Take home assignment

API Gateway Service Assignment

Overview

Create a simple API Gateway/Proxy service that demonstrates your ability to build production-ready microservices in Go. The service should handle API key validation, rate limiting with cross-instance synchronization, and request proxying.

Core Requirements

1. **Proxy Service**
   - Forward HTTP requests to configurable backend services
   - Support path-based routing
   - Handle request/response headers properly
   - Show good usage of middlewares

2. **API Protection**
   - The proxy must protect backend APIs by validating access tokens on each request
   - Access tokens should be provided as HTTP headers (e.g., Authorization: Bearer <token>)
   - Token data (including rate limits, expiry, and allowed routes) should be stored in Redis
   - Support rate limiting per token
   - Handle token expiration
   - Be clear in your implementation how tokens are created/provided (e.g., you may use static tokens, generate them with a script, or use certificates—describe your approach in the README)
3. **Rate Limiting**
   - Requests must be rate limited per token
   - Distributed rate limiting (synchronization across multiple instances) is optional; if implemented, highlight your approach

4. **Technical Requirements**
   - Written in Go 1.21+
   - Redis for token storage and rate limit synchronization
   - Environment-based configuration
   - Concurrent request handling
   - Proper error handling and logging

Required Endpoints

```
POST /api/v1/*      // Proxy endpoint
```

Token Data Structure

```
{
    "api_key": "xxx-xxx-xxx",
    "rate_limit": 100,
    "expires_at": "2024-12-31T23:59:59Z",
    "allowed_routes": ["/api/v1/users/*", "/api/v1/products/*"]
}
```

Minimum Deliverables

- Working proxy service with token validation and rate limiting
- Dockerfile
- Unit tests
- Basic documentation
- (Optional) Implementation of distributed rate limiting

Bonus Features (Optional)

1. Health check and readiness endpoints
2. Metrics endpoint
3. Helm charts
4. OpenAPI documentation
5. Integration tests
6. Circuit breaker
7. Prometheus metrics integration

Development Approach

We encourage you to use all resources available to professional developers, including documentation, libraries, and modern development tools. Leveraging AI assistants for code generation, problem-solving, or architectural guidance is perfectly acceptable as we're interested in your problem-solving approach and final solution quality rather than whether every line was manually typed.

Time Limit

Don't worry about rushing through everything! This assignment is really about showing us how you think and structure your code, not checking off every feature on the list. While you could tackle the core stuff in 2-3 hours, take whatever time feels right to you. We'd much rather see a clean, well-thought-out solution than a rushed attempt to do it all.

Submission

1. You will receive a very crude repo which you need to work on
2. Include documentation in README.md or via godocs
3. Ensure code is well-commented
4. Include build and run instructions

What We're Looking For

1. Clean, well-structured code
2. Proper handling of concurrent operations
3. Thoughtful error handling
4. Critical area testing coverage
5. Clear documentation
6. Effective distributed system design (if attempted)

Note on Submission

When submitting your solution, we appreciate any highlights of potential inconsistencies you've identified or your personal take on the assignment requirements. This reflects the real-world scenario where requirements often benefit from critical analysis and refinement.