**Refill Agent API — Contract-Driven Design Documentation**

**📌 Project Overview**

The **Refill Agent API** is a Flask-based application that simulates the behavior of a healthcare agent handling medication refill requests. The logic is governed by a **Contract-Driven Design** using Relari's Agent Contract paradigm, enforcing **preconditions**, **path conditions**, and **postconditions** to ensure consistent, safe, and logical behavior.

**🧱 Directory Structure**

graphql

CopyEdit

agent-contracts-refill/

│

├── app/

│ ├── \_\_init\_\_.py

│ ├── contracts.py # Defines the RefillContract (with all contract checks)

│ ├── logic.py # Core logic for handling refill requests

│ ├── mock\_db.py # Mock database simulating patient data

│ └── routes.py # API route that triggers the contract-driven logic

│

├── run.py # Flask app launcher

└── requirements.txt # (optional) For Python package dependencies

**🔄 API Endpoint**

**POST /process\_refill**

* **Description:** Processes a refill request after validating via contract-based conditions.
* **Request Body (JSON):**

json

CopyEdit

{

"patient\_id": "12345",

"prescription\_number": "RX123",

"medication\_name": "XYZ"

}

* **Success Response (200):**

json

CopyEdit

{

"message": "Refill approved for John Doe. Pickup in 2 hours. Confirmation: REF45123"

}

* **Error Responses:**
  + Missing fields:

json

CopyEdit

{

"message": "Preconditions not met.",

"errors": ["Missing or empty field: patient\_id"]

}

* + Unsupported medication:

json

CopyEdit

{

"message": "Path conditions not met.",

"errors": ["Medication not supported"]

}

* + Postcondition failure:

json

CopyEdit

{

"message": "Postconditions failed after refill.",

"errors": ["Postcondition failed: Refill not approved"]

}

**✅ Contract-Driven Design**

**✔️ 1. Preconditions**

Ensures that the input meets all basic schema requirements.

Implemented using Pydantic for validation:

python

CopyEdit

class RefillInput(BaseModel):

patient\_id: str

prescription\_number: str

medication\_name: str

Fails if any required field is missing or empty.

**✔️ 2. Path Conditions**

Controls the valid logical paths the agent can take.  
Example: Only specific medications are eligible for refills.

python

CopyEdit

if self.data.get("medication\_name") not in ["XYZ", "ABC", "123", "Paracetamol"]:

self.errors.append("Medication not supported")

**✔️ 3. Postconditions**

Checks if the action performed meets expected outcomes.  
Example: Message must contain "Refill approved".

python

CopyEdit

if "Refill approved" not in response\_message:

self.errors.append("Postcondition failed: Refill not approved")

**💡 Mock Database**

mock\_db.py simulates patient and prescription records:

python

CopyEdit

MOCK\_DB = {

"12345": {

"name": "John Doe",

"prescriptions": {

"RX123": {

"medication": "XYZ",

"last\_filled": "2024-03-15"

}

}

}

}

**🔧 Business Logic (logic.py)**

* Extracts values from input
* Checks contracts in this order:
  1. check\_preconditions
  2. check\_path\_conditions
  3. Executes refill and simulates update
  4. check\_postconditions
* Returns appropriate message or error response

**🧪 Testing the API**

You can test the /process\_refill endpoint using curl:

bash

CopyEdit

curl -X POST http://127.0.0.1:5000/process\_refill ^

-H "Content-Type: application/json" ^

-d "{\"patient\_id\": \"12345\", \"prescription\_number\": \"RX123\", \"medication\_name\": \"XYZ\"}"

**🚀 How to Run**

1. Navigate to project root:

bash

CopyEdit

cd agent-contracts-refill

1. Start Flask server:

bash

CopyEdit

python run.py

1. Server will run at:  
   [**http://127.0.0.1:5000**](http://127.0.0.1:5000)

**🏁 Final Notes**

* **Scalability:** Easily extendable to handle other agent actions.
* **Maintainability:** Contracts are modular and separated from logic.
* **Verifiability:** Each stage of execution is explicitly validated.
* **Inspired by:** [Relari Agent Contracts](https://agent-contracts.relari.ai/contracts/contracts)