Yes do it

Perfect, Jagdev! Modularizing your app will make it easier to maintain, extend, and debug. Here's how we’ll split your app.py into clean, reusable modules:

**📁 Suggested File Structure**

multi-company-stock-app/

│

├── app.py

├── logger.py

├── auth.py

├── dashboard.py

├── config.yaml

├── user\_logs.csv

├── logger.js

└── requirements.txt

**1️⃣ logger.py — Visitor Logging**

python

import pandas as pd

import datetime

import os

import streamlit as st

import streamlit.components.v1 as components

LOG\_FILE = "user\_logs.csv"

def initialize\_log\_file():

if not os.path.exists(LOG\_FILE) or os.path.getsize(LOG\_FILE) == 0:

pd.DataFrame(columns=["Timestamp", "Device", "Latitude", "Longitude"]).to\_csv(LOG\_FILE, index=False)

def inject\_logger\_script():

components.html("""

<script src="logger.js"></script>

""", height=0)

def log\_visitor():

params = st.query\_params

if "device" in params and "latitude" in params and "longitude" in params:

device = params["device"]

lat = params["latitude"]

lon = params["longitude"]

timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

if "logged" not in st.session\_state:

pd.DataFrame([[timestamp, device, lat, lon]],

columns=["Timestamp", "Device", "Latitude", "Longitude"]).to\_csv(LOG\_FILE, mode='a', header=False, index=False)

st.session\_state.logged = True

def show\_logs():

try:

logs = pd.read\_csv(LOG\_FILE)

if logs.empty:

st.info("📭 No visitor logs yet.")

else:

st.dataframe(logs)

except pd.errors.EmptyDataError:

st.warning("⚠️ Visitor log file is empty or corrupted.")

logs = pd.DataFrame(columns=["Timestamp", "Device", "Latitude", "Longitude"])

st.dataframe(logs)

**2️⃣ auth.py — Authentication**

python

import streamlit as st

import streamlit\_authenticator as stauth

import yaml

from yaml.loader import SafeLoader

def load\_authenticator():

with open("config.yaml") as file:

config = yaml.load(file, Loader=SafeLoader)

authenticator = stauth.Authenticate(

config['credentials'],

config['cookie']['name'],

config['cookie']['key'],

config['cookie']['expiry\_days'],

config['preauthorized']

)

return authenticator

**3️⃣ dashboard.py — Stock Dashboard**

python

import streamlit as st

import pandas as pd

import requests

import os

API\_KEY = os.environ.get("POLYGON\_API\_KEY")

BASE\_URL = "https://api.polygon.io/v2/aggs/ticker"

START\_DATE = "2025-01-01"

END\_DATE = "2025-05-11"

COMPANIES = {

"Apple Inc. (AAPL)": "AAPL",

"Alphabet Inc. (GOOGL)": "GOOGL",

"Microsoft Corporation (MSFT)": "MSFT",

"NVIDIA Corporation (NVDA)": "NVDA",

"Tesla, Inc. (TSLA)": "TSLA",

"Infosys Ltd. (INFY)": "INFY",

"Amazon.com, Inc. (AMZN)": "AMZN",

"Meta Platforms, Inc. (META)": "META",

"Intel Corporation (INTC)": "INTC",

"Advanced Micro Devices, Inc. (AMD)": "AMD"

}

def fetch\_stock\_data(ticker):

if not API\_KEY:

st.error("API key not found. Please set POLYGON\_API\_KEY in your environment.")

return None

url = f"{BASE\_URL}/{ticker}/range/1/day/{START\_DATE}/{END\_DATE}?apiKey={API\_KEY}"

response = requests.get(url)

if response.status\_code != 200:

st.error("Failed to fetch data.")

return None

data = response.json()

if "results" not in data:

st.warning("No results found.")

return None

df = pd.DataFrame(data["results"])

df["date"] = pd.to\_datetime(df["t"], unit="ms").dt.date

df = df.rename(columns={

"v": "Volume",

"vw": "VWAP",

"o": "Open",

"c": "Close",

"h": "High",

"l": "Low",

"n": "Trades"

})

return df[["date", "Volume", "VWAP", "Open", "Close", "High", "Low", "Trades"]]

def show\_dashboard():

st.title("📊 Multi-Company Stock Dashboard")

selected\_company = st.selectbox("Select a Company", list(COMPANIES.keys()))

ticker = COMPANIES[selected\_company]

df = fetch\_stock\_data(ticker)

if df is not None:

st.subheader(f"📈 {selected\_company} Stock Data")

st.dataframe(df, use\_container\_width=True)

st.line\_chart(df.set\_index("date")[["Open", "Close", "High", "Low"]])

st.bar\_chart(df.set\_index("date")["Volume"])

else:

st.stop()

**4️⃣ app.py — Main Entry Point**

python

import streamlit as st

from auth import load\_authenticator

from dashboard import show\_dashboard

from logger import initialize\_log\_file, inject\_logger\_script, log\_visitor, show\_logs

# --- Setup ---

initialize\_log\_file()

inject\_logger\_script()

log\_visitor()

# --- Authentication ---

authenticator = load\_authenticator()

name, authentication\_status, username = authenticator.login("Login", "sidebar")

if authentication\_status is False:

st.error("❌ Incorrect username or password")

elif authentication\_status is None:

st.warning("🔐 Please log in to continue")

elif authentication\_status:

authenticator.logout("Logout", "sidebar")

st.sidebar.success(f"✅ Logged in as {name}")

# --- Navigation ---

st.sidebar.title("📌 Navigation")

page = st.sidebar.radio("Go to:", ["📊 Stock Dashboard", "👥 Visitor Logs"])

if page == "📊 Stock Dashboard":

show\_dashboard()

elif page == "👥 Visitor Logs":

st.title("👥 Visitor Logs")

show\_logs()

Would you like me to generate a requirements.txt for this setup or help you refactor the JavaScript logger into a more dynamic format?

Yes

Great! Here's everything you need to complete your modular setup, Jagdev — including a clean requirements.txt and a more dynamic version of your JavaScript logger.

**📦 requirements.txt**

This file lists all the Python dependencies your app needs:

streamlit==1.35.0

streamlit-authenticator==0.2.3

PyYAML==6.0.1

pandas==2.2.2

requests==2.31.0

✅ Tip: Run pip install -r requirements.txt to install everything at once.

**🧠 logger.js — Dynamic Visitor Logger**

Here’s a simple script that collects device info and geolocation, then redirects with query parameters:

html

<script>

(async function() {

function getDeviceType() {

const ua = navigator.userAgent;

if (/mobile/i.test(ua)) return "Mobile";

if (/tablet/i.test(ua)) return "Tablet";

return "Desktop";

}

function redirectWithParams(device, lat, lon) {

const url = new URL(window.location.href);

url.searchParams.set("device", device);

url.searchParams.set("latitude", lat);

url.searchParams.set("longitude", lon);

window.location.replace(url.toString());

}

const device = getDeviceType();

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(

pos => {

const lat = pos.coords.latitude.toFixed(6);

const lon = pos.coords.longitude.toFixed(6);

redirectWithParams(device, lat, lon);

},

err => {

console.warn("Geolocation failed:", err.message);

redirectWithParams(device, "N/A", "N/A");

},

{ timeout: 5000 }

);

} else {

redirectWithParams(device, "N/A", "N/A");

}

})();

</script>

📁 Save this as logger.js in the root of your project.

**✅ Final Checklist**

| **Task** | **Status** |
| --- | --- |
| Modularized code into auth.py, dashboard.py, logger.py | ✅ |
| Main app logic in app.py | ✅ |
| Visitor logging with device + location | ✅ |
| Safe CSV handling | ✅ |
| requirements.txt for easy setup | ✅ |
| JavaScript logger for client-side tracking | ✅ |

Would you like me to help you add analytics (e.g., daily visitor counts or device distribution) to the Visitor Logs page next?