High-Throughput Request Gateway with Circuit Breaker

A lightweight, efficient API Gateway built using FastAPI that routes requests to backend microservices with round-robin load balancing and a built-in circuit breaker mechanism for fault tolerance.

Features:

- Round-Robin Load Balancing across multiple service instances
- Per-URL Circuit Breaker to isolate failed services
- Configurable Service Mappings via YAML (`config.yaml`)
- Hot Configuration Reloading without restarting the gateway
- Cooldown & Retry Logic with half-open state recovery
- Built with FastAPI and httpx for performance and scalability

Architecture Overview:

- The gateway sits in front of microservice replicas and handles:
- Request routing based on service name
- Load balancing using round-robin strategy
- Failure detection and circuit breaking
- Transparent recovery from failed services

```
Client ──> Gateway (/v1/proxy/{service}) ──> Healthy Backend
└──X──> Skipped (Breaker Open)
```

Project Structure:

File/Directory	Description
main.py	FastAPI app with routing and proxy logic
circuit_breaker.py	Circuit breaker implementation
config.yaml	Service-to-URL mapping configuration
requirements.txt	Python dependencies
dockerfile	Docker file commands
README.md	Project documentation

Installation and Usage:

1. **Install Dependencies:**
   pip install -r requirements.txt

2. **Start Dummy Backend Services (Optional for Testing):**
   python -m http.server 9001
   python -m http.server 9002

3. **Run the Gateway:**
   uvicorn main:app --host 127.0.0.1 --port 8000

4. **Send a Proxy Request:**
   curl http://127.0.0.1:8000/v1/proxy/user-service

5. **Reload Config Without Restart:**
   curl -X POST http://127.0.0.1:8000/v1/reload-config

Circuit Breaker Logic:

Each backend URL maintains its own circuit state:

- Closed: Normal traffic flow.
- Open: Skips traffic after 5 consecutive failures.
- Cooldown: Skips traffic for 60 seconds.
- Half-Open: Sends 1 test request.
  - ✅ If success → circuit closes.
  - ❌ If failure → circuit reopens.

This logic prevents request pileups to failed services and enables automatic recovery.

Configuration File: config.yaml:

```
user-service:
  - http://localhost:9001
  - http://localhost:9002
```

Testing Strategy:

- Simulate failures by manually stopping one backend
- Validate load balancing with multiple replicas
- Observe circuit breaker triggering and recovery using logs or request responses