Chatgpt prompt: Write Interactive brokers API interface program and front end to create an order to

trail 20% behind as the stock falls execute buy order for 100 quantity as it turns

Then trail the stock 20% till it continues to rise, when it turns around starts to fall execute the sell order doubling the position

Then trail the stock as it falls execute buy order as it turns to rise

Then continue the loop for 5 iterations

with user authentication, error checking, and more advanced order management and display if order executed successfully or not

=================================

To create an Interactive Brokers (IB) API interface program and a front-end that follows the specified trading strategy with user authentication, error checking, and advanced order management, we'll break down the solution into several parts.

1. \*\*Python Backend with IB API\*\*: Manages the trading logic and communicates with the IB API.

2. \*\*Flask Web Interface\*\*: Provides a front-end for user interaction and authentication.

3. \*\*Error Handling and Order Status Tracking\*\*: Ensures robust operation and user feedback.

### Backend: Python with Interactive Brokers API

Install the necessary packages:

```bash

pip install ibapi flask flask-login

```

#### `ib\_trading\_strategy.py`

This Python script implements the specified trading strategy and manages the IB API interaction.

```python

from ibapi.client import EClient

from ibapi.wrapper import EWrapper

from ibapi.contract import Contract

from ibapi.order import Order

import threading

import time

class IBApp(EClient, EWrapper):

def \_\_init\_\_(self):

EClient.\_\_init\_\_(self, self)

self.nextOrderId = None

self.order\_status = None

self.iterations = 0

self.max\_iterations = 5

self.position = 0

self.trail\_percent = 20

def error(self, reqId, errorCode, errorString):

print(f"Error {reqId}, {errorCode}, {errorString}")

def nextValidId(self, orderId):

self.nextOrderId = orderId

self.start\_trading()

def orderStatus(self, orderId, status, filled, remaining, avgFillPrice, permId, parentId, lastFillPrice, clientId, whyHeld, mktCapPrice):

print(f"OrderStatus. Id: {orderId}, Status: {status}, Filled: {filled}, Remaining: {remaining}")

self.order\_status = status

if status == 'Filled' and self.iterations < self.max\_iterations:

if self.position > 0:

self.position \*= 2

self.sell\_trailing\_stop(self.position)

else:

self.position = 100

self.buy\_trailing\_stop(self.position)

self.iterations += 1

def create\_stock\_contract(self, symbol):

contract = Contract()

contract.symbol = symbol

contract.secType = "STK"

contract.exchange = "SMART"

contract.currency = "USD"

return contract

def create\_trailing\_order(self, action, quantity):

order = Order()

order.action = action

order.orderType = "TRAIL"

order.totalQuantity = quantity

order.trailingPercent = self.trail\_percent

return order

def buy\_trailing\_stop(self, quantity):

contract = self.create\_stock\_contract(self.symbol)

order = self.create\_trailing\_order("BUY", quantity)

self.placeOrder(self.nextOrderId, contract, order)

def sell\_trailing\_stop(self, quantity):

contract = self.create\_stock\_contract(self.symbol)

order = self.create\_trailing\_order("SELL", quantity)

self.placeOrder(self.nextOrderId, contract, order)

def start\_trading(self):

self.buy\_trailing\_stop(100)

def run\_app(symbol):

app = IBApp()

app.symbol = symbol

app.connect("127.0.0.1", 7497, 0)

# Start the socket in a thread

api\_thread = threading.Thread(target=app.run, daemon=True)

api\_thread.start()

time.sleep(1) # Sleep interval to allow time for connection to server

while app.nextOrderId is None:

time.sleep(0.1)

# Wait for the iterations to complete

while app.iterations < app.max\_iterations:

time.sleep(0.1)

app.disconnect()

return "Trading completed"

if \_\_name\_\_ == "\_\_main\_\_":

symbol = "AAPL" # Example stock

print(run\_app(symbol))

```

### Front-end: Flask Web Interface with User Authentication

#### `app.py`

This Flask app includes user authentication, form handling, and order status display.

```python

from flask import Flask, request, render\_template, redirect, url\_for, flash, session

from flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user

from ib\_trading\_strategy import run\_app

import subprocess

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

# In-memory user store

users = {'user': {'password': 'password'}}

class User(UserMixin):

def \_\_init\_\_(self, id):

self.id = id

@login\_manager.user\_loader

def load\_user(user\_id):

return User(user\_id)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

if username in users and users[username]['password'] == password:

user = User(username)

login\_user(user)

return redirect(url\_for('index'))

else:

flash('Invalid username or password')

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

@app.route('/submit', methods=['POST'])

@login\_required

def submit():

symbol = request.form['symbol']

result = run\_app(symbol)

flash(f"Trading result for {symbol}: {result}")

return redirect(url\_for('index'))

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

```

#### `templates/index.html`

HTML template for the main page with a form to create a trailing buy order.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Trailing Buy/Sell Order</title>

</head>

<body>

<h1>Automated Trading Strategy</h1>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul>

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

<form action="{{ url\_for('submit') }}" method="POST">

<label for="symbol">Stock Symbol:</label>

<input type="text" id="symbol" name="symbol" required>

<br><br>

<button type="submit">Submit</button>

</form>

<br>

<a href="{{ url\_for('logout') }}">Logout</a>

</body>

</html>

```

#### `templates/login.html`

HTML template for the login page.

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

</head>

<body>

<h1>Login</h1>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul>

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %}

<form action="{{ url\_for('login') }}" method="POST">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<br><br>

<button type="submit">Login</button>

</form>

</body>

</html>

```

### Running the Application

1. Start the Flask server by running `app.py`:

```bash

python app.py

```

2. Access the web interface in your browser at `http://127.0.0.1:5000`.

3. Log in using the credentials (`user`/`password`).

4. Fill out the form with the stock symbol and submit to start the trading strategy.

This setup provides a user-friendly interface with authentication, error checking, and order status tracking, ensuring that the user can log in, submit an order, and receive feedback on the order's status and results. The trading strategy is automated for five iterations as requested.