#DYNMAIC ROUTING ENGINE

The Dynamic Routing Engine is a C++-based system that computes the most optimal top-K transaction paths across global payment corridors. It leverages an A\*-based search algorithm, integrates real-time ML retry scoring via HTTP from a Python Flask server, and enforces regional compliance filters.

This engine is core to the intelligent path selection in the Transactly platform, simulating how a real-world fintech might route cross-border transactions optimally, securely, and intelligently.

->router.cpp

Purpose: Implements the A\* search algorithm to find top-K most optimal paths between source and destination nodes in a payment graph

Functions:-

load\_graph(): Initializes a simulated corridor graph with latency and cost attributes.

get\_top\_k\_routes(source, target, K): Uses A\* logic to explore the best K paths using weighted latency and cost

get\_cost(corridor): Combines latency and cost using a linear weight formula.

get\_route(source, goal): Filters paths using compliance and dynamic retry scoring via ML.

->retry\_score.cpp

Purpose: Makes HTTP POST requests from C++ to a local Python Flask server to get ML-generated retry scores.

->compliance.hpp

Purpose: Contains hardcoded or configurable compliance rules that filter out disallowed corridors (e.g., embargoed countries).

JSON payload:

{

"corridor\_id": "A\_B",

"status": "active",

"corridor\_type": "primary",

"success\_rate\_7d": random(0,1),

"latency\_ms": 180,

"cost\_score": 0.10,

"past\_retry\_success\_rate": random(0,1)

}

GRAPH USED HERE:-

IN

├── SG ── US

│ └── IR ✖️ (non-compliant)

├── AE ── US

│ └── EU ── US

├── RU ── KP ── US

Testing & Validation

Run Flask ML Server:

python app.py

Build & Run C++ Server:

g++ -std=c++17 main.cpp -lcurl

./Microservice.out

Query the Microservice:

start node=start

end node=end

curl "http://localhost:8080/getRoute?start='start'&goal='end'"

example:

curl "http://localhost:8080/getRoute?start=IN&goal=US"