Okay, I've analyzed the code you provided. It's a web-based trading and chart visualization platform.

Here's the README.md and detailed documentation:

Markdown

# Trading Chart Visualization Platform

## Project Overview

This project is a web-based trading and chart visualization platform designed to provide users with real-time and historical stock market data, interactive charts, and broker integration capabilities. It supports user authentication, setup for multiple brokers (Alice Blue and Upstox), and dynamic fetching of master contract lists for various exchanges (NSECM, NSEFO, MCXFO). The frontend is built with HTML, Tailwind CSS, and JavaScript, utilizing TradingView's Lightweight Charts library for chart rendering. The backend is powered by Flask and uses Socket.IO for real-time communication.

## Features

\* **\*\*User Authentication\*\***: Secure signup and login functionality for platform users.

\* **\*\*Broker Integration\*\***:

\* Setup and secure storage of API credentials for Alice Blue and Upstox brokers.

\* Handles API session management and token refresh.

\* **\*\*Master Contract Management\*\***:

\* Downloads and caches master contract lists for NSECM, NSEFO, and MCXFO exchanges from specified URLs.

\* Provides a searchable list of tradable instruments.

\* **\*\*Interactive Charting\*\***:

\* Displays historical and real-time candlestick charts using TradingView's Lightweight Charts library.

\* Supports various chart timeframes (1 minute, 3 minutes, 5 minutes, 15 minutes, 30 minutes, 1 hour, 1 day, 1 week, 1 month).

\* Allows users to search for and switch between different financial instruments.

\* Real-time price updates via WebSockets (Socket.IO).

\* **\*\*Data Handling\*\***:

\* Fetches historical OHLCV (Open, High, Low, Close, Volume) data.

\* Subscribes to real-time market data feeds.

\* Caching mechanisms for frequently accessed data like contract lists.

\* **\*\*Configuration\*\***: User-specific configurations, including encrypted API keys, are stored in a `config.ini` file.

\* **\*\*Web Interface\*\***:

\* Landing page, login/signup pages, broker API setup page, and a comprehensive chart visualization page.

\* Responsive design using Tailwind CSS.

## Project Structure

.

├── backend.py # Main Flask application, Socket.IO handling, broker API logic

├── config.ini # Configuration file (API keys, user settings - encrypted)

├── requirements.txt # Python dependencies (currently empty, should be populated)

├── csv/ # Directory for storing master contract CSV files

│ ├── MCXFO.csv

│ ├── NSECM.csv

│ └── NSEFO.csv

├── templates/ # HTML templates for the web interface

│ ├── broker\_setup.html # Page for setting up broker API credentials

│ ├── chart\_page.html # Main trading chart interface

│ ├── landing.html # Application landing page

│ ├── login.html # User login page for the platform

│ ├── platform\_login.html # (Likely a duplicate or alternative login - review usage)

│ └── platform\_signup.html # User signup page for the platform

└── NSEFO.csv # (Appears to be a duplicate, likely belongs in csv/)

## Prerequisites

\* Python 3.x

\* PIP (Python package installer)

\* Broker API credentials (Alice Blue and/or Upstox) if you intend to connect to live data.

## Installation

1. \*\*Clone the repository (if applicable) or download the files.\*\*

2. \*\*Create a virtual environment (recommended):\*\*

```bash

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

```

3. \*\*Install dependencies:\*\*

While `requirements.txt` is currently empty, based on the `backend.py` imports, you would typically run:

```bash

pip install Flask python-socketio requests Flask-CORS Werkzeug cryptography pandas

```

(Add any other necessary libraries to `requirements.txt` and install them.)

4. \*\*Configure `config.ini`:\*\*

Create or update `config.ini` in the root directory. It will be used to store user credentials and API keys. The application will prompt for broker setup if credentials are not found or are invalid.

A typical structure might look like this initially (though the application manages its creation and encryption):

```ini

[DEFAULT]

encryption\_key = your\_generated\_fernet\_key\_here

[aliceblue\_user1]

username = aliceblue\_username

password = aliceblue\_password\_or\_pin

api\_key = aliceblue\_api\_key

api\_secret = aliceblue\_api\_secret\_or\_encryption\_key

# session\_id, etc., will be added by the application

[upstox\_user1]

api\_key = upstox\_api\_key

api\_secret = upstox\_api\_secret

redirect\_uri = your\_upstox\_redirect\_uri

# access\_token, etc., will be added by the application

```

\*Note: The application is designed to handle the creation and encryption of sensitive details in `config.ini` during the broker setup process.\*

5. \*\*Master Contracts:\*\*

The application attempts to download master contract files into the `csv/` directory. Ensure this directory exists. If downloads fail, you might need to manually place the `NSECM.csv`, `NSEFO.csv`, and `MCXFO.csv` files in the `csv/` directory. The expected download URLs are hardcoded in `backend.py`.

## Usage

1. \*\*Run the Flask application:\*\*

```bash

python backend.py

```

2. Open your web browser and navigate to `http://127.0.0.1:5000/` (or the address shown in the console).

3. \*\*Sign up or Log in:\*\* Create a platform account or log in if you already have one.

4. \*\*Broker Setup:\*\*

\* Navigate to the broker setup page (usually `/broker\_setup` after login).

\* Enter your API credentials for Alice Blue or Upstox. These will be encrypted and saved in `config.ini`.

5. \*\*Chart Page:\*\*

\* Once logged in and broker(s) are set up, navigate to the chart page (`/chart`).

\* Search for financial instruments.

\* Select instruments to view their candlestick charts.

\* Change timeframes and potentially apply indicators (if implemented).

## Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

## License

(Specify your license here, e.g., MIT, GPL, etc. If no license is chosen, you can state "All Rights Reserved.")

---

**Detailed Documentation**

**1.** backend.py

This is the core of the application, a Flask-based web server that handles user authentication, broker API interactions, data fetching, real-time updates via Socket.IO, and serves the frontend HTML pages.

**Key Libraries Used:**

* **Flask**: Micro web framework for Python.
* **python-socketio**: Library for real-time, bidirectional communication between web clients and servers.
* **requests**: HTTP library for making API calls to brokers and downloading contract files.
* **Flask-CORS**: Flask extension for handling Cross-Origin Resource Sharing (CORS).
* **Werkzeug**: WSGI utility library (used for security, routing, and running the development server).
* **cryptography**: Library for encrypting and decrypting sensitive data like API keys stored in config.ini.
* **configparser**: For reading and writing configuration files.
* **pandas**: Used for handling and searching within the CSV contract files.
* **json**: For working with JSON data.
* **logging**: For application logging.
* **os, threading, datetime, math**: Standard Python libraries for various utility functions.

**Overall Structure:**

* **Configuration Loading**: Reads config.ini to get API keys and other settings. Handles encryption key generation if not present.
* **Flask App Initialization**: Sets up the Flask app, CORS, and Socket.IO.
* **Broker API Classes (**AliceBlueApi**,** UpstoxApi**)**: These classes encapsulate the logic for interacting with the respective broker APIs. They handle:
  + Login and session generation.
  + Fetching historical data.
  + Subscribing to and handling real-time market data feeds.
  + Managing API tokens and refreshing them.
  + Storing and retrieving credentials securely using config.ini and the cryptography library.
* **Data Management Functions**:
  + download\_master\_contracts(): Downloads master contract CSV files from specified URLs if they don't exist or are outdated.
  + load\_master\_contract(): Loads instruments from CSV files into pandas DataFrames for efficient searching.
  + search\_scrip(): Provides an endpoint for searching instruments across different exchanges (NSECM, NSEFO, MCXFO).
* **Flask Routes**:
  + /: Serves the landing page (landing.html).
  + /signup, /login, /logout: Handle platform user authentication.
  + /platform\_login: (Appears similar to /login, may need review for distinct functionality).
  + /broker\_setup: Allows users to input and save their broker API credentials.
  + /check\_broker\_login\_status: Checks if broker APIs are connected.
  + /chart: Serves the main chart visualization page (chart\_page.html).
  + /get\_active\_brokers: Returns a list of successfully configured brokers.
  + /search\_scrip\_backend: Backend endpoint for scrip search functionality used by the chart page.
  + /get\_user\_details: Returns details of the logged-in user.
* **Socket.IO Event Handlers**:
  + connect, disconnect: Handle client connections and disconnections.
  + get\_historical\_data: Fetches historical candlestick data for a given instrument and timeframe from the selected broker.
  + subscribe\_feed: Subscribes to real-time price feeds for specified instruments.
  + unsubscribe\_feed: Unsubscribes from real-time feeds.
  + The backend listens for messages from broker WebSockets and emits formatted data to connected clients.

**Broker API Credential Management (**config.ini**)**:

* The config.ini file is crucial for storing user API keys and other settings.
* **Encryption**: Sensitive information like API secrets, passwords, and session tokens are encrypted using the Fernet symmetric encryption algorithm from the cryptography library.
  + An encryption\_key is generated and stored in the [DEFAULT] section of config.ini if one doesn't exist. This key is used for all encryption/decryption operations for that user's configuration.
* **Sections**: Credentials for each broker and user are stored in separate sections (e.g., [aliceblue\_someuser], [upstox\_someuser]).
* **Setup**: The /broker\_setup route guides users to input their credentials, which are then encrypted and saved to config.ini.
* **Loading**: When the application starts or when a user logs in, it attempts to load and decrypt the credentials for the configured brokers.

**Master Contract CSV Files**:

* CSV files (NSECM.csv, NSEFO.csv, MCXFO.csv) located in the csv/ directory contain lists of all tradable instruments for the respective exchanges/segments.
* **Downloading**: The download\_master\_contracts() function checks if these files exist and are recent. If not, it attempts to download them from predefined URLs (hardcoded in backend.py).
* **Loading & Searching**: The load\_master\_contract() function loads these CSVs into pandas DataFrames. The search\_scrip() and /search\_scrip\_backend functionalities use these DataFrames to allow users to search for instruments by name or symbol.

**2. HTML Templates (**templates/**)**

These files define the structure and basic layout of the web pages presented to the user. They use Tailwind CSS for styling and include JavaScript for client-side interactions, especially chart\_page.html.

* landing.html:
  + Purpose: The initial page users see. Likely provides an overview of the platform, features, and navigation to login/signup.
  + Content: Includes navigation links for "Charts", "Broker Setup", "Login", and "Sign Up".
* login.html:
  + Purpose: Allows existing platform users to log in.
  + Content: A form with fields for "Username" and "Password", and a "Login" button.
* platform\_signup.html:
  + Purpose: Allows new users to create an account on the platform.
  + Content: A form with fields for "Username", "Password", and "Confirm Password", and a "Sign Up" button.
* platform\_login.html:
  + Purpose: Seems to be an alternative or duplicate login page. Its specific role should be clarified by its usage in backend.py routes. The initial content suggests it's very similar to login.html.
  + Content: A form for "Username" and "Password", and a "Login" button, with a link to the signup page.
* broker\_setup.html:
  + Purpose: Allows logged-in users to configure their broker API credentials (Alice Blue, Upstox).
  + Content: Forms for each supported broker, requesting necessary API keys, secrets, usernames, passwords/PINs, and redirect URIs where applicable. Includes "Save" buttons and status messages. JavaScript on this page handles form submissions, sending data to the backend /broker\_setup route, and displaying success/error messages.
* chart\_page.html:
  + Purpose: The main interface for chart visualization and trading interactions.
  + Key Client-Side JavaScript Functionalities:
    - **Socket.IO Connection**: Establishes a WebSocket connection with the backend for real-time data.
    - **TradingView Lightweight Charts**:
      * Initializes and configures the chart.
      * Handles adding candlestick series, volume series, and potentially indicators.
      * Manages chart timeframe changes (1m, 3m, 5m, etc.).
      * Updates the chart with historical and real-time data received via Socket.IO.
    - **Scrip Search**:
      * Provides an input field for users to search for instruments.
      * Sends search queries to the /search\_scrip\_backend endpoint.
      * Displays search results in a dropdown, allowing users to select an instrument to chart.
    - **Broker and Instrument Selection**: Allows users to choose the active broker and the instrument to display.
    - **Data Fetching**:
      * Requests historical data via Socket.IO (get\_historical\_data event) when a new scrip is selected or timeframe changes.
      * Subscribes to real-time feeds (subscribe\_feed event) for the currently charted instrument.
    - **UI Updates**: Dynamically updates UI elements like the current price, scrip name, and broker status.
    - **Logout**: Provides a logout link.

**3. CSV Files (**csv/**)**

* NSECM.csv: Master list of instruments traded on the NSE Cash Market segment.
* NSEFO.csv: Master list of instruments traded on the NSE Futures and Options segment. (A duplicate also exists in the root folder).
* MCXFO.csv: Master list of instruments traded on the MCX Futures and Options segment.

Structure (General for all CSVs):

These files are comma-separated value files. Each row typically represents a unique tradable instrument. The columns include:

* ExchangeSegment: e.g., NSEFO, NSECM, MCXFO
* ExchangeInstrumentID: Broker-specific instrument identifier.
* InstrumentType: e.g., FUTSTK, OPTIDX, EQ, FUTCOM.
* Name: Scrip symbol or name (e.g., WIPRO, BANKNIFTY, GOLD).
* Description: A more detailed description of the instrument.
* Series: e.g., EQ, FUTSTK, FUTIDX.
* NameWithSeries: Combination of name and series.
* InstrumentID: A unique ID for the instrument.
* PriceBandHigh, PriceBandLow: Circuit limits.
* FreezeQty: Quantity beyond which orders might be frozen.
* TickSize: Minimum price movement.
* LotSize: Number of units in one lot (especially for F&amp;O).
* Multiplier: For calculating contract value.
* UnderlyingInstrumentId, UnderlyingIndexName: For derivative instruments.
* ContractExpiration: Expiry date for derivative contracts.
* StrikePrice, OptionType: For options contracts.
* displayName: User-friendly name for display.
* ISIN: (For NSECM) International Securities Identification Number.
* PriceNumerator, PriceDenominator: Used for price calculations.
* FullDescription: A comprehensive description.

These files are essential for looking up instrument tokens/IDs based on user search queries, which are then used to fetch market data from broker APIs.

**4.** config.ini

* Purpose: To store user-specific configurations, primarily broker API credentials and the Fernet encryption key.
* Structure:

Ini, TOML

[DEFAULT]

encryption\_key = b'Generated\_Fernet\_Key\_String\_Here==' # This is generated by the app

[brokerAlias\_username] # e.g., aliceblue\_testuser or upstox\_myaccount

api\_key = ENCRYPTED\_API\_KEY

api\_secret = ENCRYPTED\_API\_SECRET

username = ENCRYPTED\_USERNAME

password = ENCRYPTED\_PASSWORD\_OR\_PIN

# Other broker-specific encrypted fields like session\_id, access\_token, etc.

* Management:
  + The backend.py script manages the creation and updating of this file.
  + Sensitive values are encrypted before being written and decrypted when read.
  + If the file or encryption\_key is missing, it's created on the first run or when a user attempts to set up broker credentials.
* Current State: The provided config.ini is empty. It will be populated as users configure their broker details through the /broker\_setup page.

**5.** requirements.txt

* Purpose: This file should list all Python package dependencies required to run the project. This allows for easy setup of the environment using pip install -r requirements.txt.
* Current State: The provided requirements.txt is empty.
* Recommended Contents (based on backend.py imports):
* Flask
* python-socketio
* Flask-SocketIO
* requests
* Flask-CORS
* Werkzeug
* cryptography
* pandas
* gunicorn # (Often used for production deployment)
* eventlet # (Often used with SocketIO for async)

This documentation should give you a comprehensive understanding of the project's components and their interactions.