Self-Documentating API-Gateway

Solozobov Evgeny, Verhovod Nikita, Polienko Artem, Kochnev Victor

Project statement

A typical ingress implementation in k8s forwards incoming traffic to a particular application. An improved API Gateway would integrate Single Sign-On, automatic OpenAPI schema generation, request validation and response caching. The goal of the project is to develop such a Kubernetes operator to extend existing ingress implementations to API Gateway. The project should provide means to define API schema using Custom Resource Definitions in K8s directly and generate them from the source code following a set of conventions.

Team: WR
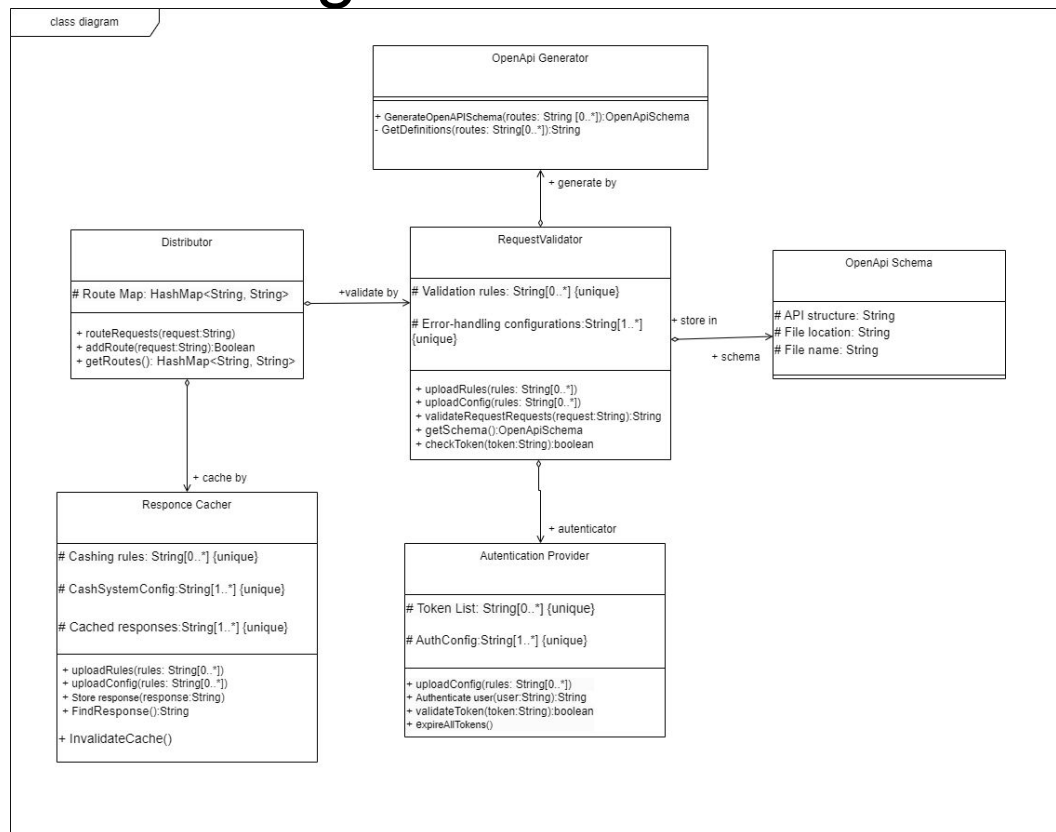
Project repo: github.com/laym0n/wr

This report: tiny.cc/project-team-wr

Use case diagram
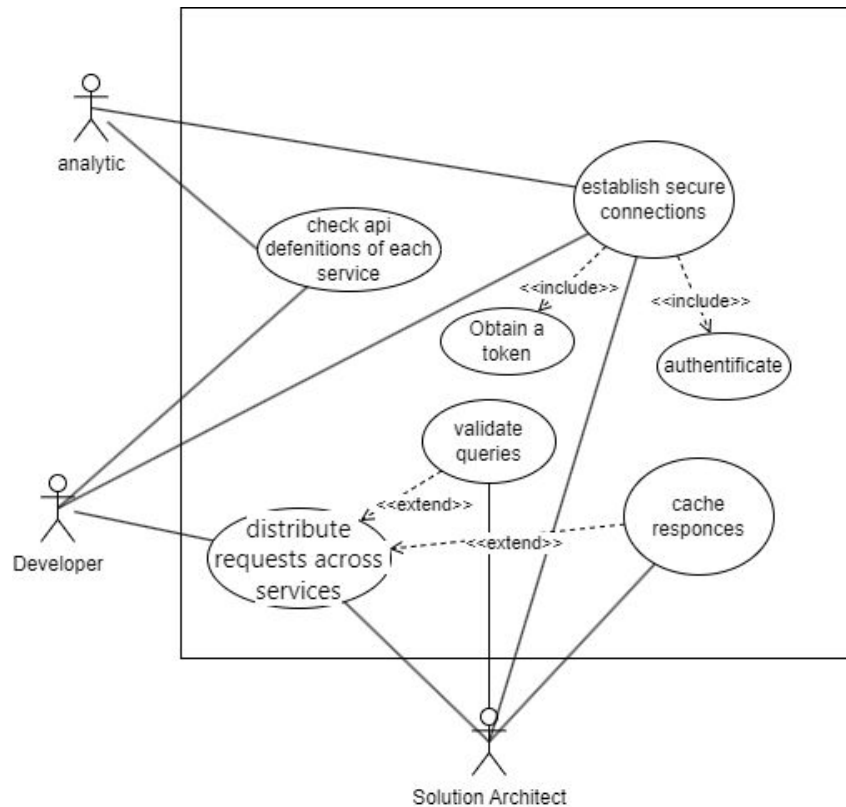
Class candidates (problem domain)

Candidate	Criteria	Stored information	Operations
Distributer	SIOUT	Route Map	- Route requests - Add routes - Get routes
OpenAPI Schema	SAUT	API structure File location File name	
Request Validator	SIAOUT	Validation rules, error-handling configurations, OpenAPI Schema	- Validate requests - Report validation errors - Get schema - Upload rules - Upload config - Check token
Response Cacher	SIAOU	Cached system configuration, Cached responses, caching rules	- Store response - Retrieve cached responses - Invalidate cache entries - Upload config - Upload rules
Authentication provider	SIAOUT	Token List, authentication configurations	- Authenticate user - Validate token - Expire All Tokens - Upload config
OpenApi Generator	OUT		- Generate OpenAPI schema - Take definitions from other services

Domain model diagram

Domain model diagram

DDD

<<service>>OpenApi Generator

+ GenerateOpenAPISchema(routes: String [0..*]):OpenApiSchema
- GetDefinitions(routes: String[0..*]):String

+ generate by

<<root>>Distributor

Route Map: HashMap<String, String>

+ routeRequests(request:String)
+ addRoute(request:String):Boolean
+ getRoutes(): HashMap<String, String>

+validate by

<<root>>RequestValidator

Validation rules: String[0..*] {unique}

Error-handling configurations:String[1..*]
{unique}

+ uploadRules(rules: String[0..*])
+ uploadConfig(rules: String[0..*])
+ validateRequestRequests(request:String):String
+ getSchema():OpenApiSchema
+ checkToken(token:String)::boolean

+ store in

+ schema

<<value>>OpenApi Schema

API structure: String
File location: String
File name: String

+ cache by

<<entity>>Responce Cacher

Cashing rules: String[0..*] {unique}

CashSystemConfig:String[1..*] {unique}

Cached responses:String[1..*] {unique}

+ uploadRules(rules: String[0..*])
+ uploadConfig(rules: String[0..*])
+ Store response(response:String)
+ FindResponse():String

+ InvalidateCache()

+ autenticator

<<root>>Autentication Provider

Token List: String[0..*] {unique}

AuthConfig:String[1..*] {unique}

+ uploadConfig(rules: String[0..*])
+ Authenticate user(user:String):String
+ validateToken(token:String):boolean
+ expireAllTokens()