# Documento de Arquitectura Técnica - Módulo Backend Core

## 1. Propósito del Módulo

El módulo `backend` constituye el núcleo funcional y arquitectónico del sistema, sirviendo como base para los módulos específicos de negocio (por ejemplo, usuarios, pedidos, pagos, etc.). Está construido con Kotlin utilizando el framework Ktor, y soporta despliegue tanto embebido como serverless.

## 2. Tecnologías y Frameworks

- Ktor: Framework principal para la construcción del servidor HTTP y APIs.

- Kodein DI: Sistema de inyección de dependencias utilizado para registrar e inyectar funciones de negocio.

- Gson: Serialización/deserialización de objetos JSON.

- AWS Cognito: Utilizado para validación de tokens JWT.

- AWS Lambda: Permite ejecutar el backend como funciones serverless.

- RateLimiter (TokenBucket): Límite de tasa de peticiones por usuario.

## 3. Componentes Principales

- Application.kt:

- Inicializa el servidor embebido (Netty).

- Instala RateLimiting para prevenir abuso.

- Configura la inyección de dependencias con Kodein.

- Expone la ruta POST “/{business}/{function}”

que despacha llamadas a funciones registradas en el contenedor DI.

Donde {business} es el nombre de un negocio que tiene que haber sido dado de alta previamente en la aplicacion con el endpoint correspondiente, y function es el nombre declarado para la funcion dinamica que atendera el request en el module de Kodein.

Por ejemplo, la siguiente declaracion del module disponibiliza la funcion “signup” que para ser invocada debe coincidir con el parametro {function} de la ruta que se expone

*bind*<Function> (tag="signup") **{**

*singleton* **{** SignUp(*instance*(), *instance*(), *instance*()) **}**

**}**

En definitiva, la ruta dinamica “/{business}/{function}” expondra tantos endpoints como declaraciones de funciones existan en el module.

- Ruteo Dinámico de Funciones:

- Las funciones se registran con un tag dentro de Kodein.

- Se determina la función desde la URL y se instancia mediante DI.

- Function & SecuredFunction:

- Function es una interfaz genérica que obliga a la implementacion de una nueva funcion / endpoint en la aplicacion. Para su ejecucion se invoca al metodo “execute”.

- SecuredFunction extiende Function y valida tokens JWT. Busca en headers["Authorization"]la existencia de un token valido para permitir continuar con la ejecucion. Para su ejecucion se invoca a execute al igual que Function.

- LambdaRequestHandler:

- Permite reutilizar la lógica de funciones en AWS Lambda. Y la implementacion del ruteo es similar a Application con rutas dinamicas del estilo “/{business}/{function}”.

- Request y Response:

- Clases genéricas base para entrada y salida de datos. Considerar que ambas son heredadas por las clases que quieren exponer la interfaz / mensaje de comunicación para funciones / enpoints que sean declarados dinámicamente en el module.

Para saber cual es el request que pertenece a una determinada funcion / endpoint se puede revisar por nomenclatura, ya que sera el [nombre de la funcion]Request o se puede revisar las instrucciones de la funcion execute, en sus primeras lineas tendran alguna instruccion como esta de ejemplo:

Gson().fromJson(textBody, ar.com.intrale.SignUpRequest::class.*java*)

Donde el primer parametro es el request en formato String, y el segundo parametro es la estructura o interfaz de nuestro request.

Cuando querramos saber que retorna nuestra funcion / endpoint tendremos que revisar los return de la funcion execute para tener claro los Response o clases que heredan de response.

* De esta manera se puede obtener claramente rutas expuestas dinamicamente, funciones que atienden esas rutas dinamicas y relacion entre estas (mapeos) e interfaces de comunicacion para cada una de ellas tanto request como response.

- Excepciones Personalizadas:

- RequestValidationException, UnauthorizedException, ExceptionResponse, etc.

## 4. Ejecución en Ambientes

- Modo Local / Embebido:

- Utiliza embeddedServer(Netty).

- Modo AWS Lambda:

- Ejecuta LambdaRequestHandler con DI.

- Decodifica el body Base64 y expone cabeceras para CORS.

## 5. Patrón de Modularización

Este backend actúa como una superclase funcional. Los módulos de negocio importan este backend y registran sus propias funciones bajo nombres específicos, permitiendo:

- Reutilización de lógica común.

- Separación de responsabilidades.

- Escalabilidad horizontal por servicio.

## 6. Diagrama de interacción de componentes

La siguiente imagen ilustra el flujo de interacción entre los componentes clave del backend:

