Chatgpt prompt: Write Interactive brokers API interface program and front end to create an order to trail 20% behind the stock as the stock rises and execute sell as the stock starts falling and hits the 20% fall with user authentication, error checking, and more advanced order management and display if order executed successfully or not

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

Creating a more advanced version of the Interactive Brokers API interface program with user authentication, error checking, and advanced order management involves several steps. Below, I'll provide a comprehensive solution that includes these features.

### Backend: Python with Interactive Brokers API

Ensure you have the necessary packages installed:

```bash

pip install ibapi flask flask-login

```

#### `ib\_trailing\_stop.py`

This Python script connects to the IB API and manages the trailing stop order with error checking and order status tracking.

```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

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

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

def nextValidId(self, orderId):

self.nextOrderId = orderId

self.start()

def start(self):

symbol = self.symbol

trail\_percent = self.trail\_percent

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

self.reqContractDetails(1, stock\_contract)

order = self.create\_trailing\_stop\_order(trail\_percent)

self.placeOrder(self.nextOrderId, stock\_contract, order)

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

self.order\_status = status

print(f"OrderStatus. Id: {orderId}, Status: {status}")

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\_stop\_order(self, trail\_percent):

order = Order()

order.action = "SELL"

order.orderType = "TRAIL"

order.trailingPercent = trail\_percent

return order

def run\_app(symbol, trail\_percent):

app = IBApp()

app.symbol = symbol

app.trail\_percent = trail\_percent

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 until the order status is known

while app.order\_status is None:

time.sleep(0.1)

app.disconnect()

return app.order\_status

```

### 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\_trailing\_stop 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']

trail\_percent = 20.0 # fixed 20% trailing stop

order\_status = run\_app(symbol, trail\_percent)

flash(f"Order Status for {symbol}: {order\_status}")

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 stop 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 Stop Order</title>

</head>

<body>

<h1>Create Trailing Stop Order</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 place the order.

This setup provides a user-friendly interface with authentication, error checking, and order status tracking. It ensures that the user can log in, submit an order, and receive feedback on the order's status.