As we were having difficulty with web scraping , here's the new idea for the bench marking program: Project Title: “Meeting Summary & Action Plan Generator” Core Idea: User uploads a meeting transcript (.txt or .pdf). Two agents coordinate: Agent A (The Summarizer): Reads the meeting transcript and generates a concise summary. Agent B (The Strategist): Reads Agent A’s summary and creates an action plan — assigning tasks, setting deadlines, and highlighting priorities. Why It’s a Great Fit No need for scraping or external data. Everything is local and file-based. You can use GPT-2 or GPT-3 locally or via HuggingFace. Showcases clear multi-agent collaboration. Agentic Architecture Agents communicate via messages or shared memory. Agent A outputs the summary → Agent B picks it up and creates the action plan. Add memory, reasoning, or self-correction loops later if needed. Microservice Architecture Upload Service: Takes .txt file input Summarizer Service: Generates the summary (could use transformers like T5) Planner Service: Takes summary and outputs structured tasks (can fine-tune or use zero-shot prompt) Frontend: Minimal UI with upload button and result view Example Output Input File: Transcript from a team meeting Agent A Output (Summary): Discussed project deadlines, client feedback, and backend integration issues. Agent B Output (Action Plan): Alice: Refactor backend API by April 3 Bob: Implement client feedback UI by April 6 Team: Weekly sync scheduled for Friday

**ChatGPT said:**

Memory updated

This is a fantastic pivot, Rishabh — elegant, impactful, and totally self-contained. Here's a quick breakdown to help you start implementing:

**✅ Architecture Breakdown**

**1. Frontend**

* Upload .txt or .pdf
* Display Summary & Action Plan

**2. Microservices**

| **Service** | **Role** |
| --- | --- |
| **Upload Service** | Accepts file, parses to plain text using PyMuPDF or pdfplumber for PDFs |
| **Summarizer (Agent A)** | Uses T5/GPT-2 to generate a concise summary of the transcript |
| **Planner (Agent B)** | Converts the summary into an action plan (zero-shot prompt or template-based) |

**3. Agentic Interaction**

* Agent A writes to a shared location (summary.json)
* Agent B reads the summary and outputs a structured plan (plan.json)
* Optional: Use a memory.json for multi-round improvements or reflection.

**🧠 LLM Suggestions**

* **Summarizer**: t5-base, facebook/bart-large-cnn, or GPT-2 with summarization prompts
* **Planner**: GPT-2/GPT-3 with a tailored prompt like:

vbnet

CopyEdit

You're a meeting planner. Given this summary, generate an action plan assigning responsibilities, deadlines, and priorities:

Summary:

"Discussed project deadlines, client feedback, and backend integration issues."

Action Plan:

1. Alice: Refactor backend API by April 3

2. Bob: Implement client feedback UI by April 6

3. Team: Weekly sync scheduled for Friday

**🚀 Next Steps**

1. Want help setting up the **FastAPI microservices**?
2. Should I give you a **base directory structure with example files**?
3. Want a sample **PDF-to-text parser** module?

Let me know how you want to proceed — I can also help design the prompt templates and agent loop logic if you're planning to scale this later.

**You said:**

Now , if you remember : the concept of "functionally similar" ...Plan is to develop 2 systems: one with microservices , one with agentic ai arch....these 2 are functionally similar in the sense that they will implement this same idea ...As of now we need to focus on microservice arch: Microservice Architecture Upload Service: Takes .txt file input Summarizer Service: Generates the summary (could use transformers like T5) Planner Service: Takes summary and outputs structured tasks (can fine-tune or use zero-shot prompt) Frontend: Minimal UI with upload button and result view Forget the agentic part for now

**ChatGPT said:**

Perfect — yes, I remember the “functionally similar” concept. You're now focusing **only on the microservice architecture**, which will serve as the baseline implementation for the *Meeting Summary & Action Plan Generator*.

**✅ Microservice Architecture Plan (V1)**

We'll break it down into **4 independent microservices**:

**🟩 1. Upload Service**

* **Input**: .txt or .pdf file
* **Output**: Raw text string
* **Tools**:
  + PyMuPDF or pdfplumber for PDFs
  + FastAPI for the API
* **Endpoint**: /upload
* **Returns**: JSON { "text": "..." }

**🟨 2. Summarizer Service**

* **Input**: Raw text
* **Output**: Summary string
* **Tools**:
  + transformers library (e.g. t5-base, bart-large-cnn)
  + Pre-trained zero-shot
* **Endpoint**: /summarize
* **Returns**: JSON { "summary": "..." }

**🟦 3. Planner Service**

* **Input**: Summary
* **Output**: Action plan (structured list)
* **Method**: Prompt GPT-2 (or zero-shot using BART) with fixed prompt templates
* **Endpoint**: /plan
* **Returns**:

json

CopyEdit

{

"action\_plan": [

{ "person": "Alice", "task": "Refactor backend API", "deadline": "April 3" },

{ "person": "Bob", "task": "Implement UI", "deadline": "April 6" }

]

}

**🟥 4. Frontend (Optional Initially)**

* **Tech**: Streamlit, React, or simple HTML + JS
* **Features**:
  + Upload button (calls /upload)
  + Displays summary (from /summarize)
  + Displays action plan (from /plan)

**🛠 Suggested Directory Structure**

vbnet

CopyEdit

microservice\_app/

│

├── upload\_service/

│ └── main.py (FastAPI + pdf/text parser)

│

├── summarizer\_service/

│ └── main.py (T5 model inference)

│

├── planner\_service/

│ └── main.py (Prompt-based logic)

│

├── frontend/ (optional for now)

│ └── app.html / app.py (if using Streamlit)

│

├── shared/

│ └── utils.py (shared parsing or model loading logic)

│

└── docker-compose.yml (for orchestration later)

**🚀 Ready to Move Forward?**

I can help you with:

1. **The complete upload\_service using FastAPI** (including PDF/text handling)
2. **Summarizer microservice with t5-base**
3. **Planner microservice using GPT-2 prompt template**
4. A simple **test script** that calls all services in sequence

What would you like to start with — Upload, Summarizer, or Planner service?

**You said:**

microservice\_app/ │ ├── upload\_service/ │ └── main.py (FastAPI + pdf/text parser) │ ├── summarizer\_service/ │ └── main.py (T5 model inference) │ ├── planner\_service/ │ └── main.py (Prompt-based logic) │ ├── frontend/ (optional for now) │ └── app.html / app.py (if using Streamlit) │ ├── shared/ │ └── utils.py (shared parsing or model loading logic) │ └── docker-compose.yml (for orchestration later) give a .bat file to setup this structure

**ChatGPT said:**

Here’s a Windows .bat script to **set up the directory structure** and placeholder files as per your microservice\_app plan:

**📄 setup\_structure.bat**

bat

CopyEdit

@echo off

REM Create root folder

mkdir microservice\_app

cd microservice\_app

REM Create service folders and files

mkdir upload\_service

echo # Upload Service - FastAPI + PDF/Text Parser > upload\_service\main.py

mkdir summarizer\_service

echo # Summarizer Service - T5 Inference > summarizer\_service\main.py

mkdir planner\_service

echo # Planner Service - Prompt-based Planning > planner\_service\main.py

REM Optional frontend

mkdir frontend

echo # Frontend Placeholder > frontend\app.html

REM Shared utils folder

mkdir shared

echo # Shared Utility Functions > shared\utils.py

REM Docker Compose setup

echo # Docker Compose file for microservices > docker-compose.yml

echo.

echo ✅ Microservice architecture structure created successfully!

pause

**🛠 How to Use**

1. Save the above code into a file named setup\_structure.bat
2. Double-click the file or run it in cmd inside your desired location.

Let me know if you want to auto-populate the files with **starter code** for each microservice.

**You said:**

Good, now code for this:Upload Service: Takes .txt file input

**ChatGPT said:**

Here’s the complete code for the **Upload Service** using **FastAPI**, which handles both .txt and .pdf file uploads, parses the text, and returns it as JSON.

**📁 upload\_service/main.py**

python

CopyEdit

from fastapi import FastAPI, UploadFile, File, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from PyPDF2 import PdfReader

import os

app = FastAPI()

# Allow frontend or other services to call this

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

@app.post("/upload/")

async def upload\_file(file: UploadFile = File(...)):

if not file.filename.endswith(('.txt', '.pdf')):

raise HTTPException(status\_code=400, detail="Only .txt or .pdf files are allowed.")

contents = ""

if file.filename.endswith('.txt'):

contents = (await file.read()).decode('utf-8')

elif file.filename.endswith('.pdf'):

try:

temp\_path = f"temp\_{file.filename}"

with open(temp\_path, "wb") as f:

f.write(await file.read())

reader = PdfReader(temp\_path)

for page in reader.pages:

contents += page.extract\_text() or ""

os.remove(temp\_path)

except Exception as e:

raise HTTPException(status\_code=500, detail=f"PDF parsing failed: {str(e)}")

return {"text": contents.strip()}

**🧪 How to Run It**

bash

CopyEdit

cd upload\_service

uvicorn main:app --reload --port 8000

* **Endpoint**: POST /upload/
* **Form Field**: file
* **Returns**: {"text": "..."}

**✅ Features**

* Accepts .txt and .pdf
* Uses PyPDF2 (you can switch to pdfplumber for more complex PDFs)
* Temporary PDF file is cleaned up
* Handles UTF-8 encoding

Want the corresponding **cURL** or Python requests example to test it?

**You said:**

🧪 How to Run It bash Copy Edit cd upload\_service uvicorn main:app --reload --port 8000 Endpoint: POST /upload/ Form Field: file Returns: {"text": "..."} About this : how about we quickly write the code for the other services and then test them, I know that you are being prudent and I appreciate it (Normally I prefer unit testing too as a developer ) but as this is a small project and I need to finish up quickly...so...

**ChatGPT said:**

Got it — speed mode ON ⚡

Let's write all microservices **one by one** as minimal working FastAPI apps, ready to plug and play.

**✅ 1. upload\_service/main.py**

(Already done above — handles .txt and .pdf uploads)

**✅ 2. summarizer\_service/main.py**

python

CopyEdit

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from transformers import pipeline

app = FastAPI()

summarizer = pipeline("summarization", model="t5-small") # or "facebook/bart-large-cnn"

class TextInput(BaseModel):

text: str

@app.post("/summarize/")

def summarize\_text(data: TextInput):

try:

result = summarizer(data.text, max\_length=100, min\_length=30, do\_sample=False)

return {"summary": result[0]["summary\_text"]}

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

**✅ 3. planner\_service/main.py**

python

CopyEdit

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from transformers import pipeline

app = FastAPI()

generator = pipeline("text-generation", model="gpt2")

class SummaryInput(BaseModel):

summary: str

@app.post("/plan/")

def generate\_plan(data: SummaryInput):

prompt = (

"You are a project strategist. Based on the following meeting summary, create an action plan with assigned tasks and deadlines.\n\n"

f"Meeting Summary:\n{data.summary}\n\nAction Plan:\n"

)

try:

response = generator(prompt, max\_length=150, num\_return\_sequences=1)

return {"action\_plan": response[0]["generated\_text"].split("Action Plan:")[-1].strip()}

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

**✅ Requirements File for All Services (Optional Shared)**

Create a requirements.txt in each folder or a global one:

nginx

CopyEdit

fastapi

uvicorn

transformers

torch

PyPDF2

pydantic

**⚙️ Run Each Service**

From each folder:

bash

CopyEdit

uvicorn main:app --reload --port <PORT>

| **Service** | **Port** |
| --- | --- |
| Upload | 8000 |
| Summarizer | 8001 |
| Planner | 8002 |

Do you want me to also:

* Give you a **test script** that chains these 3 services end-to-end?
* Write the docker-compose.yml to spin up all 3 instantly?

Let me know — you're super close to done.

**You said:**

You know what to do ;) --------> Create a bat file for this: ⚙️ Run Each Service From each folder: bash Copy Edit uvicorn main:app --reload --port <PORT> Service Port Upload 8000 Summarizer 8001 Planner 8002

**ChatGPT said:**

Absolutely 😎 — here's a .bat file to **run all 3 services in parallel** using separate terminals for each.

**📄 run\_services.bat**

bat

CopyEdit

@echo off

echo Starting Upload Service on port 8000...

start cmd /k "cd upload\_service && uvicorn main:app --reload --port 8000"

timeout /t 2 > nul

echo Starting Summarizer Service on port 8001...

start cmd /k "cd summarizer\_service && uvicorn main:app --reload --port 8001"

timeout /t 2 > nul

echo Starting Planner Service on port 8002...

start cmd /k "cd planner\_service && uvicorn main:app --reload --port 8002"

echo ✅ All services started in separate terminals.

pause

**✅ What It Does**

* Opens **3 new command windows** (one for each service)
* Sets correct directory and launches uvicorn
* Adds a **2-second delay** between launches to avoid overlap

You can now double-click run\_services.bat from your root microservice\_app folder to launch everything.

Let me know if you want a .bat for shutdown, or a Python script to test the full chain (upload → summarize → plan)!

**You said:**

are bat files not recognised by powershell?

**ChatGPT said:**

Great question!

**💡 Quick Answer:**

Yes, **.bat files do work in PowerShell**, **but** there are a few things to keep in mind:

**✅ How to Run a .bat file in PowerShell**

You must **prefix the path with .\** if you're in the same directory:

powershell

CopyEdit

.\run\_services.bat

If that doesn’t work, try:

powershell

CopyEdit

Start-Process .\run\_services.bat

**⚠️ Common Issues in PowerShell**

| **Issue** | **Solution** |
| --- | --- |
| .bat opens but closes instantly | Add a pause at the end (which I already included) |
| Script execution restricted | Run: Set-ExecutionPolicy RemoteSigned -Scope CurrentUser |
| Typing run\_services.bat does nothing | Must use .\ in PowerShell |

**✅ Pro Tip**

If you’re using VSCode’s terminal (PowerShell by default), just right-click on run\_services.bat in the file explorer and hit **"Run in Command Prompt"** for guaranteed behavior.

Let me know if you want a .ps1 version of the script rewritten specifically for PowerShell!

**You said:**

yes .ps1 would be great

**ChatGPT said:**

Absolutely! Here's your PowerShell version of the script to start all three FastAPI services in **separate PowerShell terminals**.

**📄 run\_services.ps1**

powershell

CopyEdit

Write-Host "🚀 Starting Upload Service on port 8000..."

Start-Process powershell -ArgumentList 'cd upload\_service; uvicorn main:app --reload --port 8000'

Start-Sleep -Seconds 2

Write-Host "🚀 Starting Summarizer Service on port 8001..."

Start-Process powershell -ArgumentList 'cd summarizer\_service; uvicorn main:app --reload --port 8001'

Start-Sleep -Seconds 2

Write-Host "🚀 Starting Planner Service on port 8002..."

Start-Process powershell -ArgumentList 'cd planner\_service; uvicorn main:app --reload --port 8002'

Write-Host "✅ All services started in separate PowerShell windows."

**✅ How to Run**

1. Save it as run\_services.ps1 in your microservice\_app/ directory.
2. Open PowerShell and **navigate to that folder**.
3. Run:

powershell

CopyEdit

.\run\_services.ps1

**🛡 If You Hit Execution Policy Issues**

Run this once:

powershell

CopyEdit

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Then confirm with Y.

Let me know if you'd like a PowerShell script to stop all running services or test them in sequence too!

**You said:**

PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app> .\run\_services.ps1 ðŸš€ Starting Upload Service on port 8000... ðŸš€ Starting Summarizer Service on port 8001... ðŸš€ Starting Planner Service on port 8002... âœ… All services started in separate PowerShell windows. PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app>

**ChatGPT said:**

Perfect execution! 🎯

Looks like your PowerShell script ran smoothly — all three services should now be live on:

* **Upload Service** → http://127.0.0.1:8000/docs
* **Summarizer Service** → http://127.0.0.1:8001/docs
* **Planner Service** → http://127.0.0.1:8002/docs

You can test them individually using the Swagger UI (/docs) provided by FastAPI.

**✅ What’s Next?**

Would you like me to:

1. 🧪 Give you a **Python test script** that:
   * Uploads a .txt file to port 8000
   * Sends it to summarizer on port 8001
   * Passes that to planner on port 8002
   * Prints the final action plan?
2. 🐳 Or help you dockerize these microservices using docker-compose.yml?

You’re in the final stretch — just say the word.

**You said:**

actually i expected you to connect them using a front end and then test

**ChatGPT said:**

Ah, got it — you want to wire them together with a **minimal frontend** to test the entire pipeline in a smooth UI. Let’s do this properly.

**✅ Goal: End-to-End Frontend with Upload → Summary → Action Plan**

We'll use **Streamlit** for a quick and clean interface.

**📁 frontend/app.py**

python

CopyEdit

import streamlit as st

import requests

st.set\_page\_config(page\_title="Meeting Summary & Action Plan Generator")

st.title("📄 Meeting Summary & Action Plan Generator")

uploaded\_file = st.file\_uploader("Upload a .txt or .pdf meeting transcript", type=["txt", "pdf"])

if uploaded\_file is not None:

st.success("File uploaded. Sending to Upload Service...")

files = {"file": (uploaded\_file.name, uploaded\_file, uploaded\_file.type)}

upload\_response = requests.post("http://127.0.0.1:8000/upload/", files=files)

if upload\_response.status\_code == 200:

raw\_text = upload\_response.json()["text"]

st.subheader("📝 Raw Transcript:")

st.text\_area("", raw\_text, height=200)

if st.button("Generate Summary"):

st.info("Sending to Summarizer Service...")

summary\_response = requests.post("http://127.0.0.1:8001/summarize/", json={"text": raw\_text})

if summary\_response.status\_code == 200:

summary = summary\_response.json()["summary"]

st.subheader("🧠 Summary:")

st.text\_area("", summary, height=150)

st.info("Sending to Planner Service...")

plan\_response = requests.post("http://127.0.0.1:8002/plan/", json={"summary": summary})

if plan\_response.status\_code == 200:

action\_plan = plan\_response.json()["action\_plan"]

st.subheader("🗂️ Action Plan:")

st.text\_area("", action\_plan, height=200)

else:

st.error("Planner service error: " + plan\_response.text)

else:

st.error("Summarizer service error: " + summary\_response.text)

else:

st.error("Upload service error: " + upload\_response.text)

**📦 Install Streamlit (once)**

bash

CopyEdit

pip install streamlit

**▶️ Run the App**

bash

CopyEdit

cd frontend

streamlit run app.py

It will launch in your browser at http://localhost:8501.

**✅ What It Does**

* Uploads file → calls **Upload Service**
* Gets raw text → calls **Summarizer**
* Gets summary → calls **Planner**
* Displays all results in a clean UI

Let me know if you want:

* Buttons to save results
* Markdown formatting for action plan
* Dockerizing the frontend too

You’re almost demo-ready 💼✅

**You said:**

now first kill all processes running because of run\_services.ps1

**ChatGPT said:**

To cleanly **kill all running FastAPI services** started by run\_services.ps1, you can use PowerShell to terminate all uvicorn processes.

**✅ Run this in PowerShell:**

powershell

CopyEdit

Get-Process -Name "python" | Where-Object { $\_.Path -like "\*uvicorn\*" } | Stop-Process -Force

However, in many setups, uvicorn runs as a Python child process, so this command is more reliable:

powershell

CopyEdit

Get-Process -Name "python" | Stop-Process -Force

⚠️ This will stop **all Python processes** — including Jupyter or other apps. Use only if you're okay with that.

**🧪 Safer Option (Filter by Command Line)**

You can also search only for uvicorn:

powershell

CopyEdit

Get-CimInstance Win32\_Process | Where-Object { $\_.CommandLine -like "\*uvicorn\*" } | ForEach-Object { Stop-Process -Id $\_.ProcessId -Force }

This will stop only those Python processes running uvicorn.

Let me know if you want a .ps1 shutdown script version of this!

**You said:**

now that we have the front end , how to run , give ps1 file if we will have to run the services separately and include running streamlit app in the .ps1 as well

**ChatGPT said:**

Perfect — here's your all-in-one **PowerShell script (start\_app.ps1)** to:

1. Start each microservice in its own terminal window.
2. Wait a bit between launches.
3. Finally start the **Streamlit frontend app**.

**📄 start\_app.ps1**

powershell

CopyEdit

Write-Host "🚀 Starting Upload Service on port 8000..."

Start-Process powershell -ArgumentList 'cd upload\_service; uvicorn main:app --reload --port 8000'

Start-Sleep -Seconds 2

Write-Host "🚀 Starting Summarizer Service on port 8001..."

Start-Process powershell -ArgumentList 'cd summarizer\_service; uvicorn main:app --reload --port 8001'

Start-Sleep -Seconds 2

Write-Host "🚀 Starting Planner Service on port 8002..."

Start-Process powershell -ArgumentList 'cd planner\_service; uvicorn main:app --reload --port 8002'

Start-Sleep -Seconds 2

Write-Host "🌐 Launching Streamlit Frontend..."

Start-Process powershell -ArgumentList 'cd frontend; streamlit run app.py'

Write-Host "✅ All services and frontend started."

**✅ How to Use**

1. Save as start\_app.ps1 in your microservice\_app/ directory.
2. Open PowerShell and run:

powershell

CopyEdit

.\start\_app.ps1

**🔁 Optional: Want a stop\_app.ps1 script?**

Let me know — I can give you one to **cleanly kill all uvicorn + streamlit processes** in one shot.

You're fully app-ready now 🔥

**You said:**

Want a stop\_app.ps1 script? yes,please

**ChatGPT said:**

Perfect! Here's your **stop\_app.ps1** script to cleanly stop all services launched by start\_app.ps1, including:

* All uvicorn instances (your microservices)
* The streamlit frontend app

**📄 stop\_app.ps1**

powershell

CopyEdit

Write-Host "🛑 Stopping all running uvicorn services..."

Get-CimInstance Win32\_Process | Where-Object { $\_.CommandLine -like "\*uvicorn\*" } | ForEach-Object { Stop-Process -Id $\_.ProcessId -Force }

Write-Host "🛑 Stopping streamlit frontend..."

Get-CimInstance Win32\_Process | Where-Object { $\_.CommandLine -like "\*streamlit run\*" } | ForEach-Object { Stop-Process -Id $\_.ProcessId -Force }

Write-Host "✅ All services and frontend have been stopped."

**✅ How to Use**

1. Save this as stop\_app.ps1 in your microservice\_app/ directory.
2. Open PowerShell and run:

powershell

CopyEdit

.\stop\_app.ps1

Let me know if you'd like a .bat version too, or if you want to log which processes were killed. You're now fully automated for development! 💻✅

**You said:**

generate a txt to test it

**ChatGPT said:**

Here’s a simple **test transcript file** in .txt format you can use to test the full pipeline:

**📄 test\_meeting.txt**

vbnet

CopyEdit

Team Meeting - March 30, 2025

Attendees: Alice, Bob, Clara, David

- Discussed feedback received from the client regarding the user interface.

- Alice mentioned the backend API is causing delays and may need refactoring.

- Bob volunteered to handle UI improvements based on the client's suggestions.

- Clara raised concerns about the upcoming release deadline and testing coverage.

- The team agreed to conduct a weekly sync every Friday at 10 AM.

- David suggested integrating logging and monitoring tools in the next sprint.

Next Steps:

- Assign tasks for UI, backend, testing, and observability.

- Track deadlines for all major components.

**🧪 How to Test**

1. Place this file in any accessible folder.
2. Upload it via the **Streamlit frontend** you just built.
3. You should get:
   * Summary from Summarizer Service
   * Action Plan from Planner Service

Let me know if you want a PDF version of this file too for testing the PDF parser!

**You said:**

Could you generate txt such that they are talking to each other as its a transcript

**ChatGPT said:**

Absolutely! Here's a more realistic **meeting transcript in conversational format**, written like people are talking to each other during the meeting.

**📄 team\_meeting\_transcript.txt**

vbnet

CopyEdit

Meeting Transcript – April 1, 2025

Alice: Alright team, let’s get started. First on the agenda — the client feedback on our latest demo. Bob, you’ve looked into it?

Bob: Yes, I have. The client wants the dashboard to be more responsive and suggested adding a dark mode option.

Clara: That makes sense. We’ve had similar feedback from internal testers too. Who’s taking up the UI changes?

Bob: I can handle that. I’ll start refactoring the frontend components this week.

David: Great. While you’re on that, I’ll work on integrating monitoring tools into the backend. We’ve had some performance dips lately.

Alice: Thanks, David. Also, the backend API latency is still a concern. I think we need to profile and optimize those endpoints.

Clara: I can take a look at that. I’ll run some load tests and identify bottlenecks.

Alice: Perfect. Let’s aim to wrap all this before next Friday’s internal review. How about we schedule a sync every Friday at 11 AM?

Team: Agreed.

David: One more thing — do we need to prepare a brief report for the client summarizing what we’ve addressed?

Alice: Yes, good point. Bob, after your UI changes, can you document the updates and send the report?

Bob: Sure, will do.

Alice: Great work, everyone. Let’s stay on track and follow up on progress during our weekly sync.

**📦 Save As**

* File name: team\_meeting\_transcript.txt
* Format: UTF-8 .txt

This is *much closer to a real transcript*, and your system should now:

* Extract a clean summary
* Generate an action plan with assigned tasks & deadlines

Let me know if you want a longer/messier version to stress test!

**You said:**

requests.exceptions.ConnectionError: HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: /upload/ (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x00000275399B46E0>: Failed to establish a new connection: [WinError 10061] No connection could be made because the target machine actively refused it')) Traceback: File "C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\frontend\app.py", line 15, in <module> upload\_response = requests.post("http://127.0.0.1:8000/upload/", files=files) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 115, in post return request("post", url, data=data, json=json, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 59, in request return session.request(method=method, url=url, \*\*kwargs) ~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 589, in request resp = self.send(prep, \*\*send\_kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 703, in send r = adapter.send(request, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\adapters.py", line 700, in send raise ConnectionError(e, request=request)

**ChatGPT said:**

Thanks for sharing the traceback — the error you're seeing:

Failed to establish a new connection: [WinError 10061] No connection could be made because the target machine actively refused it

means that the **Upload Service (port 8000)** was **not running** when your frontend tried to send the file.

**✅ Here's How to Fix It**

**1. Start the Upload Service Manually**

Open a new PowerShell terminal and run:

bash

CopyEdit

cd upload\_service

uvicorn main:app --reload --port 8000

Make sure it shows:

nginx

CopyEdit

Uvicorn running on http://127.0.0.1:8000

You can also test in the browser:  
Go to 👉 http://127.0.0.1:8000/docs  
If the Swagger UI opens, the service is up.

**2. Or Run Everything via PowerShell Script**

If you prefer automation, re-run:

powershell

CopyEdit

.\start\_app.ps1

This launches:

* Upload service on port 8000
* Summarizer on 8001
* Planner on 8002
* Streamlit frontend

Make sure all 4 terminals stay open and show “running.”

**❗ Still Not Working?**

Check:

* You don't have another process using port 8000
* Your main.py is error-free
* You're not closing the window where uvicorn is running

Let me know if you want a **health-check script** to verify all 3 services before launching the frontend!

**You said:**

PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\upload\_service> Uvicorn running on http://127.0.0.1:8000 Uvicorn : The term 'Uvicorn' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:1 + Uvicorn running on http://127.0.0.1:8000 + ~~~~~~~ + CategoryInfo : ObjectNotFound: (Uvicorn:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\upload\_service> PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\upload\_service>

**ChatGPT said:**

Ah I see — you're trying to **run a status message directly**, not the actual command.

This line:

nginx

CopyEdit

Uvicorn running on http://127.0.0.1:8000

is a **log message**, not something you run manually.

**✅ What You Should Run Instead**

To start the upload service, **run this exact command** in your PowerShell terminal:

powershell

CopyEdit

uvicorn main:app --reload --port 8000

If that gives you:

'uvicorn' is not recognized as the name of a cmdlet...

Then it means uvicorn is not installed or not in PATH.

**🛠 Fix: Install Uvicorn**

Run:

powershell

CopyEdit

pip install uvicorn

If you're using a virtual environment, make sure it's activated:

powershell

CopyEdit

.\venv\Scripts\activate

Then try again:

powershell

CopyEdit

uvicorn main:app --reload --port 8000

Let me know if you're using a virtual environment or want help setting one up. Once uvicorn runs, the terminal will log:

nginx

CopyEdit

Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

Then you’re all set! ✅

**You said:**

PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\upload\_service> uvicorn main:app --reload --port 8000 uvicorn : The term 'uvicorn' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again. At line:1 char:1 + uvicorn main:app --reload --port 8000 + ~~~~~~~ + CategoryInfo : ObjectNotFound: (uvicorn:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\upload\_service>

**ChatGPT said:**

Thanks for the update — the error confirms that **uvicorn is not installed or not added to PATH** on your system.

Let's fix that step by step:

**✅ Step 1: Install uvicorn**

In your PowerShell terminal, run:

powershell

CopyEdit

pip install uvicorn

If you're using Python via Anaconda or virtualenv, make sure you're in the correct environment.

**✅ Step 2: Confirm Installation**

After installation, check if uvicorn is available:

powershell

CopyEdit

uvicorn --version

If this works, you're good to go.

**⚠️ Still Not Recognized?**

If it's still not recognized:

**Option 1: Use Python to run Uvicorn directly**

powershell

CopyEdit

python -m uvicorn main:app --reload --port 8000

This **always works** even if uvicorn is not in PATH.

**✅ Summary: Run the Upload Service**

powershell

CopyEdit

cd upload\_service

python -m uvicorn main:app --reload --port 8000

This is the safest and recommended way on Windows when uvicorn is not globally accessible.

Let me know once this works — we’ll move to testing the pipeline with Streamlit.

**You said:**

good it worked , now new error : requests.exceptions.ConnectionError: HTTPConnectionPool(host='127.0.0.1', port=8000): Max retries exceeded with url: /upload/ (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x000001FE5E3A46E0>: Failed to establish a new connection: [WinError 10061] No connection could be made because the target machine actively refused it')) Traceback: File "C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\frontend\app.py", line 15, in <module> upload\_response = requests.post("http://127.0.0.1:8000/upload/", files=files) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 115, in post return request("post", url, data=data, json=json, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 59, in request return session.request(method=method, url=url, \*\*kwargs) ~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 589, in request resp = self.send(prep, \*\*send\_kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 703, in send r = adapter.send(request, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\adapters.py", line 700, in send raise ConnectionError(e, request=request)

**ChatGPT said:**

Thanks for the detailed traceback. This error:

[WinError 10061] No connection could be made because the target machine actively refused it

means the frontend still **cannot reach the Upload Service**, even though you started it.

**🔍 Let's Debug Step-by-Step**

**✅ 1. Make sure you're running the service with this:**

bash

CopyEdit

python -m uvicorn main:app --reload --port 8000

⚠️ If you're using python instead of uvicorn, you **must** include -m.

**✅ 2. Check if Upload Service is Actually Running**

Open your browser and go to:

🔗 http://127.0.0.1:8000/docs

* If this opens the FastAPI Swagger UI → ✅ service is running.
* If it doesn't load → ❌ service is still not up.

**✅ 3. Check for Port Conflicts or Errors in Terminal**

* Look at the terminal where you ran the upload\_service.
* If you see errors like "port already in use" or "module not found," the service failed to start.

**✅ 4. Check Your Streamlit Frontend Terminal**

* Make sure it **doesn’t try to make the request immediately on app load**.
* Wrap the requests.post() in a button or file upload block to ensure the service is ready before calling.

**✅ 5. Retry the Working Combo**

Here’s your working stack sequence:

**Terminal 1 (Upload Service):**

bash

CopyEdit

cd upload\_service

python -m uvicorn main:app --reload --port 8000

**Terminal 2 (Frontend):**

bash

CopyEdit

cd frontend

streamlit run app.py

Upload the .txt file in the UI after confirming http://127.0.0.1:8000/docs works in the browser.

Want me to modify your Streamlit code so it waits/checks if the service is running first before sending the request?

**You said:**

wait , i hadnt done pip install -r requirements.txt, let me do that and try rerunning

**ChatGPT said:**

Ah yes — that would explain it. If the required packages like fastapi, uvicorn, or requests weren’t installed, the backend service might silently fail or not start at all.

**✅ Here's the Right Order Now:**

1. **Install dependencies** (from your project root):

bash

CopyEdit

pip install -r requirements.txt

Make sure your requirements.txt has at least:

nginx

CopyEdit

fastapi

uvicorn

requests

pydantic

transformers

torch

streamlit

PyPDF2

If you don’t have a requirements.txt, I can generate one for you based on our setup.

1. **Start Upload Service**

bash

CopyEdit

cd upload\_service

python -m uvicorn main:app --reload --port 8000

Check: http://127.0.0.1:8000/docs in browser.

1. **Start Frontend**

bash

CopyEdit

cd ../frontend

streamlit run app.py

Once done, try uploading your .txt file again from the frontend UI.

Let me know if everything works now or if you want the full working requirements.txt file!

**You said:**

upload worked: error in summerizer microservice: requests.exceptions.ConnectionError: HTTPConnectionPool(host='127.0.0.1', port=8001): Max retries exceeded with url: /summarize/ (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x000001A50AC72850>: Failed to establish a new connection: [WinError 10061] No connection could be made because the target machine actively refused it')) Traceback: File "C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\frontend\app.py", line 24, in <module> summary\_response = requests.post("http://127.0.0.1:8001/summarize/", json={"text": raw\_text}) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 115, in post return request("post", url, data=data, json=json, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\api.py", line 59, in request return session.request(method=method, url=url, \*\*kwargs) ~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 589, in request resp = self.send(prep, \*\*send\_kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\sessions.py", line 703, in send r = adapter.send(request, \*\*kwargs) File "C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\requests\adapters.py", line 700, in send raise ConnectionError(e, request=request)

**ChatGPT said:**

Nice! Since the **upload worked**, it means the frontend is connected correctly to port 8000. Now, the new error:

❌ ConnectionError on port 8001 for /summarize/

means the **summarizer microservice is not running**.

**✅ Fix: Start the Summarizer Service**

In a new terminal (PowerShell or CMD), run:

powershell

CopyEdit

cd summarizer\_service

python -m uvicorn main:app --reload --port 8001

Then test that it’s running by opening in your browser:

🔗 http://127.0.0.1:8001/docs  
(You should see the FastAPI Swagger UI with a /summarize/ POST endpoint.)

**✅ Reminder**

All services need to be running simultaneously:

| **Service** | **Command** | **Port** |
| --- | --- | --- |
| Upload | python -m uvicorn main:app --reload --port 8000 | 8000 |
| Summarizer | python -m uvicorn main:app --reload --port 8001 | 8001 |
| Planner | python -m uvicorn main:app --reload --port 8002 | 8002 |
| Frontend | streamlit run app.py | 8501 |

Let me know if you want a “status check script” that verifies all services before launching the frontend 🚦✅

**You said:**

It was already started but it was taking time to load some stuff: INFO: Will watch for changes in these directories: ['C:\\Users\\risha\\OneDrive\\Desktop\\VCC\_Project\\microservice\_app\\summarizer\_service'] INFO: Uvicorn running on http://127.0.0.1:8001 (Press CTRL+C to quit) INFO: Started reloader process [93344] using StatReload config.json: 100%|████████████████████████████████████████████████████████████████| 1.21k/1.21k [00:00<00:00, 3.59MB/s] C:\Users\risha\AppData\Roaming\Python\Python313\site-packages\huggingface\_hub\file\_download.py:144: UserWarning: huggingface\_hub cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\risha\.cache\huggingface\hub\models--t5-small. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the HF\_HUB\_DISABLE\_SYMLINKS\_WARNING environment variable. For more details, see https://huggingface.co/docs/huggingface\_hub/how-to-cache#limitations. To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development warnings.warn(message) Xet Storage is enabled for this repo, but the 'hf\_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: pip install huggingface\_hub[hf\_xet] or pip install hf\_xet model.safetensors: 100%|████████████████████████████████████████████████████████████| 242M/242M [00:18<00:00, 13.2MB/s] generation\_config.json: 100%|██████████████████████████████████████████████████████████| 147/147 [00:00<00:00, 724kB/s] tokenizer\_config.json: 100%|██████████████████████████████████████████████████████| 2.32k/2.32k [00:00<00:00, 14.2MB/s] spiece.model: 100%|█████████████████████████████████████████████████████████████████| 792k/792k [00:00<00:00, 1.17MB/s] tokenizer.json: 100%|█████████████████████████████████████████████████████████████| 1.39M/1.39M [00:00<00:00, 1.58MB/s] Device set to use cpu INFO: Started server process [90048] INFO: Waiting for application startup. INFO: Application startup complete.

**ChatGPT said:**

Perfect — that log confirms everything's working now! 🎉

**🔍 What Happened?**

* The **summarizer service was downloading the model weights** (t5-small) from Hugging Face.
* This takes some time the **first time only** (models are ~200MB+).
* While that was happening, the port wasn't ready to accept requests → hence the ConnectionError.

**✅ What Now?**

Since you see this line:

makefile

CopyEdit

INFO: Application startup complete.

it means the Summarizer is now **fully running and ready** 💡

**🧪 Next Steps:**

1. Go back to your **Streamlit frontend** at http://localhost:8501
2. Re-upload your .txt transcript
3. Hit **"Generate Summary"** again

➡️ You should now get the summary and move on to the **Planner** service (port 8002).

Let me know if:

* You want to **pre-download models** to avoid delay in demo
* The **Planner service** throws a similar delay or needs preloading too
* You want a loading indicator in Streamlit to handle such delays gracefully

You're *very close* to a full working pipeline 🔁✅

**You said:**

It worked , but is that model loading lag a one time thing or do we need to add a fix for that?

**ChatGPT said:**

I'm glad it worked! 🎉  
Now regarding your question about the **model loading lag** — here’s the complete picture:

**🔄 Model Loading Delay: One-Time or Recurring?**

**✅ It's mostly a one-time thing when:**

* The model is downloaded **for the first time** from Hugging Face (e.g., t5-small, gpt2)
* It is **cached locally** in:

makefile

CopyEdit

C:\Users\<your\_username>\.cache\huggingface

* As long as the cache remains, the loading will be **almost instant** next time.

**⚠️ When you might see delays again:**

| **Situation** | **What Happens** |
| --- | --- |
| You clear .cache folder | Model will be redownloaded |
| You change the model name (e.g. t5-base → t5-small) | New model will be downloaded |
| You're running it on a new device | No cache = full download again |
| You're loading large models like bart-large-cnn or flan-t5-large | It always takes longer, even cached |
| You restart your backend service (cold start) | ~1–2 seconds to load from disk (no download though) |

**✅ Best Practices (Optional Fixes)**

1. **Preload the model at startup** (already happening in our code)
2. Add a **loading spinner** or message in Streamlit when waiting for response:

python

CopyEdit

with st.spinner("Generating summary..."):

response = requests.post(...)

1. (Optional) **Warm up the service** by hitting it with a dummy request at app start

**💡 TL;DR**

* ✅ The long delay is **only the first time**
* ⚙️ After that, model is cached → fast response
* ❄️ If you want, we can add a **startup time check** or **health status badge** in the frontend

Let me know if you'd like a health-check endpoint or an improved prompt for the planner!

**You said:**

with st.spinner("Generating summary..."): response = requests.post(...) Where to place this: # Frontend Placeholder import streamlit as st import requests st.set\_page\_config(page\_title="Meeting Summary & Action Plan Generator") st.title("📄 Meeting Summary & Action Plan Generator") uploaded\_file = st.file\_uploader("Upload a .txt or .pdf meeting transcript", type=["txt", "pdf"]) if uploaded\_file is not None: st.success("File uploaded. Sending to Upload Service...") files = {"file": (uploaded\_file.name, uploaded\_file, uploaded\_file.type)} upload\_response = requests.post("http://127.0.0.1:8000/upload/", files=files) if upload\_response.status\_code == 200: raw\_text = upload\_response.json()["text"] st.subheader("📝 Raw Transcript:") st.text\_area("", raw\_text, height=200) if st.button("Generate Summary"): st.info("Sending to Summarizer Service...") summary\_response = requests.post("http://127.0.0.1:8001/summarize/", json={"text": raw\_text}) if summary\_response.status\_code == 200: summary = summary\_response.json()["summary"] st.subheader("🧠 Summary:") st.text\_area("", summary, height=150) st.info("Sending to Planner Service...") plan\_response = requests.post("http://127.0.0.1:8002/plan/", json={"summary": summary}) if plan\_response.status\_code == 200: action\_plan = plan\_response.json()["action\_plan"] st.subheader("🗂️ Action Plan:") st.text\_area("", action\_plan, height=200) else: st.error("Planner service error: " + plan\_response.text) else: st.error("Summarizer service error: " + summary\_response.text) else: st.error("Upload service error: " + upload\_response.text)

**ChatGPT said:**

Great! You can add the with st.spinner() blocks exactly where you’re calling the external services (summarizer and planner). Here's your **updated Streamlit code** with proper spinners added:

**✅ Updated Code with Spinners**

python

CopyEdit

import streamlit as st

import requests

st.set\_page\_config(page\_title="Meeting Summary & Action Plan Generator")

st.title("📄 Meeting Summary & Action Plan Generator")

uploaded\_file = st.file\_uploader("Upload a .txt or .pdf meeting transcript", type=["txt", "pdf"])

if uploaded\_file is not None:

st.success("File uploaded. Sending to Upload Service...")

files = {"file": (uploaded\_file.name, uploaded\_file, uploaded\_file.type)}

upload\_response = requests.post("http://127.0.0.1:8000/upload/", files=files)

if upload\_response.status\_code == 200:

raw\_text = upload\_response.json()["text"]

st.subheader("📝 Raw Transcript:")

st.text\_area("", raw\_text, height=200)

if st.button("Generate Summary"):

with st.spinner("🧠 Generating summary..."):

summary\_response = requests.post("http://127.0.0.1:8001/summarize/", json={"text": raw\_text})

if summary\_response.status\_code == 200:

summary = summary\_response.json()["summary"]

st.subheader("🧠 Summary:")

st.text\_area("", summary, height=150)

with st.spinner("🗂️ Generating action plan..."):

plan\_response = requests.post("http://127.0.0.1:8002/plan/", json={"summary": summary})

if plan\_response.status\_code == 200:

action\_plan = plan\_response.json()["action\_plan"]

st.subheader("🗂️ Action Plan:")

st.text\_area("", action\_plan, height=200)

else:

st.error("Planner service error: " + plan\_response.text)

else:

st.error("Summarizer service error: " + summary\_response.text)

else:

st.error("Upload service error: " + upload\_response.text)

**💡 Result:**

* You’ll now see a **loading spinner** when:
  + Summary is being generated
  + Action plan is being generated

It’s clean, smooth, and user-friendly ✅  
Let me know if you want to add **progress bars**, **retry buttons**, or **health check indicators** next!

**You said:**

Good job , it worked,I realized that I love microservice architecture because its so easy to debug as everything is in separate components

**ChatGPT said:**

Yesss Rishabh! 🙌 That’s *exactly* the beauty of microservice architecture — glad you experienced it firsthand:

**💡 Why Microservices Feel Great:**

✅ **Isolation**  
Break something? Only *one* service goes down — not the whole app.

✅ **Focused Debugging**  
Logs and errors are clean and localized. Easier to test, fix, and re-deploy just that part.

✅ **Scalability**  
If the summarizer becomes heavy, you can scale **just that container** later.

✅ **Tech Flexibility**  
Want to try a Flan-T5 summarizer in PyTorch and a planner in Rust? You can do that without changing the rest.

✅ **Clean Mental Model**  
You think like a systems architect — each piece has one job and does it well.

**🧠 What You Just Built:**

* A full microservice pipeline 🧱
* NLP + LLM powered backends 🤖
* A UI that ties it all together ⚙️
* Real-world architecture patterns 💼

Want to take it even further with:

* Dockerization? 🐳
* Health check microservice?
* Retry/resilience logic?
* Add a memory module for multi-turn sessions?

Let’s scale this like a pro when you're ready! 🚀

**You said:**

yes, help me complete the docker-compose.yml, also tell me why do we use docker-compose.yml in some cases and Dockerfile in other cases

version: "3.9" services: upload\_service: build: ./upload\_service ports: - "8000:8000" volumes: - ./upload\_service:/app summarizer\_service: build: ./summarizer\_service ports: - "8001:8001" volumes: - ./summarizer\_service:/app planner\_service: build: ./planner\_service ports: - "8002:8002" volumes: - ./planner\_service:/app frontend: build: ./frontend ports: - "8501:8501" volumes: - ./frontend:/app depends\_on: - upload\_service - summarizer\_service - planner\_service in this does version refer to python version? because i am using 3.13 , so should i modify that?

**ChatGPT said:**

​

**You said:**

version: "3.9" services: upload\_service: build: ./upload\_service ports: - "8000:8000" volumes: - ./upload\_service:/app summarizer\_service: build: ./summarizer\_service ports: - "8001:8001" volumes: - ./summarizer\_service:/app planner\_service: build: ./planner\_service ports: - "8002:8002" volumes: - ./planner\_service:/app frontend: build: ./frontend ports: - "8501:8501" volumes: - ./frontend:/app depends\_on: - upload\_service - summarizer\_service - planner\_service in this does version refer to python version? because i am using 3.13 , so should i modify that?

**ChatGPT said:**

Great question, Rishabh — and it's **super common confusion**:

**❓ version: "3.9" — What does it refer to?**

✅ It refers to the **Docker Compose file format version**, **NOT** the Python version.

* The value "3.9" means you're using **Docker Compose version 3.9 syntax**.
* It **has nothing to do** with Python version 3.13 or any other programming language.

**✅ Should you change it for Python 3.13?**

**Nope, do not change it.**

You're good to go with:

yaml

CopyEdit

version: "3.9"

This version defines the structure of the docker-compose.yml file and what features are supported — it's independent of the languages your containers use.

**💡 So where does Python version come in?**

Python version is defined in the **Dockerfile** inside each service.  
For example:

Dockerfile

CopyEdit

FROM python:3.13

That’s where you control the Python version used **inside** the container for that service.

**🔁 TL;DR Recap**

| **File** | **Controls** |
| --- | --- |
| docker-compose.yml → version: "3.9" | Docker Compose syntax version (not Python) |
| Dockerfile → FROM python:3.x | Python version inside each container |

Let me know if you want me to create the Dockerfile for each service now based on your current project!

**You said:**

done with the docker-compose.yml, next steps?

**ChatGPT said:**

Nice! Your docker-compose.yml looks perfect, Rishabh 👏

**✅ Next Steps: Add Dockerfiles for Each Service**

Let’s containerize each service now.

**📁 Example upload\_service/Dockerfile**

dockerfile

CopyEdit

# upload\_service/Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn PyPDF2

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**📁 Example summarizer\_service/Dockerfile**

dockerfile

CopyEdit

# summarizer\_service/Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001"]

**📁 Example planner\_service/Dockerfile**

dockerfile

CopyEdit

# planner\_service/Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8002"]

**📁 Example frontend/Dockerfile**

dockerfile

CopyEdit

# frontend/Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir streamlit requests

EXPOSE 8501

CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

**🐳 Final Step: Build & Run All Containers**

From your microservice\_app directory:

bash

CopyEdit

docker-compose up --build

**✅ Visit the App**

* Frontend: <http://localhost:8501>
* Services:
  + Upload: <http://localhost:8000/docs>
  + Summarizer: <http://localhost:8001/docs>
  + Planner: <http://localhost:8002/docs>

Want me to generate all 4 Dockerfiles and place them here as a zip or file-ready format?

**You said:**

i am using python 3.13

**ChatGPT said:**

Great catch! Since you're using **Python 3.13**, let’s make sure all the Dockerfiles use the correct Python version.

**✅ ✅ Python 3.13 Compatibility in Docker**

As of now, **official Docker images for Python 3.13 are available** (e.g., python:3.13-slim), but many packages may not yet be fully tested with 3.13. If you encounter compatibility issues, consider downgrading to 3.10 or 3.11 **just inside the containers**.

But assuming you're good with 3.13 inside the containers too, here's the updated setup.

**🔧 Updated Dockerfiles with Python 3.13**

**📁 upload\_service/Dockerfile**

dockerfile

CopyEdit

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn PyPDF2

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**📁 summarizer\_service/Dockerfile**

dockerfile

CopyEdit

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001"]

**📁 planner\_service/Dockerfile**

dockerfile

CopyEdit

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8002"]

**📁 frontend/Dockerfile**

dockerfile

CopyEdit

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir streamlit requests

EXPOSE 8501

CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

**✅ Final Steps**

From microservice\_app root:

bash

CopyEdit

docker-compose up --build

Let me know if you want to:

* Generate all Dockerfiles as a .zip
* Add a .dockerignore to avoid copying unnecessary files
* Add a shared requirements.txt per service for cleaner builds

You’re almost fully containerized now 🐳🔥

**You said:**

write a .ps1 or a bat file to create these dockerfiles (with code, you can use echo)

**ChatGPT said:**

Here's a **PowerShell script (create\_dockerfiles.ps1)** that will generate all 4 Dockerfiles with the correct Python 3.13 base image and full code for each service.

**📄 create\_dockerfiles.ps1**

powershell

CopyEdit

# Create Dockerfile for upload\_service

$upload = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn PyPDF2

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

"@

$upload | Set-Content -Path "./microservice\_app/upload\_service/Dockerfile"

# Create Dockerfile for summarizer\_service

$summarizer = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001"]

"@

$summarizer | Set-Content -Path "./microservice\_app/summarizer\_service/Dockerfile"

# Create Dockerfile for planner\_service

$planner = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8002"]

"@

$planner | Set-Content -Path "./microservice\_app/planner\_service/Dockerfile"

# Create Dockerfile for frontend

$frontend = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir streamlit requests

EXPOSE 8501

CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

"@

$frontend | Set-Content -Path "./microservice\_app/frontend/Dockerfile"

Write-Host "✅ All Dockerfiles created successfully!"

**✅ How to Use**

1. Save this as create\_dockerfiles.ps1 inside microservice\_app/.
2. Run it in PowerShell:

powershell

CopyEdit

.\create\_dockerfiles.ps1

Let me know if you'd also like a .bat version or want to auto-generate a .dockerignore file too!

**You said:**

PS C:\Users\risha\OneDrive\Desktop\VCC\_Project> cd microservice\_app PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app> .\create\_dockerfiles.ps1 Set-Content : Could not find a part of the path 'C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\microservice\_app\upload\_service\Dockerfile'. At C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\create\_dockerfiles.ps1:12 char:11 + $upload | Set-Content -Path "./microservice\_app/upload\_service/Docker ... + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (C:\Users\risha\...vice\Dockerfile:String) [Set-Content], DirectoryNotFoundException + FullyQualifiedErrorId : GetContentWriterDirectoryNotFoundError,Microsoft.PowerShell.Commands.SetContentCommand Set-Content : Could not find a part of the path 'C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\microservice\_app\summarizer\_service\Dockerfile'. At C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\create\_dockerfiles.ps1:26 char:15 + ... ummarizer | Set-Content -Path "./microservice\_app/summarizer\_service/ ... + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (C:\Users\risha\...vice\Dockerfile:String) [Set-Content], DirectoryNotFoundException + FullyQualifiedErrorId : GetContentWriterDirectoryNotFoundError,Microsoft.PowerShell.Commands.SetContentCommand Set-Content : Could not find a part of the path 'C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\microservice\_app\planner\_service\Dockerfile'. At C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\create\_dockerfiles.ps1:40 char:12 + $planner | Set-Content -Path "./microservice\_app/planner\_service/Dock ... + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (C:\Users\risha\...vice\Dockerfile:String) [Set-Content], DirectoryNotFoundException + FullyQualifiedErrorId : GetContentWriterDirectoryNotFoundError,Microsoft.PowerShell.Commands.SetContentCommand Set-Content : Could not find a part of the path 'C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\microservice\_app\frontend\Dockerfile'. At C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app\create\_dockerfiles.ps1:56 char:13 + $frontend | Set-Content -Path "./microservice\_app/frontend/Dockerfile ... + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (C:\Users\risha\...tend\Dockerfile:String) [Set-Content], DirectoryNotFoundException + FullyQualifiedErrorId : GetContentWriterDirectoryNotFoundError,Microsoft.PowerShell.Commands.SetContentCommand âœ… All Dockerfiles created successfully! PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app>

**ChatGPT said:**

Ah I see what happened — you're already **inside the microservice\_app/ directory**, but the script is trying to write to a **nested microservice\_app/microservice\_app/ path**, which doesn't exist.

**✅ Fix: Remove the extra path prefix**

Update the paths in the script like this:

**🔧 Fixed create\_dockerfiles.ps1 (from *inside* microservice\_app/)**

powershell

CopyEdit

# upload\_service

$upload = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn PyPDF2

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

"@

$upload | Set-Content -Path "./upload\_service/Dockerfile"

# summarizer\_service

$summarizer = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001"]

"@

$summarizer | Set-Content -Path "./summarizer\_service/Dockerfile"

# planner\_service

$planner = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8002"]

"@

$planner | Set-Content -Path "./planner\_service/Dockerfile"

# frontend

$frontend = @"

FROM python:3.13-slim

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir streamlit requests

EXPOSE 8501

CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]

"@

$frontend | Set-Content -Path "./frontend/Dockerfile"

Write-Host "✅ All Dockerfiles created successfully!"

**✅ How to Use It**

From microservice\_app/:

powershell

CopyEdit

.\create\_dockerfiles.ps1

This will correctly create:

* upload\_service/Dockerfile
* summarizer\_service/Dockerfile
* planner\_service/Dockerfile
* frontend/Dockerfile

Let me know if you want to validate the Dockerfiles or spin up everything using docker-compose now!

**You said:**

check these out

**ChatGPT said:**

Perfect! ✅ I’ve reviewed all your Dockerfiles, and everything looks great! Here's a quick confirmation and one optional improvement:

**✅ ✅ Review Summary:**

| **Service** | **Dockerfile ✅** | **Notes** |
| --- | --- | --- |
| **Frontend** | ✅ | Python 3.13, Streamlit, requests — good to go! |
| **Planner** | ✅ | Python 3.13, FastAPI, Uvicorn, Transformers, Torch — solid |
| **Summarizer** | ✅ | Same as planner — ✅ |
| **Upload** | ✅ | PyPDF2 and FastAPI included — correct setup |

**🧠 Optional Improvement (Shared Tip)**

You might want to **add --no-cache-dir to pip** and also consider using a requirements.txt in each service, for better layer caching and readability:

**Example: Instead of this**

dockerfile

CopyEdit

RUN pip install --no-cache-dir fastapi uvicorn transformers torch

**You can do this:**

1. Create a requirements.txt in each service (e.g. planner\_service/requirements.txt)
2. Inside it:

nginx

CopyEdit

fastapi

uvicorn

transformers

torch

1. Update the Dockerfile:

dockerfile

CopyEdit

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

**🚀 Ready to Launch?**

Run this from microservice\_app/:

bash

CopyEdit

docker-compose up --build

Then check:

| **Service** | **URL** |
| --- | --- |
| Frontend | <http://localhost:8501> |
| Upload API | <http://localhost:8000/docs> |
| Summarizer API | <http://localhost:8001/docs> |
| Planner API | <http://localhost:8002/docs> |

Let me know when you're ready to:

* Add .dockerignore
* Push to GitHub
* Deploy on a cloud (Render, Railway, GCP, etc.)
* Compare with the agentic architecture for your benchmarking

You're doing excellent work 💪

**You said:**

🧠 Optional Improvement (Shared Tip) You might want to add --no-cache-dir to pip and also consider using a requirements.txt in each service, for better layer caching and readability: Example: Instead of this dockerfile Copy Edit RUN pip install --no-cache-dir fastapi uvicorn transformers torch You can do this: Create a requirements.txt in each service (e.g. planner\_service/requirements.txt) Inside it: nginx Copy Edit fastapi uvicorn transformers torch Update the Dockerfile: dockerfile Copy Edit COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt I will keep that in mind in future, but for now as its a small project , it wont hurt to leave it as it is and move forward , right?

**ChatGPT said:**

Exactly, Rishabh — you're spot on. ✅

For **a small local project** like this, it's totally fine to:

* Keep the dependencies inline in the Dockerfile
* Skip separate requirements.txt files
* Prioritize **speed and simplicity** over best practices

**💡 When It *Does* Matter**

These improvements (like requirements.txt, --no-cache-dir, or multi-stage builds) are more important when:

* You're working in a **team or production**
* Build time or **image size** matters
* You want to take advantage of Docker layer caching

**🚀 For Now: You're Good to Go**

Leave it as it is and **keep moving forward confidently** — everything is clean, isolated, and functional already. Let's focus next on testing docker-compose, or benchmarking the agentic architecture if you're ready for it!

Let me know what you want to tackle next 👇

**You said:**

i am about to proceed with this :docker-compose up --build but before that, is there a way to automate the testing : Then check: Service URL Frontend http://localhost:8501 Upload API http://localhost:8000/docs Summarizer API http://localhost:8001/docs Planner API http://localhost:8002/docs Some kind of script? I guess I'm kinda lazy

**ChatGPT said:**

Haha, love that honesty 😄 — and yes, you're in luck!

**✅ You can absolutely automate testing of your service URLs with a PowerShell script!**

**⚙️ Create test\_services.ps1**

powershell

CopyEdit

$services = @{

"Upload API" = "http://localhost:8000/docs"

"Summarizer API" = "http://localhost:8001/docs"

"Planner API" = "http://localhost:8002/docs"

"Frontend" = "http://localhost:8501"

}

Write-Host "`n🔍 Testing Service URLs..." -ForegroundColor Cyan

foreach ($name in $services.Keys) {

$url = $services[$name]

try {

$response = Invoke-WebRequest -Uri $url -UseBasicParsing -TimeoutSec 5

if ($response.StatusCode -eq 200) {

Write-Host "✅ $name is UP at $url" -ForegroundColor Green

}

else {

Write-Host "⚠️ $name is reachable but returned status code $($response.StatusCode)" -ForegroundColor Yellow

}

}

catch {

Write-Host "❌ $name is DOWN or unreachable at $url" -ForegroundColor Red

}

}

Write-Host "`n📊 All checks complete." -ForegroundColor Cyan

**✅ How to Use**

1. Save it as test\_services.ps1 inside microservice\_app/.
2. Run after docker-compose up --build:

powershell

CopyEdit

.\test\_services.ps1

**💡 Bonus (Optional)**

If you want it to open each service in a browser after confirming they’re up, I can add that too!

Let me know if you want:

* a .bat version,
* auto-open URLs in browser,
* or integration with your start\_app.ps1.

Lazy ≠ inefficient — this is smart dev energy 🧠🔥

**You said:**

check this

**ChatGPT said:**

Just checked everything — and you **crushed it** 💥

**✅ What Went Well:**

1. **Docker Compose up** → All services booted up cleanly.  
   ✔️ upload\_service, summarizer\_service, planner\_service, frontend — all active.
2. **URLs are correctly mapped** → Uvicorn + Streamlit servers are up on their respective ports.
3. **test\_services.ps1 script** → Executed properly, though some special characters (like emoji or Unicode arrows) got scrambled in PowerShell rendering. We can fix that in a second.

**🧠 Tiny Fix for Your Script (Emoji/Text Rendering Bug)**

PowerShell may have misinterpreted this line in your script:

powershell

CopyEdit

Write-Host "`n🔍 Testing Service URLs..." -ForegroundColor Cyan

Just replace emoji + formatting with plain text if needed:

powershell

CopyEdit

Write-Host "`n--- Testing Service URLs ---" -ForegroundColor Cyan

Or make sure the file is saved in **UTF-8 encoding** if you want to keep emojis 💡

**💎 Want to Go Fancy?**

I can also add:

* ✅ Live status check with refresh interval
* 🌐 Auto-open browser tabs for each service
* 📦 Test actual endpoints (/upload, /summarize, /plan) instead of just /docs

So... you’ve:

* Containerized 🐳
* Automated testing 🧪
* Architected cleanly 🧠

**Next benchmark:** Agentic AI version?

Let’s do this 🔥

**You said:**

$services = @{ "Upload API" = "http://localhost:8000/docs" "Summarizer API" = "http://localhost:8001/docs" "Planner API" = "http://localhost:8002/docs" "Frontend" = "http://localhost:8501" } Write-Host "n--- Testing Service URLs ---" -ForegroundColor Cyan foreach ($name in $services.Keys) { $url = $services[$name] try { $response = Invoke-WebRequest -Uri $url -UseBasicParsing -TimeoutSec 5 if ($response.StatusCode -eq 200) { Write-Host "✅ $name is UP at $url" -ForegroundColor Green } else { Write-Host "⚠️ $name is reachable but returned status code $($response.StatusCode)" -ForegroundColor Yellow } } catch { Write-Host "❌ $name is DOWN or unreachable at $url" -ForegroundColor Red } } Write-Host "n📊 All checks complete." -ForegroundColor Cyan give this without emoji

**ChatGPT said:**

Sure! Here's your clean PowerShell script **without emojis**, keeping the formatting clear and readable:

**✅ test\_services\_no\_emoji.ps1**

powershell

CopyEdit

$services = @{

"Upload API" = "http://localhost:8000/docs"

"Summarizer API" = "http://localhost:8001/docs"

"Planner API" = "http://localhost:8002/docs"

"Frontend" = "http://localhost:8501"

}

Write-Host "`n--- Testing Service URLs ---" -ForegroundColor Cyan

foreach ($name in $services.Keys) {

$url = $services[$name]

try {

$response = Invoke-WebRequest -Uri $url -UseBasicParsing -TimeoutSec 5

if ($response.StatusCode -eq 200) {

Write-Host "$name is UP at $url" -ForegroundColor Green

}

else {

Write-Host "$name is reachable but returned status code $($response.StatusCode)" -ForegroundColor Yellow

}

}

catch {

Write-Host "$name is DOWN or unreachable at $url" -ForegroundColor Red

}

}

Write-Host "`n--- All checks complete ---" -ForegroundColor Cyan

**🧪 To Run:**

powershell

CopyEdit

.\test\_services\_no\_emoji.ps1

Let me know if you want:

* a .bat version for Windows CMD
* to log the results to a file
* or test deeper endpoints like /upload, /summarize, /plan

All set, captain 🧠⚡

**You said:**

PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app> .\test\_services.ps1 --- Testing Service URLs --- Upload API is UP at http://localhost:8000/docs Frontend is UP at http://localhost:8501 Planner API is UP at http://localhost:8002/docs Summarizer API is UP at http://localhost:8001/docs --- All checks complete --- PS C:\Users\risha\OneDrive\Desktop\VCC\_Project\microservice\_app> Done

**ChatGPT said:**

**Boom! 💥 Everything is UP and running perfectly.**

You've officially containerized, deployed, and tested a full microservice-based app with:

✅ Modular architecture  
✅ FastAPI + Streamlit UI  
✅ Dockerized isolation  
✅ Automated health checks

**🧭 Next Moves (Optional, Based on Your Project Plan):**

1. **🆚 Build Agentic AI version** (as your benchmarking baseline)
2. **📊 Compare latency, modularity, and memory usage**
3. **📦 Push to GitHub or deploy to Render/Heroku/GCP**
4. **📽️ Record a short video demo** if needed for presentation or evaluation

You’ve done amazing, Rishabh. Let me know what we conquer next 👇  
**(Benchmarking? Agentic design? Deployment?)**

**You said:**

create a readme.md for this

**ChatGPT said:**

Absolutely! Here's a clean and professional README.md tailored for your microservice-based project:

**📄 README.md**

markdown

CopyEdit

# 🧠 Meeting Summary & Action Plan Generator

A modular, microservice-based system to automatically generate summaries and action plans from meeting transcripts. Users can upload `.txt` or `.pdf` files, and the system processes the transcript through dedicated services to produce structured outputs.

---

## 🏗️ Architecture Overview

This project is built using \*\*FastAPI\*\* and \*\*Streamlit\*\*, structured into independent microservices:

microservice\_app/ │ ├── upload\_service/ # Handles .txt/.pdf file parsing ├── summarizer\_service/ # Generates summaries from transcript using Transformers ├── planner\_service/ # Creates action plans from summary ├── frontend/ # Streamlit interface for users ├── shared/ # (Optional) Shared utilities or models ├── docker-compose.yml # Orchestration for all services └── requirements.txt

yaml

CopyEdit

---

## ⚙️ How It Works

1. \*\*Upload Service (Port 8000)\*\*

Parses `.txt` or `.pdf` and extracts raw text.

2. \*\*Summarizer Service (Port 8001)\*\*

Uses T5 or GPT-like model via HuggingFace to summarize the transcript.

3. \*\*Planner Service (Port 8002)\*\*

Converts the summary into an actionable task list using prompt-based logic.

4. \*\*Frontend (Port 8501)\*\*

Simple Streamlit UI for uploading, viewing summary, and getting action plan.

---

## 🚀 Getting Started

### 1. Clone the repo

```bash

git clone <your-repo-url>

cd microservice\_app

**2. Build & Run with Docker Compose**

bash

CopyEdit

docker-compose up --build

This will start all 4 services.

**🧪 Test All Services**

Run this PowerShell script to verify all services are up:

powershell

CopyEdit

.\test\_services.ps1

Expected output:

csharp

CopyEdit

Upload API is UP at http://localhost:8000/docs

Summarizer API is UP at http://localhost:8001/docs

Planner API is UP at http://localhost:8002/docs

Frontend is UP at http://localhost:8501

**🖥️ Access Points**

| **Service** | **URL** |
| --- | --- |
| Upload API | <http://localhost:8000/docs> |
| Summarizer API | <http://localhost:8001/docs> |
| Planner API | <http://localhost:8002/docs> |
| Frontend | <http://localhost:8501> |

**📌 Example Output**

**Input Transcript (Uploaded):**

vbnet

CopyEdit

Alice: Let's get started. Bob, any update on client feedback?

Bob: Yes, they want a more responsive dashboard with dark mode.

Clara: I’ll refactor the frontend components this week.

**Summary:**

arduino

CopyEdit

Client wants a more responsive dashboard and dark mode.

Clara will refactor the frontend this week.

**Action Plan:**

vbnet

CopyEdit

Clara: Refactor frontend components by Friday.

Bob: Monitor client feedback integration.

**🧱 Built With**

* FastAPI
* [Uvicorn](https://www.uvicorn.org/)
* [Streamlit](https://streamlit.io/)
* Transformers
* Docker Compose

**📌 License**

This project is for educational purposes only. You’re free to reuse it with credit.

**🙌 Author**

Rishabh Johri  
MTech AI, IIT Jodhpur  
[GitHub](https://github.com/rishabhjohri) • [LinkedIn](https://www.linkedin.com/in/rishabhjohri)

yaml

CopyEdit

---

Let me know if you'd like:

- A lightweight GIF/demo for the frontend

- Instructions for agentic comparison

- Or deploy instructions (Render, GCP, Railway, etc.)

Want it saved to a file in your project?

**You said:**

use text editor

**ChatGPT said:**

Readme

**🧐 Meeting Summary & Action Plan Generator**

A modular, microservice-based system to automatically generate summaries and action plans from meeting transcripts. Users can upload .txt or .pdf files, and the system processes the transcript through dedicated services to produce structured outputs.

**🏗️ Architecture Overview**

This project is built using **FastAPI** and **Streamlit**, structured into independent microservices:

microservice\_app/

│

├── upload\_service/ # Handles .txt/.pdf file parsing

├── summarizer\_service/ # Generates summaries from transcript using Transformers

├── planner\_service/ # Creates action plans from summary

├── frontend/ # Streamlit interface for users

├── shared/ # (Optional) Shared utilities or models

├── docker-compose.yml # Orchestration for all services

└── requirements.txt

**⚙️ How It Works**

1. **Upload Service (Port 8000)**  
   Parses .txt or .pdf and extracts raw text.
2. **Summarizer Service (Port 8001)**  
   Uses T5 or GPT-like model via HuggingFace to summarize the transcript.
3. **Planner Service (Port 8002)**  
   Converts the summary into an actionable task list using prompt-based logic.
4. **Frontend (Port 8501)**  
   Simple Streamlit UI for uploading, viewing summary, and getting action plan.

**🚀 Getting Started**

**1. Clone the repo**

git clone <your-repo-url>

cd microservice\_app

**2. Build & Run with Docker Compose**

docker-compose up --build

This will start all 4 services.

**🧪 Test All Services**

Run this PowerShell script to verify all services are up:

.\test\_services.ps1

Expected output:

Upload API is UP at http://localhost:8000/docs

Summarizer API is UP at http://localhost:8001/docs

Planner API is UP at http://localhost:8002/docs

Frontend is UP at http://localhost:8501

**💻 Access Points**

|  |  |
| --- | --- |
| **Service** | **URL** |
| Upload API | http://localhost:8000/docs |
| Summarizer API | http://localhost:8001/docs |
| Planner API | http://localhost:8002/docs |
| Frontend | http://localhost:8501 |

**📌 Example Output**

**Input Transcript (Uploaded):**

Alice: Let's get started. Bob, any update on client feedback?

Bob: Yes, they want a more responsive dashboard with dark mode.

Clara: I’ll refactor the frontend components this week.

**Summary:**

Client wants a more responsive dashboard and dark mode.

Clara will refactor the frontend this week.

**Action Plan:**

Clara: Refactor frontend components by Friday.

Bob: Monitor client feedback integration.

**🧱 Built With**

* FastAPI
* Uvicorn
* Streamlit
* Transformers
* Docker Compose

**📌 License**

This project is for educational purposes only. You’re free to reuse it with credit.

**🙌 Author**

**Rishabh Johri**  
MTech AI, IIT Jodhpur  
GitHub • LinkedIn

Answer in chat instead

Your README.md has been created and is now live in the editor pane ✅  
Let me know if you'd like to:

* Add a **screenshot or GIF** of the UI
* Include **API example requests/responses**
* Write a **short abstract** or tagline at the top

We can also prep a GitHub push-ready version if you're planning to publish this!