Riffy - Architecture and Design Document

# Project Overview

Riffy is a lightweight, high-performance reverse proxy written in Rust. It supports forwarding HTTP requests to multiple upstream servers with basic round-robin load balancing. The proxy can be dynamically configured using environment variables and is designed to be containerized and deployed in Kubernetes environments.

# Functions Overview

## 1. handle\_proxy

Purpose:  
This function is responsible for receiving an incoming HTTP request from the client, forwarding it to one of the configured upstream servers, and returning the response from the upstream server to the original client. It implements simple round-robin load balancing to distribute the requests across multiple upstream servers.

Parameters:  
- req: The incoming Request<Body> from the client, which needs to be forwarded to an upstream server.  
- upstream\_servers: An Arc<Vec<String>> containing the list of upstream servers to which requests can be forwarded. It’s wrapped in an Arc to ensure thread safety.  
- counter: An Arc<AtomicUsize> that maintains a global counter used to perform round-robin load balancing across the upstream servers. The AtomicUsize ensures the counter can be updated safely across threads.

Functionality:  
- Round-Robin Load Balancing: The function selects the next upstream server in the list using the counter (counter.fetch\_add(1, Ordering::SeqCst)), and uses modulo arithmetic to ensure the index stays within bounds (% upstream\_servers.len()).  
- URI Construction: The function constructs the new URI by combining the upstream server's base URL with the path and query from the client's request.  
- Request Forwarding: A new request is created using hyper::Request::builder() to replicate the original client's request but directed to the selected upstream server. The hyper::Client forwards this request to the upstream server and returns the response.  
- Error Handling: Errors during request construction or forwarding are captured using Box<dyn std::error::Error + Send + Sync>, allowing flexibility in error types.

## 2. main

Purpose:  
The main function initializes the proxy server, loads environment variables, configures upstream servers, and starts listening for incoming client requests. It also sets up round-robin load balancing across multiple upstream servers.

Functionality:  
- Environment Variable Loading: It uses the dotenv crate to load environment variables from a .env file. The UPSTREAM\_SERVERS variable provides a comma-separated list of upstream servers, and LISTEN\_PORT specifies the port on which the proxy listens.  
- Round-Robin Counter Initialization: An Arc<AtomicUsize> is initialized to maintain a counter for round-robin load balancing, ensuring that each request is distributed across the upstream servers evenly.  
- Upstream Server List: The upstream server list is stored as a Vec<String> inside an Arc to ensure safe access across multiple threads.  
- Service Function Setup: The make\_service\_fn and service\_fn are used to create a new service instance for each connection, forwarding requests to handle\_proxy. The shared upstream server list and the counter are passed into each instance using Arc::clone.  
- Server Setup: The hyper::Server binds to the port specified by the LISTEN\_PORT environment variable and starts listening for incoming requests.

- SSL/TLS Configuration (Optional): Although the current implementation does not handle SSL/TLS termination, the code provides placeholders for future SSL support by loading certificate paths from SSL\_CERT\_PATH and SSL\_KEY\_PATH.