?apikey=YOUR_API_KEY&exchange=NSE
```

**5.. Response Structure**

- **Successful Response (HTTP 200 OK):**

```
Unset
{

    "data": [
        {
            "symbol": "ASHOKLEY.NSE",
            "name": "Ashok Leyland Ltd.",
            "exchange": "NSE",
            "country": "India",
            "type": "stock",
            "currency": "INR",
            "timezone": "Asia/Kolkata",
            "mic_code": "XNSE"
        },
        {
            "symbol": "TCS.NSE",
            "name": "Tata Consultancy Services Ltd.",
            "exchange": "NSE",
            "country": "India",
            "type": "stock",
            "currency": "INR",
            "timezone": "Asia/Kolkata",
            "mic_code": "XNSE"
        },
        // ... more stock data
    ],
    "meta": {
        "current_page": 1,
        "next_page": 2,
        "prev_page": null,
        "total_pages": 10,
        "per_page": 50,
        "total_items": 500
    },
    "status": "ok"
}
```

**6. Response Fields:**

- `data` (Array of Objects): An array containing stock information.
  - `symbol` (String): The stock symbol.
  - `name` (String): The name of the stock.
  - `exchange` (String): The exchange code.
  - `country` (String): The country code.
  - `type` (String): The asset type (e.g., "stock").
  - `currency` (String): The currency.
  - `timezone` (String): The timezone.
  - `mic_code` (String): Market Identifier Code.
- `meta` (Object): Metadata about the response.
  - `current_page` (Integer): The current page number.
  - `next_page` (Integer/null): The next page number (or null if no next page).
  - `prev_page` (Integer/null): The previous page number (or null if no previous page).
  - `total_pages` (Integer): The total number of pages.
  - `per_page` (Integer): The number of items per page.
  - `total_items` (Integer): The total number of items.
- `status` (String): The status of the request ("ok" or "error").

**7. Error Responses:**

- Twelve Data returns error messages within the JSON response if there are issues (e.g., invalid API key, rate limit exceeded, invalid parameters).
- The Json response will contain a status of "error", and a message string describing the error.

**8. Rate Limiting:**

- Twelve Data has rate limits. 800 credits per day.

---

## -> Individual_Stocks (Alpha Vantage)

### 1. Overview

- **Description:** This endpoint retrieves daily time series data for a specific stock symbol from the Bombay Stock Exchange (BSE).
- **Purpose:** To provide historical daily stock price data (open, high, low, close, volume).
- **Base URL:** `https://www.alphavantage.co/query`
- **Endpoint:** (Query Parameters)
- **Method:** `GET`
- **Format:** JSON

### 2. Authentication

- **API Key:** An API key is required for authentication. You provided: `5022J6M48MQT903A`.
- **Important:** While your API key is included in the example URL, it is crucial to understand that sharing API keys publicly is a security risk. In a real-world application, you should handle API keys securely.
- **How to Obtain an API Key:**
    - Visit the Alpha Vantage website (https://www.alphavantage.co/).
    - Sign up for a free API key.

### 3. Endpoint Details

- **Endpoint:** (Query Parameters)
- **Method:** `GET`
- **Query Parameters:**
    - `function` (Required): Specifies the API function. In this case, `TIME_SERIES_DAILY`.
    - `symbol` (Required): The stock symbol and exchange. In this case, `ASHOKLEY.BSE`.
    - `outputsize` (Optional): Specifies the amount of data returned. `full` returns the full time series; `compact` returns the latest 100 data points. In your example you provided `full`.
    - `apikey` (Required): Your Alpha Vantage API key. (Example: `5022J6M48MQT903A`)
- **Request Example (URL):**

```
Unset

https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol
=ASHOKLEY.BSE&outputsize=full&apikey=5022J6M48MQT903A
```

## 4. Response Structure

- **Successful Response (HTTP 200 OK)**

Java
```java
{
    "Meta Data": {
        "1. Information": "Daily Time Series with full history",
        "2. Symbol": "ASHOKLEY.BSE",
        "3. Last Refreshed": "2024-05-20",
        "4. Output Size": "Full size",
        "5. Time Zone": "US/Eastern"
    },
    "Time Series (Daily)": {
        "2024-05-20": {
            "1. open": "240.0000",
            "2. high": "245.0000",
            "3. low": "238.0000",
            "4. close": "243.0000",
            "5. volume": "1234567"
        },
        "2024-05-17": {
            "1. open": "235.0000",
            "2. high": "241.0000",
            "3. low": "234.0000",
            "4. close": "239.0000",
            "5. volume": "987654"
        },
        // ... more daily data
    }
}
```

- **Response Fields:**
  - `Meta Data` (Object): Metadata about the API response.
    - `1. Information` (String): Description of the API function.
    - `2. Symbol` (String): The stock symbol.
    - `3. Last Refreshed` (String): The date of the last data update.
    - `4. Output Size` (String): The output size (`full` or `compact`).
    - `5. Time Zone` (String): The time zone of the data.
  - `Time Series (Daily)` (Object): Daily time series data.
    - Each date (String, YYYY-MM-DD) is a key, and its value is an object with:
      - `1. open` (String): Open price.
      - `2. high` (String): High price.
      - `3. low` (String): Low price.
      - `4. close` (String): Close price.
      - `5. volume` (String): Volume.
- **Error Responses:**
  - Alpha Vantage returns error messages within the JSON response if there are issues (e.g., invalid API key, rate limit exceeded, invalid symbol).

**5. Examples**

**Retrieving Full Daily Data for ASHOKLEY.BSE:**

```
Unset

https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol
=ASHOKLEY.BSE&outputsize=full&apikey=YOUR_API_KEY
```

**Retrieving Compact Daily Data for ASHOKLEY.BSE:**

```
Unset

https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol
=ASHOKLEY.BSE&outputsize=compact&apikey=YOUR_API_KEY
```

**6. Rate Limiting**

- Alpha Vantage has rate limits. The free tier typically allows 5 API calls per minute and 500 API calls per day.
- Refer to the Alpha Vantage documentation for the most up-to-date rate limit information.

---

**-> MF Listing API: -**

**Base URL:**
`https://api.mfapi.in`

**Endpoint:**
`/mf`

## Description:

This endpoint provides a list of all available mutual fund schemes. It returns data on the mutual fund schemes such as their scheme code, name, and category.

**Request Format**
**HTTP Method:**

`GET`

**Request URL:**

```
https://api.mfapi.in/mf
```

**Example Request:**

```
GET https://api.mfapi.in/mf
```

**Response Format**

The API responds with a JSON object containing an array of mutual fund schemes.

**Sample Response:**

```
[
  {
    "scheme_code": "100312",
    "scheme_name": "Nippon India Small Cap Fund",
    "category": "Equity - Small Cap"
  },
  {
    "scheme_code": "100134",
    "scheme_name": "SBI Bluechip Fund",
    "category": "Equity - Large Cap"
  },
  {
    "scheme_code": "100119",
    "scheme_name": "HDFC Top 200 Fund",
    "category": "Equity - Large Cap"
  }
]
```

**Response Fields:**

Each item in the response array represents a mutual fund scheme and contains the following fields:

- **scheme_code**: (string) The unique identifier for the mutual fund scheme.
- **scheme_name**: (string) The name of the mutual fund scheme.
- **category**: (string) The category of the mutual fund (e.g., equity, debt, hybrid).

**Notes:**

- This endpoint does not require any authentication or API key.
- The response contains a list of mutual fund schemes. Depending on the total number of schemes available, pagination or filtering might be applied (if the API supports that in the future).

---

**-> Individual MFListing API :-**

**Base URL:**

```
Unset

https://api.mfapi.in
```

**Endpoint:**

```
Unset

/mf/{scheme_code}
```

**Description:**

This API provides details of the mutual fund with a given `scheme_code`. The data returned includes various metrics related to the mutual fund's performance, such as NAV (Net Asset Value), AUM (Assets Under Management), returns, and more.

**Request Format**

**HTTP Method:**

`GET`

**Request URL:**

```
Unset

https://api.mfapi.in/mf/{scheme_code}
```

- `{scheme_code}`: The unique code of the mutual fund scheme you want to query. For example, `100312`.

**Example Request:**

```
Unset

GET https://api.mfapi.in/mf/100312
```

**Response Format :**

The API returns the response in JSON format.

**Sample Response:**

```
Unset

{

  "scheme_code": "100312",

  "scheme_name": "Nippon India Small Cap Fund",

  "category": "Equity - Small Cap",

  "fund_manager": "Mr. Abhinav Bhushan",

  "aum": "4500.4 Cr",

  "nav": {

    "date": "2025-03-19",

    "value": "99.2"

  },

  "returns": {

    "1Y": "24.5%",

    "3Y": "56.7%",

    "5Y": "81.3%"
  },
  "expense_ratio": "1.2%",
  "launch_date": "2003-01-01"
}
```

**Response Fields:**

- **scheme_code**: (string) The unique identifier for the mutual fund scheme.
- **scheme_name**: (string) The name of the mutual fund scheme.
- **category**: (string) The category of the mutual fund (e.g., equity, debt, hybrid).
- **fund_manager**: (string) The name of the fund manager responsible for managing the fund.
- **aum**: (string) The assets under management of the mutual fund (e.g., "4500.4 Cr").
- **nav**:
  - **date**: (string) The date of the NAV value.
  - **value**: (float) The Net Asset Value (NAV) of the scheme on the given date.
- **returns**: (object) The performance returns over different periods.
  - **1Y**: (string) The 1-year return (e.g., "24.5%").
  - **3Y**: (string) The 3-year return (e.g., "56.7%").
  - **5Y**: (string) The 5-year return (e.g., "81.3%").
- **expense_ratio**: (string) The expense ratio of the mutual fund scheme (e.g., "1.2%").
- **launch_date**: (string) The launch date of the mutual fund.
-

**Notes:**

- The mutual fund scheme code (e.g., `100312`) must be a valid scheme code in the API's database.
- The API does not require an API key or authentication.

---

## Rest API's List :

1. Login API Endpoint Documentation:

### 1. Overview

- **Endpoint:** `/public/login`
- **Method:** `POST`

- **Description:** This endpoint is used to authenticate users by verifying their username and password. Upon successful authentication, the server may return a token or session identifier.

**2. Request**

- **URL:** `http://localhost:5545/public/login`
- **Headers:** `Content-Type: application/json` (Indicates that the request body is in JSON format)
- **Body:**
  - **Format:** JSON
  - **Schema:**

Unset
```
{
  "username": "string",
  "password": "string"
}
```

  - **Properties:**
    - `username` (string, required): The username of the user attempting to log in.
    - `password` (string, required): The password of the user attempting to log in.
- **Example Request (cURL):**

Unset
```
curl --location 'http://localhost:5545/public/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "username" : "kumar",
    "password": "kumar@1234"
}'
```

**3. Response**

- **Success Status Code:** `200 OK`
- **Error Status Codes:**

    - 400 Bad Request: The request body is invalid (e.g., missing fields, incorrect data types).
    - 401 Unauthorized: Authentication failed (e.g., incorrect username or password).
    - 500 Internal Server Error: An unexpected error occurred on the server.

---

## 2 . User Registration Endpoint API Documentation:

1. Overview :

- **Endpoint:** `/public/register-new`
- **Method:** `POST`
- **Description:** This endpoint is used to register a new user in the system. It creates a new user account based on the provided user details.

**2. Request :**

- **URL:** `http://localhost:5545/public/register-new`
- **Headers:**
    - `Content-Type: application/json` (Indicates that the request body is in JSON format)
- **Body:**
    - **Format:** JSON
    - **Schema:**
- **JSON**

```
Unset

{

              "username": "string",

              "full_name": "string",

              "email_id": "string",

              "password": "string"

            }
```

- **Properties:**

    - `username` (string, required): The desired username for the new user. This should be unique.
    - `full_name` (string, required): The full name of the new user.
    - `email_id` (string, required): The email address of the new user. This should be a valid email format and ideally unique.
    - `password` (string, required): The desired password for the new user.

    Example Request (cURL):

```
Unset
            curl --location 'http://localhost:5545/public/register-new'
            \

            --header 'Content-Type: application/json' \

            --data-raw '{

                "username" : "kumar",

                "full_name" : "Kumar S",

                "email_id": "kumars@gmail.com",

                "password" : "kumar@1234"

            }'
```

Response :

- Success Status Code: `201 Created` (Indicates that a resource was successfully created)

- Error Status Codes:
    - `400 Bad Request`: The request body is invalid (e.g., missing fields, incorrect data types) or contains validation errors (e.g., invalid email format).
    - `409 Conflict`: A user with the provided username or email already exists.
    - `500 Internal Server Error`: An unexpected error occurred on the server.
- Success Response Body:
    - Format: JSON (Often, a successful registration may return a simplified confirmation or user details)

```
Unset
            {

              "user_id": "integer",
              "username": "string",

              "email_id": "string",

              "message": "string"

            }
```

---

3. Stock Data Synchronization Endpoint  API Documentation:

**1. Overview :**

- **Endpoint:** `/api/stocks/sync`
- **Method:** `GET`
- **Description: This endpoint is used to synchronize stock data. It likely triggers a process on the server to fetch or update stock information from an external source or perform an internal data refresh.**

**2. Request :**

- **URL:** `http://localhost:5545/api/stocks/sync`

- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` **(Provides authentication credentials using Basic Authentication)**
- **Body:**
  - **None (GET requests typically do not have a request body)**
- **Example Request (cURL):**

```
Unset

curl --location 'http://localhost:5545/api/stocks/sync' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:**
  - `200 OK`**: Indicates that the synchronization process was initiated successfully. The response body might contain details about the process.**
  - `202 Accepted`**: Indicates that the request has been accepted for processing, but the synchronization process has not yet completed. This is appropriate if the synchronization is an asynchronous operation.**

- Error Status Codes:
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `500 Internal Server Error`: An unexpected error occurred on the server during the synchronization process.

---

4. Get All Stock Data Endpoint API Documentation:

**1. Overview**

- Endpoint: `/api/stocks/show-all`
- Method: `GET`
- Description: This endpoint retrieves a list of all stock data available in the system.

**2. Request**

- URL: `http://localhost:5545/api/stocks/show-all`
- Headers:
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- Body:
    - None (GET requests typically do not have a request body)
- Example Request (cURL):

```
Unset

curl --location 'http://localhost:5545/api/stocks/show-all' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- Success Status Code: `200 OK`

- Error Status Codes:

    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `500 Internal Server Error`: An unexpected error occurred on the server while retrieving the stock data.
- Success Response Body:
    - Format: JSON (Likely, and highly recommended for structured data)
    - Schema (Example - Adapt to your actual response):

```
Unset

[
  {
    "stock_id": "integer",
    "symbol": "string",
    "name": "string",
    "price": "number",
    "volume": "integer",
    "timestamp": "string"
    //  ...  other stock data fields
  },
```

---

5. Search Stock Data Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stocks/search`
- **Method:** `GET`
- **Description: This endpoint allows searching for stock data based on a query string. It returns a list of stocks that match the search criteria.**

**2. Request**

- **URL:** `http://localhost:5545/api/stocks/search`
- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` **(Provides authentication credentials using Basic Authentication)**
- **Query Parameters:**
  - `query` **(string, required): The search query string. This is used to filter the stock data (e.g., by symbol, name, or other relevant fields).**
- **Body:**
  - **None (GET requests typically do not have a request body)**
- **Example Request (cURL):**

```
Unset
curl --location 'http://localhost:5545/api/stocks/search?query=hcl' \

           --header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

3. Response

- Success Status Code: `200 OK`
- Error Status Codes:
  - `400 Bad Request`: The request is malformed (e.g., the `query` parameter is missing).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `500 Internal Server Error`: An unexpected error occurred on the server during the search.
- Success Response Body:
  - Format: JSON (Likely, and highly recommended for structured data)
  - Schema (Example - Adapt to your actual response):

```
Unset
           [

             {

               "stock_id": "integer",

               "symbol": "string",

               "name": "string",

               "price": "number",

               "volume": "integer",

               "timestamp": "string"

               //  ...  other stock data fields

             },
```

---

## 6. Fetch Latest Stock Data Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stocks/fetch-stock-data/latest`
- **Method:** `GET`
- **Description:** This endpoint retrieves the latest available stock data for a specific stock symbol.

**2. Request**

- **URL:** `http://localhost:5545/api/stocks/fetch-stock-data/latest`

- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - `symbol` (string, required): The ticker symbol of the stock for which to retrieve data (e.g., "ASHOKLEY").
- **Body:**
  - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location
'http://localhost:5545/api/stocks/fetch-stock-data/latest?symbol=ASHOKLEY'
\
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0' \
--data ''
```

**3. Response**

- **Success Status Code:** `200 OK`

- **Error Status Codes:**

  - `400 Bad Request`: The request is malformed (e.g., the `symbol` parameter is missing).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `404 Not Found`: Stock data for the specified symbol was not found.
  - `500 Internal Server Error`: An unexpected error occurred on the server while fetching the stock data.
- **Success Response Body:**

  - **Format:** JSON (Likely, and highly recommended for structured data)

  - **Schema (Example - Adapt to your actual response):**

```
Unset

{
  "stock_id": "integer",
  "symbol": "string",
  "name": "string",
  "price": "number",
  "open": "number",
  "high": "number",
  "low": "number",
  "close": "number",
  "volume": "integer",
  "timestamp": "string"
  //  ...  other stock data fields

            }
```

---

## 7. Fetch Historical Stock Data Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stocks/history`
- **Method:** `GET`
- **Description:** This endpoint retrieves historical stock data for a specific stock symbol within a specified date range.

**2. Request**

- **URL:** `http://localhost:5545/api/stocks/history`
- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - `symbol` (string, required): The ticker symbol of the stock for which to retrieve historical data (e.g., "INFY").
  - `startDate` (string, required): The start date of the historical data range. The format should be specified (e.g., YYYY-MM-DD).
  - `endDate` (string, required): The end date of the historical data range. The format should be specified (e.g., YYYY-MM-DD).
- **Body:**
  - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location
'http://localhost:5545/api/stocks/history?symbol=INFY&startDate=&endDate='
\
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:** `200 OK`

- **Error Status Codes:**
    - `400 Bad Request`: The request is malformed (e.g., missing `symbol`, `startDate`, or `endDate` parameters, invalid date format).
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `404 Not Found`: Stock data for the specified symbol or within the given date range was not found.
    - `500 Internal Server Error`: An unexpected error occurred on the server while fetching the historical stock data.
- **Success Response Body:**
    - **Format:** JSON (Likely, and highly recommended for structured data)
    - **Schema (Example - Adapt to your actual response):**

```
Unset
[
  {
    "date": "string",
    "open": "number",
    "high": "number",
    "low": "number",
    "close": "number",
    "volume": "integer",
    "adj_close": "number"
    // ... other historical data fields
  },
  {
    "date": "string",
    "open": "number",
    "high": "number",
    "low": "number",
    "close": "number",
    "volume": "integer",
    "adj_close": "number"
    // ... other historical data fields
  },
  // ... more historical data objects (one for each date)

]
```

---

## 8. Check Stock Existence Endpoint API Documentation:

### 1. Overview

- **Endpoint:** `/api/stocks/check`
- **Method:** `GET`
- **Description:** This endpoint checks if stock data exists for a given stock symbol. It's useful to verify if a stock is supported or if data is available before making more detailed requests.

**2. Request**

- **URL:** `http://localhost:5545/api/stocks/check`
- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - `symbol` (string, required): The ticker symbol of the stock to check (e.g., "HCLTECH").
- **Body:**
  - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset

curl --location 'http://localhost:5545/api/stocks/check?symbol=HCLTECH' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Codes:**
  - `200 OK`: Indicates that stock data *exists* for the provided symbol.
  - `204 No Content`: Indicates that stock data *does not exist* for the provided symbol.
- **Error Status Codes:**
  - `400 Bad Request`: The request is malformed (e.g., the `symbol` parameter is missing).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `500 Internal Server Error`: An unexpected error occurred on the server while checking for the stock data.
- **Success Response Body (200 OK):**
  - **Format:** JSON (Often a simple confirmation)
  - **Schema (Example - Adapt to your actual response):**

```
Unset
{

    "exists": "boolean",

    "message": "string"

}
```

---

## 9. Fetch Stock Data Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stocks/fetch-stock-data`
- **Method:** `GET`
- **Description: This endpoint retrieves stock data for a specific stock symbol. It's important to note that this endpoint name is very similar to `/api/stocks/fetch-stock-data/latest`,**

so clearly differentiate them in your documentation. This endpoint might return more comprehensive data than just the "latest" (e.g., a snapshot of various data points).

**2. Request**

- **URL:** `http://localhost:5545/api/stocks/fetch-stock-data`
- **Headers:**
  `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - `symbol` (string, required): The ticker symbol of the stock for which to retrieve data (e.g., "WIPRO").
- **Body:**
  - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location
'http://localhost:5545/api/stocks/fetch-stock-data?symbol=WIPRO' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:** `200 OK`

- **Error Status Codes:**

  - `400 Bad Request`: The request is malformed (e.g., the `symbol` parameter is missing).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `404 Not Found`: Stock data for the specified symbol was not found.
  - `500 Internal Server Error`: An unexpected error occurred on the server while fetching the stock data.
- **Success Response Body:**

  - **Format:** JSON (Likely, and highly recommended for structured data)

  - **Schema (Example - Adapt to your actual response):**

```
Unset
{

  "stock_id": "integer",

  "symbol": "string",

  "name": "string",

  "price": "number",

  "open": "number",
  "high": "number",

  "low": "number",

  "close": "number",

  "volume": "integer",

  "timestamp": "string",

  "market_cap": "number",

  "pe_ratio": "number"

  //    ...  other stock data fields (more than "latest" endpoint?)

            }
```

---

## 10.Create User Portfolio Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/stock-tracker/user/create-portfolio`
- **Method:** `POST`
- **Description:** This endpoint is used to create a new portfolio for a user. A portfolio is likely a collection of stocks or other financial instruments that the user wants to track.

**2. Request**

- **URL:** `http://localhost:5545/stock-tracker/user/create-portfolio`
- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - None (based on the cURL provided, but double-check if any are used)
- **Body:**
  - **Format:** (It's empty in the cURL, but this is likely incorrect. It *should* be JSON. You **MUST** verify the actual format.)

  - **Schema (Example - Adapt to your actual request body):**

```
Unset
{

    "portfolio_name": "string",

    "description": "string",

    "user_id": "integer"
    //   ...   other portfolio creation details
}
```

  - **Properties (Example - Adapt to your actual request body):**
    - `portfolio_name` (string, required): The name of the new portfolio.
    - `description` (string, optional): A description of the portfolio.
    - `user_id` (integer, required): The unique identifier of the user who owns the portfolio.
    - `...` (other fields): Include any other necessary fields for portfolio creation.
- **Example Request (cURL):**

  Bash

```
Unset
curl --location --request POST
'localhost:5545/stock-tracker/user/create-portfolio' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0' \

--data ''
```

**3. Response**

- **Success Status Code:**
  - `201 Created`: Indicates that the portfolio was successfully created. This is the most appropriate code for a resource creation action.
  - `200 OK`: Also possible, but less semantically correct than 201.
- **Error Status Codes:**
  - `400 Bad Request`: The request body is invalid (e.g., missing `portfolio_name`, incorrect data types).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `409 Conflict`: A portfolio with the same name might already exist for the user (if portfolio names must be unique).
  - `500 Internal Server Error`: An unexpected error occurred on the server during portfolio creation.
- **Success Response Body (Example - Adapt to your actual response):**
  - **Format:** JSON (Likely, and highly recommended)
  - **Schema (Example - 201 Created):**

```
Unset
{

    "portfolio_id": "integer",

    "portfolio_name": "string",

    "user_id": "integer",

    "created_at": "string",

    "message": "string" }
```

---

## 11. Get All Portfolio Holdings Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/portfolio-holdings/all-holdings`
- **Method:** `GET`
- **Description:** This endpoint retrieves a list of all holdings across all portfolios for the authenticated user. A "holding" likely represents a specific quantity of a stock or other asset within a portfolio.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/portfolio-holdings/all-holdings`

- **Headers:**
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
    - None (based on the cURL provided, but verify if any are used for filtering or pagination)
- **Body:**
    - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location
'localhost:5545/api/stock-tracker/user/porfolio-holdings/all-holdings' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:** `200 OK`

- **Error Status Codes:**
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.

    - `500 Internal Server Error`: An unexpected error occurred on the server while retrieving the portfolio holdings.
- **Success Response Body:**
    - **Format:** JSON (Likely, and highly recommended for structured data)
    - **Schema (Example - Adapt to your actual response):**

```
Unset
[

  {

    "holding_id": "integer",

    "portfolio_id": "integer",

    "stock_symbol": "string",

    "quantity": "integer",

    "purchase_date": "string",

    "purchase_price": "number"

    //    ...   other holding details

  },

  {

    "holding_id": "integer",

    "portfolio_id": "integer",

    "stock_symbol": "string",

    "quantity": "integer",

    "purchase_date": "string",

    "purchase_price": "number"

    //    ...   other holding details  },
```

---

## 12. Add Stock to Portfolio Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/portfolio-holdings/addStock`

- **Method:** `POST`
- **Description:** This endpoint is used to add a new stock holding to a user's portfolio.

**2. Request**

- **URL:** `http://192.168.207.153:5545/api/stock-tracker/user/portfolio-holdings/addStock`
- **Headers:**
  - `Content-Type: application/json` (Indicates that the request body is in JSON format)
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Body:**
  - **Format:** JSON
  - **Schema:**

```
Unset
{
  "stock_name": "string",
  "quantity": "string",
  "purchase_price": "string",
  "purchase_date": "string"
}
```

  - **Properties:**
    - `stock_name` (string, required): The name of the stock being added.
    - `quantity` (string, required): The quantity of shares purchased.
    - `purchase_price` (string, required): The price at which the shares were purchased.
    - `purchase_date` (string, required): The date when the shares were purchased.
- **Example Request (cURL):**

```
Unset
curl --location
'http://192.168.207.153:5545/api/stock-tracker/user/porfolio-holdings/addStock' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0' \
--data '{
    "stock_name": "Askok Leyland",
    "quantity": "60",
    "purchase_price": "198.00",
    "purchase_date": "2024-03-09"
}'
```

**3. Response**

- **Success Status Code:**
    - `201 Created`: Indicates that the stock was successfully added to the portfolio. This is the most appropriate code for a resource creation action.
    - `200 OK`: Also possible, but less semantically correct than 201.
- **Error Status Codes:**
    - `400 Bad Request`: The request body is invalid (e.g., missing fields, incorrect data types) or contains validation errors.
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `500 Internal Server Error`: An unexpected error occurred on the server while adding the stock.

---

## 13. Update Stock Holding Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/portfolio-holdings/update-stock`
- **Method:** `PUT`
- **Description:** This endpoint is used to update an existing stock holding within a user's portfolio.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/porfolio-holdings/update-stock`
- **Headers:**
    - `Content-Type: application/json` (Indicates that the request body is in JSON format)
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Body:**
  **Format:** JSON
  **Schema:**

```
Unset
{
  "holding_id": "integer",
  "stock_name": "string",
  "quantity": "string",
  "purchase_price": "string",
  "purchase_date": "string"
}
```

    - 
    - **Properties:**
        - `holding_id` (integer, required): The unique identifier of the holding to be updated.
        - `stock_name` (string, optional): The updated name of the stock.
        - `quantity` (string, optional): The updated quantity of shares held.

        - `purchase_price` (string, optional): The updated price at which the shares were purchased.
        - `purchase_date` (string, optional): The updated date when the shares were purchased.
- **Example Request (cURL):**

```
Unset
curl --location --request PUT
'localhost:5545/api/stock-tracker/user/porfolio-holdings/update-stock' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0' \
--data '{
    "holding_id" : 5 ,
    "stock_name": "Audi",
    "quantity": "25",
    "purchase_price": "40000.00",
    "purchase_date": "2025-04-03"}'
```

**3. Response**

- **Success Status Code:**
    - `200 OK`: Indicates that the stock holding was successfully updated.
- **Error Status Codes:**
    - `400 Bad Request`: The request body is invalid (e.g., missing `holding_id`, incorrect data types) or contains validation errors.
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `404 Not Found`: The holding with the provided `holding_id` was not found.
    - `500 Internal Server Error`: An unexpected error occurred on the server while updating the holding.

---

## 14. Delete Stock Holding Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/porfolio-holdings/delete-stock/{holdingId}`
- **Method:** `DELETE`
- **Description:** This endpoint is used to delete an existing stock holding from a user's portfolio.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/porfolio-holdings/delete-stock/{holdingId}`

    - `{holdingId}` (integer, required): This is a path parameter representing the unique identifier of the holding to be deleted. In the example, it's `16`.
- **Headers:**
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**

    - None (based on the cURL provided, but verify if any are used)
- **Body:**
    - None (DELETE requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location --request DELETE
'localhost:5545/api/stock-tracker/user/porfolio-holdings/delete-stock/16'
\
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Codes:**
    - `200 OK`: Indicates that the holding was successfully deleted, and the response may contain a confirmation message.
    - `204 No Content`: Indicates that the holding was successfully deleted, and the server does not need to send back any content. This is often preferred for DELETE requests.
- **Error Status Codes:**
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `404 Not Found`: The holding with the provided `{holdingId}` was not found.
    - `500 Internal Server Error`: An unexpected error occurred on the server while deleting the holding.

---

## 15. Fetch User Watchlist Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/watch-list/fetch-list`
- **Method:** `GET`
- **Description:** This endpoint retrieves the list of stocks or other assets that the authenticated user has added to their watchlist. A watchlist is a collection of items a user wants to monitor.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/watch-list/fetch-list`
- **Headers:**
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
    - None (based on the cURL provided, but verify if any are used for filtering or pagination)
- **Body:**
    - None (GET requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location
'http://localhost:5545/api/stock-tracker/user/watch-list/fetch-list' \
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:** `200 OK`
- **Error Status Codes:**
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `500 Internal Server Error`: An unexpected error occurred on the server while retrieving the watchlist.
- **Success Response Body:**
  - **Format:** JSON (Likely, and highly recommended for structured data)
  - **Schema (Example - Adapt to your actual response):**

```
Unset
[
  {
    "watch_list_id": "integer",
    "asset_symbol": "string",
    "asset_name": "string",
    "asset_type": "string",
    "added_date": "string"
    //    ...   other watchlist item details
  },
  {
    "watch_list_id": "integer",
    "asset_symbol": "string",
    "asset_name": "string",
    "asset_type": "string",
    "added_date": "string"
    //    ...   other watchlist item details
  },
  //    ...   more watchlist item objects ]
```

---

## 16. Add Asset to User Watchlist Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/watch-list/add`
- **Method:** `POST`
- **Description:** This endpoint is used to add a new asset (e.g., a stock) to the authenticated user's watchlist.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/watch-list/add`
- **Headers:**
  - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
  - `symbol` (string, required): The symbol or identifier of the asset to add (e.g., "ASHOKLEY").
  - `stockName` (string, required): The name of the asset to add (e.g., "Ashok Leyland Ltd"). Note that the cURL example shows URL-encoded characters (`%20` for spaces).
- **Body:**
  - None (POST requests can sometimes have an empty body, but it's less common when adding data. Verify if your API requires a body.)
- **Example Request (cURL):**

```
Unset

curl --location --request POST
'http://localhost:5545/api/stock-tracker/user/watch-list/add?symbol=ASHOKL
EY&stockName=Ashok%20Leyland%20Ltd' \
--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Code:**
  - `201 Created`: Indicates that the asset was successfully added to the watchlist. This is the most appropriate code for a resource creation action.
  - `200 OK`: Also possible, but less semantically correct than 201.
- **Error Status Codes:**
  - `400 Bad Request`: The request is malformed (e.g., missing `symbol` or `stockName` parameters).
  - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
  - `409 Conflict`: The asset might already exist in the user's watchlist.
  - `500 Internal Server Error`: An unexpected error occurred on the server while adding the asset.
- **Success Response Body (Example - Adapt to your actual response):**
  - **Format:** JSON (Likely, and highly recommended)
  - **Schema (Example - 201 Created):**

```
Unset
{
  "watch_list_id": "integer",
  "asset_symbol": "string",
  "asset_name": "string",
  "added_date": "string",
  "message": "string"  }
```

  - **Properties (Example - 201 Created):**

        - `watch_list_id` (integer): The unique identifier assigned to the newly created watchlist item.
        - `asset_symbol` (string): The symbol or identifier of the added asset.
        - `asset_name` (string): The name of the added asset.
        - `added_date` (string): The date when the asset was added to the watchlist. The format should be specified (e.g., YYYY-MM-DD).
        - `message` (string, optional): A success message confirming the addition
-

---

## 17. Delete Asset from User Watchlist Endpoint API Documentation:

**1. Overview**

- **Endpoint:** `/api/stock-tracker/user/watch-list/delete/{watchListId}`
    - `{watchListId}` (integer, required): This is a path parameter representing the unique identifier of the watchlist item to be deleted. In the example, it's `9`.
- **Method:** `DELETE`
- **Description:** This endpoint is used to remove an asset (e.g., a stock) from the authenticated user's watchlist.

**2. Request**

- **URL:** `http://localhost:5545/api/stock-tracker/user/watch-list/delete/{watchListId}`
- **Headers:**
    - `Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0` (Provides authentication credentials using Basic Authentication)
- **Query Parameters:**
    - None (based on the cURL provided, but verify if any are used)
- **Body:**
    - None (DELETE requests typically do not have a request body)
- **Example Request (cURL):**

```
Unset
curl --location --request DELETE
'http://localhost:5545/api/stock-tracker/user/watch-list/delete/9' \

--header 'Authorization: Basic TW9raW5kZXI6TW9raUAxMjM0'
```

**3. Response**

- **Success Status Codes:**
    - `200 OK`: Indicates that the asset was successfully removed from the watchlist, and the response may contain a confirmation message.
    - `204 No Content`: Indicates that the asset was successfully removed from the watchlist, and the server does not need to send back any content. This is often preferred for DELETE requests.
- **Error Status Codes:**
    - `401 Unauthorized`: Authentication failed. The provided credentials are invalid.
    - `404 Not Found`: The watchlist item with the provided `{watchListId}` was not found.
    - `500 Internal Server Error`: An unexpected error occurred on the server while deleting the asset.

- **Success Response Body (200 OK Example - Adapt if your API returns content):**
    - **Format:** JSON (Optional, might be used for a confirmation message)
    - **Schema (Example):**

```
Unset
{   "message": "string"   }
```

- 
    - **Properties (Example):**
        - `message` (string, optional): A success message confirming the deletion (e.g., "Asset with ID 9 removed from watchlist").

---

## 6. Database Schema (Sample):

**Users Table:**

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

**Stocks Table:**

```
CREATE TABLE stocks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    symbol VARCHAR(10) UNIQUE NOT NULL,
    name VARCHAR(255) NOT NULL
);
```

**Watchlist Table:**

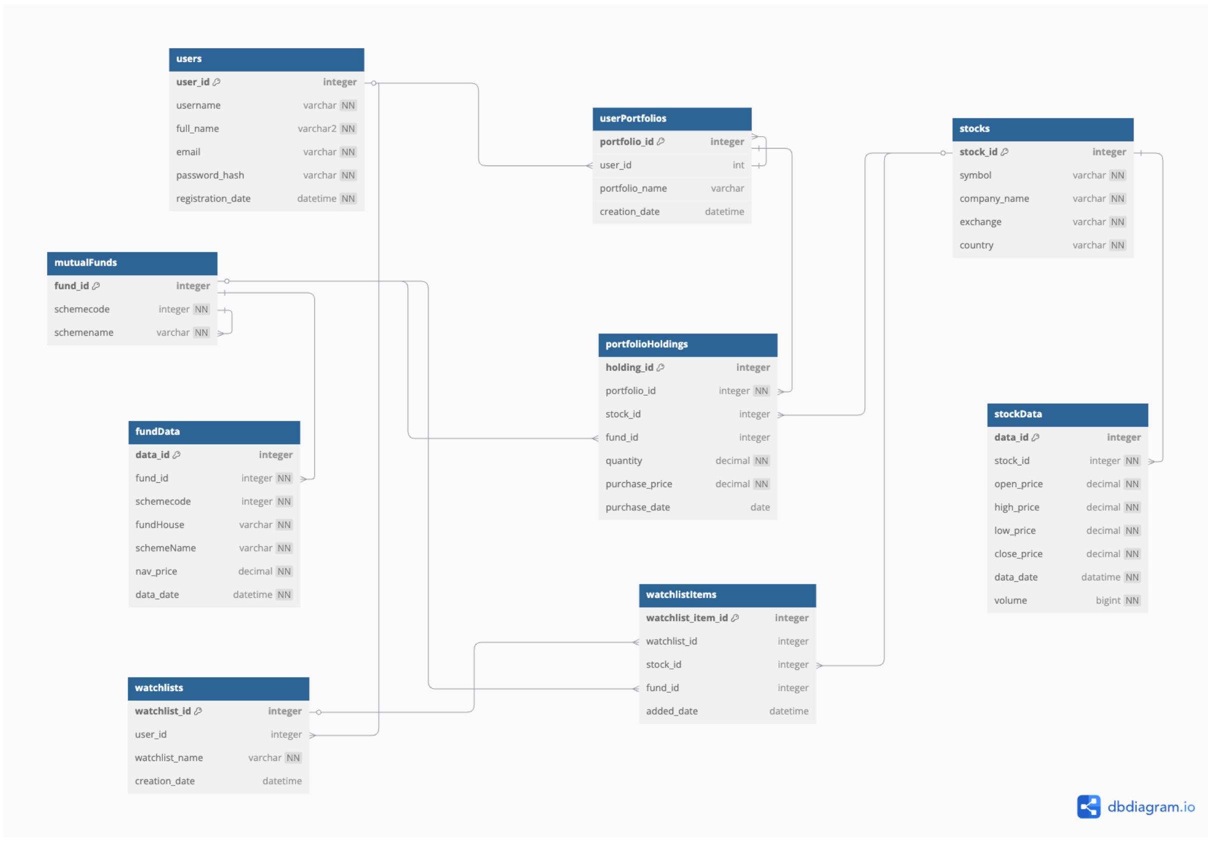
```
CREATE TABLE watchlist (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    stock_id INT,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (stock_id) REFERENCES stocks(id)
);
```

**StockPrices Table:**

```
CREATE TABLE stock_prices (
    id INT AUTO_INCREMENT PRIMARY KEY,
    stock_id INT,
    date DATE NOT NULL,
    open_price DECIMAL(10, 2),
    close_price DECIMAL(10, 2),
    high_price DECIMAL(10, 2),
    low_price DECIMAL(10, 2),
    volume INT,
    FOREIGN KEY (stock_id) REFERENCES stocks(id));
```

---

## 7. Schema Structure :



**Link :-** https://dbdiagram.io/d/Stocktracker-67d913e375d75cc84479101a

---

## 8. Communication Flow:

1. **User logs in**:
    - Flutter client sends credentials to the Spring Boot API.
    - Spring Boot validates credentials and returns a JWT token.
2. **User views stocks**:
    - Flutter client calls the Spring Boot API to fetch stocks from the database.
    - Stock data is retrieved either from the MySQL database or an external API.
3. **User adds a stock to the watchlist**:

    - Flutter client sends a request to Spring Boot API to add a stock to the user's watchlist.
    - Spring Boot stores this data in the MySQL database.

---

## 9. Security Considerations:

- Use HTTPS for all API communications.
- Input validation and sanitization to prevent SQL injection or XSS attacks.
- Rate limiting to prevent abuse of APIs.

---