There will be two files:

1. streamlit\_app.py
2. stock\_data\_updator.py

streamlit\_app.py will work as below:

1. Accepts user input of stock symbol
2. Checks if stock symbol found in stock\_list table:
   1. If found: It means we have inference in DB. Fetch and show prediction
   2. If not found:
      1. streamlit\_app.py will call data\_pipeline function of stock\_data\_updator.py
      2. data\_pipeline function include below modules(.py):
         1. data\_Processor module: it consist of below functions:
            1. validate\_symbol : this will check symbol in yf.

If found, it will return True, and call data\_collector function

If not found: It will display error msg to user in streamlit\_app.py to reenter valid symbol

* + - * 1. data\_collector: It will fetch date and close price of that stock for 1 year of interval of 1d.
        2. feature\_engineer: It will add SMA\_14, EMA\_14, RSI\_14, MACD, close\_lag1 to 7
        3. target creation: it will add next 7 days price, average of next 7 days price, trend.
      1. db\_connector: All DB tasks (check, insert, update)
      2. model\_trainer.py: Train + save .pkl, report accuracy
      3. Calls inference.py: Load .pkl + run predictions

stock\_data\_updator.py will be mainly for updating stock on daily basis:

It include data\_pipeline function: This will one of the input as: if new stock, start from new or just fetch today’s close price and train model.

1. data\_pipeline function include below modules(.py):
   1. data\_Processor module: it consist of below functions:
      1. validate\_symbol : this will check symbol in yf.
         1. If found, it will return True, and call data\_collector function
      2. data\_collector: It will fetch date and close price of that stock for 1 year of interval of 1d.
      3. feature\_engineer: It will add SMA\_14, EMA\_14, RSI\_14, MACD, close\_lag1 to 7
      4. target creation: it will add next 7 days price, average of next 7 days price, trend.
   2. db\_connector: All DB tasks (check, insert, update)
   3. model\_trainer.py: Train + save .pkl, report accuracy
   4. inference.py: Load .pkl + run predictions

🔥 **Arbaz, this is an excellent architecture!**

You’ve designed a clean, modular, **industry-style pipeline** where:  
✅ Streamlit focuses only on UI and calling backend functions  
✅ The backend (stock\_data\_updator.py) handles all the heavy data lifting  
✅ Everything is modular and reusable

Let me **review + refine** your design so it’s clean, scalable, and easy to maintain.

**🚀 Refined Process Architecture**

**📦 Main Files**

✅ **1️⃣ streamlit\_app.py (User-facing UI)**

* Accepts **user input** (stock symbol)
* Checks stock\_list table in the database:
  + **If found** → assume .pkl already exists → call inference.py to run predictions
  + **If not found** → call data\_pipeline(new\_stock=True) from stock\_data\_updator.py

✅ **Responsibilities:**

* UI only (no data processing)
* Triggers backend when needed

✅ **2️⃣ stock\_data\_updator.py (Backend engine)**

* Exposes **data\_pipeline function**
  + new\_stock=True → builds everything from scratch (1-year data, features, targets, DB, .pkl)
  + new\_stock=False → only fetches today’s data, updates DB, retrains model

✅ **Modules used:**

* data\_processor.py → handles:
  + validate\_symbol() → check if yfinance returns data
  + data\_collector() → download raw 1-year data or today’s row
  + feature\_engineer() → compute indicators, lags
  + target\_creator() → create target columns (next 7 days, avg, trend)
* db\_connector.py → check, insert, update in Supabase DB
* model\_trainer.py → train + save .pkl, return accuracy
* inference.py → load .pkl + run predictions when called

**✨ Refinements I Suggest**

**🔹 1. Separate Concerns Strictly**

✅ streamlit\_app.py → never runs processing logic directly; it only:

* Calls backend functions
* Shows user feedback (spinners, messages)

✅ stock\_data\_updator.py → never touches UI; it only:

* Runs data/DB/model processes
* Returns success/fail signals

**🔹 2. Centralize All DB Interactions**

✅ All DB tasks should be in db\_connector.py  
Even if streamlit\_app.py just needs to check stock presence, it should call:

python

CopyEdit

from db\_connector import check\_stock\_in\_list

✅ Keeps db\_connector the **only layer** that knows DB structure.

**🔹 3. Modular data\_processor.py**

✅ Functions:

* validate\_symbol(symbol)
* collect\_1year\_data(symbol)
* collect\_today\_data(symbol)
* engineer\_features(df)
* create\_targets(df)

✅ Keeps stock\_data\_updator.py small → just orchestration.

**🔹 4. Model Management**

✅ .pkl models stored per stock in Google Drive (or local, if testing)  
✅ model\_trainer.py handles:

* Training
* Saving .pkl
* Uploading (returns cloud URL)
* Reporting accuracy

✅ inference.py handles:

* Downloading .pkl on demand
* Running predictions on fresh input

**🏗 Final Process Map**

**📦 streamlit\_app.py**

| **Step** | **Module Called** |
| --- | --- |
| Input stock symbol | User input in Streamlit |
| Check in DB | db\_connector.py |
| If new stock | stock\_data\_updator.data\_pipeline(new\_stock=True) |
| If existing stock | inference.py |
| Show predictions | Streamlit charts + messages |

**📦 stock\_data\_updator.py**

| **Step** | **Module Called** |
| --- | --- |
| Validate symbol | data\_processor.py |
| Collect + process data | data\_processor.py |
| Insert/update DB tables | db\_connector.py |
| Train + save model | model\_trainer.py |
| Upload .pkl + update meta | model\_trainer.py, db\_connector.py |

**✅ Summary of Refinements**

✅ Your plan is excellent  
✅ I just recommend:

* Strict separation of **UI ↔ backend**
* Strict separation of **DB logic ↔ model logic**
* Modular code so you can easily swap pieces later (e.g., replace yfinance, add new features)