Disease Profile ➜ Synthea Module Agentic Pipeline

This repository implements a **production-style agentic workflow** that:

1. Takes a **disease or condition name** (and optionally a folder of PDFs).
2. Uses **PubMed**, **PubMed Central (PMC)**, **ClinicalTrials.gov**, and local PDFs to build a **disease profile**.
3. Converts that disease profile into a **Synthea Generic Module Framework (GMF) JSON module**.
4. Lets you run the whole thing from the **CLI** and then plug the module into **Synthea** to generate **synthetic patient data**.

---

1. High-Level Architecture

The agent graph looks like this:

> *Replace with your screenshot file path*
> `docs/images/agent-graph.png`

```
DiseaseToSyntheaPipeline
├── disease_profile_agent
│   ├── extract_text_from_pdfs_in_folder
│   ├── pubmed_search
│   ├── pubmed_get_fulltext_from_pmc
│   ├── clinicaltrials_search
│   └── clinicaltrials_get_full_content
└── SyntheaModuleGeneratorAgent
    └── validate_json
1.1 Agents
disease_profile_agent

LLM agent with tools for:

PDF text extraction

PubMed search + full text from PMC

ClinicalTrials.gov search + full content

Output: a numbered disease profile stored in session.state["disease_profile"].

SyntheaModuleGeneratorAgent

LLM agent that:

Reads disease_profile from state.

Generates a safe, simplified Synthea GMF JSON module.

Validates JSON syntax using validate_json.
```

```
Output: the final JSON string stored in session.state["json"].

DiseaseToSyntheaFlowAgent (root)

Custom BaseAgent that orchestrates:

Run disease_profile_agent, capture final text.

Save it to session.state["disease_profile"].

Run SyntheaModuleGeneratorAgent and stream out the final Synthea JSON.

2. Project Structure
text
Copy code
your-project/
├── agents/
│   ├── disease_profile.py        # Disease profile LLM agent + tools
│   ├── synthea_module.py         # Synthea module generator agent
│   └── pipeline_agent.py         # DiseaseToSyntheaFlowAgent (root)
├── config/
│   └── settings.py               # Loads .env / external API config
├── tools/
│   ├── clinicaltrials.py         # ClinicalTrials.gov v2 wrapper
│   ├── json_validator.py         # JSON syntax validator tool
│   ├── pdf_extractor.py          # Folder-based PDF text extractor
│   └── pubmed_api.py             # PubMed + PMC tools
├── .env                          # API keys (NOT committed)
├── main.py                       # CLI runner
└── requirements.txt              # Python dependencies
3. Environment Setup
3.1 Create and Activate Virtualenv
bash
Copy code
python -m venv .venv

# Linux / macOS
source .venv/bin/activate

# Windows PowerShell
.\.venv\Scripts\activate
3.2 Install Dependencies
bash
Copy code
pip install -r requirements.txt
requirements.txt:

text
Copy code
google-adk
requests
python-dotenv
pypdf
3.3 Configure Environment Variables (.env)
```

```
Create a .env file at the project root:

dotenv
Copy code
NCBI_API_KEY=YOUR_NCBI_API_KEY
GOOGLE_API_KEY=YOUR_GOOGLE_GENAI_API_KEY
NCBI_API_KEY — optional but recommended (better PubMed limits).

GOOGLE_API_KEY — required for Google GenAI (used by Google ADK).

⚠ Do not commit .env to version control. Treat it as a secret file.

config/settings.py will automatically load .env and expose NCBI_API_KEY.

4. Agents in Detail
4.1 disease_profile_agent (agents/disease_profile.py)
Type: google.adk.agents.llm_agent.Agent

Tools Wired In
extract_text_from_pdfs_in_folder(folder_path: str)

pubmed_search(term: str, max_results: int)

pubmed_get_fulltext_from_pmc(pmid: str)

clinicaltrials_search(condition: str, max_results: int)

clinicaltrials_get_full_content(nct_id: str)

Each of these is wrapped as a FunctionTool and passed into the agent:

python
Copy code
disease_profile_agent = Agent(
    name="disease_profile_agent",
    model="gemini-2.5-flash",
    description="Agent that can use PDFs, PubMed and ClinicalTrials.gov to gather
biomedical evidence.",
    instruction=SMART_RESEARCH_INSTRUCTION,
    tools=[
        pdf_extractor_tool,
        pubmed_search_tool,
        pubmed_fulltext_tool,
        ct_search_tool,
        ct_full_content_tool,
    ],
    output_key="disease_profile",
)
Behavior (SMART_RESEARCH_INSTRUCTION)
The instruction enforces:

Tool-first behavior:
The agent must call at least one tool and can’t rely on its own training data.
```

```
Strict no-hallucination:
It never invents numbers (prevalence, incidence, odds ratios, etc.).
If something isn’t in the sources, it explicitly states that it’s not available.

Output format:

Only a numbered list of disease facts (1., 2., 3., …).

Designed to feed directly into the Synthea module generator.

The profile typically covers:

Prevalence & incidence

Demographics / population

Risk factors

Etiology & pathophysiology

Symptoms & clinical presentation

Diagnosis

Natural history

Treatment strategies

Adverse events & safety

Long-term outcomes

The final text is:

streamed as the agent’s final response, and

stored in session.state["disease_profile"].

4.2 SyntheaModuleGeneratorAgent (agents/synthea_module.py)
Type: google.adk.agents.LlmAgent

JSON Validator Tool
python
Copy code
from tools.json_validator import validate_json
json_validator_tool = FunctionTool(func=validate_json)
validate_json(json_string: str):

Returns {"is_valid": True/False, "reason": "..."}

Uses json.loads under the hood.

Instruction Provider
python
```

```
Copy code
async def synthea_instruction_provider(ctx: ReadonlyContext) -> str:
    disease_profile = ctx.session.state.get("disease_profile", "")
    return (
        SYNTHEA_GENERATOR_PROMPT
        + "\n\n====================================\n"
        + "DISEASE PROFILE (FROM PREVIOUS AGENT)\n"
        + "====================================\n"
        + f"{disease_profile}\n"
        + "\nRemember: treat the above as the DISEASE PROFILE numbered facts "
          "and follow the process to output a single valid GMF JSON module."
    )
This function:

Takes the disease profile produced by disease_profile_agent.

Appends it under a clear header to SYNTHEA_GENERATOR_PROMPT.

The prompt describes an extremely constrained, SAFE MODE Synthea GMF generator.

SAFE MODE Highlights
Allowed state types only:

Initial, Terminal, Guard, Encounter, EncounterEnd,
ConditionOnset, MedicationOrder, Procedure, Observation, Death.

No numeric quantity fields:

No "exact", "range", "unit", "distribution", etc.

Observations must use only value_code (qualitative).

Transitions:

Only direct_transition allowed.

No distributed_transition, conditional_transition, etc.

Terminology rules:

The module may only use real codes (SNOMED, RxNorm, LOINC, etc.) if they are
explicitly present in the disease profile.

Otherwise, it must use obvious placeholders, for example:

SNOMED-CT: code: "999999", display: "Placeholder SNOMED Concept".

RxNorm: code: "999999", display: "Placeholder Medication".

LOINC: code: "99999-9", display: "Placeholder Observation".

Validation loop:

The agent must call validate_json internally until JSON is valid.
```

```
Only then may it return the JSON string.

Agent Definition
python
Copy code
synthea_module_generator_agent = LlmAgent(
    model='gemini-2.5-flash',
    name='SyntheaModuleGeneratorAgent',
    description=(
        "Generates a valid, minified Synthea JSON module for a specific disease, "
        "based solely on a provided disease profile from the previous agent. "
        "It must not hallucinate medical info and must validate the JSON with
json_validator_tool. "
        "Return ONLY raw JSON object. "
        "Do NOT wrap the JSON inside quotes. "
        "Do NOT escape quotes."
    ),
    instruction=synthea_instruction_provider,
    tools=[json_validator_tool],
    output_key="json",
)
4.3 DiseaseToSyntheaFlowAgent (agents/pipeline_agent.py)
Type: google.adk.agents.BaseAgent subclass

Purpose
A custom orchestrator that:

Runs disease_profile_agent to generate the disease profile.

Stores its final output text into ctx.session.state["disease_profile"].

Runs SyntheaModuleGeneratorAgent to create the Synthea GMF JSON.

Initialization
python
Copy code
class DiseaseToSyntheaFlowAgent(BaseAgent):
    disease_profile_agent: LlmAgent
    synthea_module_agent: LlmAgent

    model_config = {"arbitrary_types_allowed": True}

    def __init__(self, name, disease_profile_agent, synthea_module_agent):
        sub_agents = [disease_profile_agent, synthea_module_agent]
        super().__init__(
            name=name,
            disease_profile_agent=disease_profile_agent,
            synthea_module_agent=synthea_module_agent,
            sub_agents=sub_agents,
        )
Core Logic
python
Copy code
```

```
@override
async def _run_async_impl(
    self, ctx: InvocationContext
) -> AsyncGenerator[Event, None]:
    # 1) Run disease_profile_agent
    disease_profile_text = None
    async for event in self.disease_profile_agent.run_async(ctx):
        yield event  # stream up
        if event.is_final_response() and event.content and event.content.parts:
            parts_text = [getattr(p, "text", "") or "" for p in
event.content.parts]
            disease_profile_text = "\n".join(parts_text).strip()

    # fallback to state if needed
    if not disease_profile_text:
        disease_profile_text = (
            ctx.session.state.get("disease_profile")
            or ctx.session.state.get("disease_profile_text")
        )
    if not disease_profile_text:
        # abort if profile failed
        return

    ctx.session.state["disease_profile"] = disease_profile_text

    # 2) Run SyntheaModuleGeneratorAgent
    async for event in self.synthea_module_agent.run_async(ctx):
        yield event
Root Agent
python
Copy code
root_agent = DiseaseToSyntheaFlowAgent(
    name="DiseaseToSyntheaPipeline",
    disease_profile_agent=disease_profile_agent,
    synthea_module_agent=synthea_module_generator_agent,
)
This root_agent is the one used by the CLI runner.

5. Tools
5.1 tools/pubmed_api.py
pubmed_search(term, max_results)

Uses NCBI ESearch + ESummary.

Returns a list of article metadata (PMID, title, journal, date, URL).

pubmed_get_fulltext_from_pmc(pmid)

Maps PMID → PMCID via ELink.

Fetches PMC XML (if available).

Extracts article body text.
```

```
Returns metadata + full text + URLs.

5.2 tools/clinicaltrials.py
clinicaltrials_search(condition, max_results)

Calls ClinicalTrials.gov v2 /studies endpoint.

Returns simplified (NCT ID, titles, conditions, status, phase, dates, URL).

clinicaltrials_get_full_content(nct_id)

Fetches one study by NCT ID.

Returns detailed protocol info: eligibility, arms, interventions, outcomes,
locations, plus raw JSON.

5.3 tools/pdf_extractor.py
extract_text_from_pdfs_in_folder(folder_path)

Iterates over .pdf in the given directory.

Uses pypdf.PdfReader to extract text.

Returns a dict {filename: text} or an error.

5.4 tools/json_validator.py
validate_json(json_string)

Basic json.loads wrapper returning {is_valid, reason}.

6. Running the Pipeline via CLI
The CLI entrypoint is main.py.

6.1 Command
From the project root (with virtualenv activated):

bash
Copy code
python main.py
You should see:

text
Copy code
Disease Profile + Synthea pipeline (CLI mode). Type 'exit' to quit.

you>
6.2 Example Usage
At the you> prompt, you can type instructions like:

text
Copy code
you> Use PubMed and ClinicalTrials.gov to build a disease profile for "Malaria",
then generate a Synthea module.
The pipeline will:
```

```
Call disease_profile_agent → gather evidence via tools → output a numbered disease
profile.

Store that text in session.state["disease_profile"].

Call SyntheaModuleGeneratorAgent → generate a Synthea GMF JSON module.

Validate JSON and print the final JSON as:

text
Copy code
agent>
{"name":"Malaria_Module","gmf_version":2,"remarks":[...],"states":{...}}
You can copy that JSON directly to use with Synthea (next section).

To exit the CLI:

text
Copy code
you> exit
7. Generating Synthetic Patients with Synthea
This section walks through the exact commands (as used in Colab) to:

Clone and build Synthea.

Add the generated JSON module.

Run Synthea to generate patient data.

These commands assume you are in Google Colab (Linux-like environment), but the
same steps work locally with small modifications.

7.1 Clone Synthea
bash
Copy code
!git clone https://github.com/synthetichealth/synthea.git
%cd synthea
7.2 Build Synthea
bash
Copy code
!./gradlew build -x test
This compiles Synthea and skips tests (faster).

7.3 Add Your Custom Module
Return to the Synthea project directory (if needed):

bash
Copy code
%cd /content/synthea
Paste your generated JSON module into a Python variable.

Example (shortened; you will paste the exact module from the CLI):
```

```
python
Copy code
malaria_module_json = r'''
{
  "name": "Malaria_Module",
  "gmf_version": 2,
  "remarks": ["Module generated from disease profile for Malaria."],
  "states": {
    "Initial": {
      "type": "Initial",
      "direct_transition": "Check_Eligibility"
    },
    "Check_Eligibility": {
      "type": "Guard",
      "allow": {
        "condition_type": "And",
        "conditions": [
          {
            "condition_type": "Age",
            "operator": "<",
            "quantity": 60,
            "unit": "years"
          },
          {
            "condition_type": "Gender",
            "gender": "M"
          }
        ]
      },
      "direct_transition": "Diagnosis_Encounter"
    },
    "...": {}
  }
}
'''
Write it into the Synthea modules directory:

python
Copy code
with open('src/main/resources/modules/my_custom_module.json', 'w') as f:
    f.write(malaria_module_json)

print("wrote my_custom_module.json")
Synthea will now see my_custom_module.json as a module.

7.4 Run Synthea
There are two primary ways to run Synthea in this workflow.

Option A — ./gradlew run (mirrors your logs)
bash
Copy code
!./gradlew run
Synthea will:
```

```
Scan all modules in src/main/resources/modules/ (including my_custom_module.json).

Generate synthetic patient data according to its default configuration.

Output is written by default under:

text
Copy code
output/
  ├── csv/
  ├── fhir/
  ├── cda/
  └── ...
You will see logs like:

text
Copy code
> Task :run
Scanned 90 modules and 157 submodules.
Loading module modules/my_custom_module.json
...
BUILD SUCCESSFUL
Option B — run_synthea script (if available)
Some Synthea clones include a run_synthea helper script in the repo root.
If present, you can control the number of patients and the region:

bash
Copy code
!./run_synthea Massachusetts -p 100
Massachusetts – state / region seed.

-p 100 – number of patients.

If run_synthea is not available or fails, stick with !./gradlew run.

7.5 Inspecting Generated Data
List CSV outputs:

bash
Copy code
!ls output/csv
Preview a CSV file:

bash
Copy code
!head -n 20 output/csv/patients.csv
From Colab, you can then:

Download files via the file browser, or

Copy them to Google Drive.

8. Production Notes
Secrets & Config
```

```
Never commit real API keys.

In real deployments, use environment variables or a secret manager.

Logging

logging is used throughout for visibility.

Adjust levels (INFO, DEBUG, WARNING) as needed.

Extensibility

You can add:

More tools (e.g., other data sources).

Extra agents (e.g., a module QA agent, code-mapping agent).

Additional validation steps before writing to Synthea.

9. Quick Start Summary
Setup environment

bash
Copy code
python -m venv .venv
source .venv/bin/activate    # or .\.venv\Scripts\activate on Windows
pip install -r requirements.txt
Create .env

dotenv
Copy code
NCBI_API_KEY=YOUR_NCBI_API_KEY
GOOGLE_API_KEY=YOUR_GOOGLE_GENAI_API_KEY
Run the pipeline

bash
Copy code
python main.py
At the prompt:

text
Copy code
you> Build a disease profile from PubMed and ClinicalTrials.gov for "Malaria" and
generate a Synthea module.
Copy the final agent> JSON output.

In Colab / Linux, generate patients

bash
Copy code
!git clone https://github.com/syntheticahealth/synthea.git
%cd synthea
```

```
!./gradlew build -x test

%cd /content/synthea
# paste module JSON into python string and write to:
# src/main/resources/modules/my_custom_module.json

!./gradlew run     # or ./run_synthea Massachusetts -p 100
Grab synthetic patient data from output/.

You now have an end-to-end, agentic, production-style workflow:
disease evidence → disease profile → Synthea module JSON → synthetic patient data.

makefile
Copy code
::contentReference[oaicite:0]{index=0}
```