Diseño estructural - TaskHub

Version 1.

Omar Yesid Hernandez Sotelo
William Cuello Haydar

Universidad Tecnologica De Bolivar

29 de marzo de 2025

Índice

1. Introducción

TaskHub es una plataforma de gestión de proyectos basada en una arquitectura de microservicios que implementa patrones de diseño fundamentales para garantizar escalabilidad, mantenibilidad y robustez. Este documento describe la estructura del sistema, compuesta por servicios especializados en autenticación, gestión de proyectos, procesamiento de documentos y notificaciones, coordinados a través de un API Gateway central.

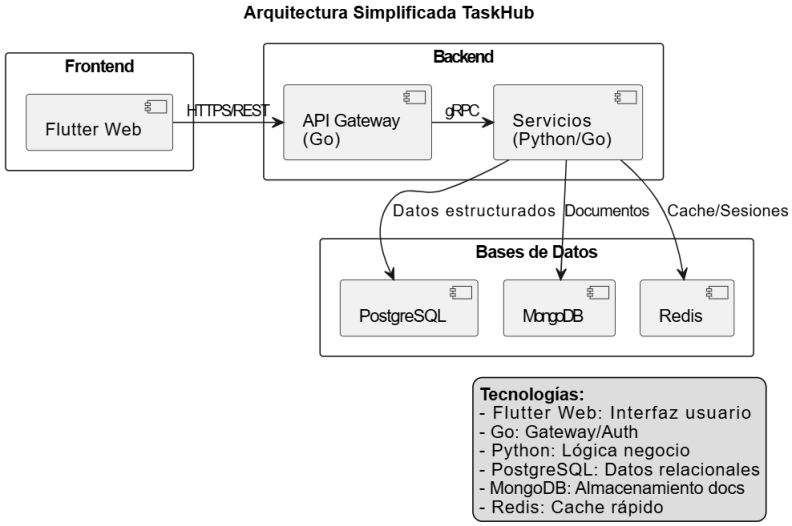

Figura 1: Diagrama por capas del Proyecto

2. Diagrama de clases

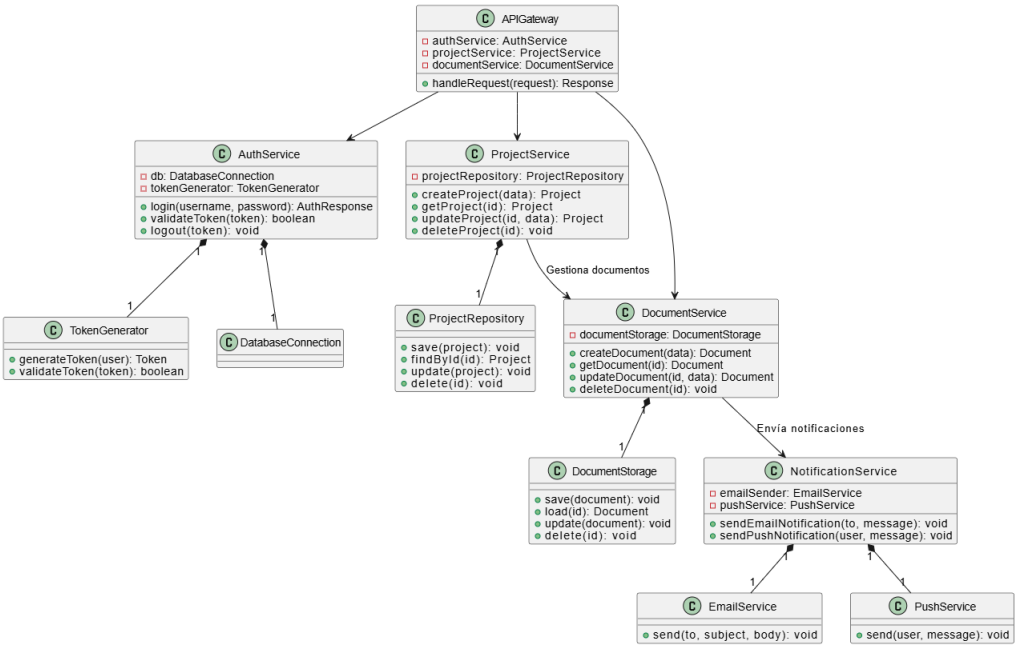

Figura 2: Diagrama de clases de TaskHub

3. Descripción de los Microservicios

3.1. Servicio de Autenticación (AuthService)

- **Función principal**: Gestiona la autenticación de usuarios y la generación/validación de tokens.
- **Componentes**:
  - `DatabaseConnection`: Conexión a la base de datos de usuarios
  - `TokenGenerator`: Genera y valida tokens de acceso
- **Operaciones clave**:
  - `login(username, password)`: Autentica credenciales
  - `validateToken(token)`: Verifica validez de tokens
  - `logout(token)`: Invalida tokens

3.2. Servicio de Proyectos (ProjectService)

- **Función principal**: Gestiona el ciclo de vida de los proyectos.
- **Componentes**:
  - `ProjectRepository`: Persistencia de datos de proyectos
- **Operaciones clave**:
  - CRUD completo de proyectos (`createProject`, `getProject`, etc.)

3.3. Servicio de Documentos (DocumentService)

- **Función principal**: Maneja la creación y gestión de documentos.
- **Componentes**:
  - `DocumentStorage`: Almacenamiento persistente de documentos
- **Operaciones clave**:
  - CRUD completo de documentos

3.4. Servicio de Notificaciones (NotificationService)

- **Función principal**: Envía notificaciones a usuarios.
- **Componentes**:
  - `EmailService`: Envío de correos electrónicos
  - `PushService`: Envío de notificaciones push
- **Operaciones clave**:
  - `sendEmailNotification(to, message)`
  - `sendPushNotification(user, message)`

3.5. API Gateway

- **Función principal**: Punto único de entrada para todas las solicitudes.
- **Responsabilidades**:
  - Enrutamiento de peticiones a los servicios adecuados
  - Validación inicial de tokens con AuthService
  - Balanceo de carga y gestión de errores
- **Integraciones**:
  - Se comunica con todos los servicios principales

4. Diagrama de clases con Patrones de Diseño

4.1. Descripción General

La arquitectura implementa seis patrones fundamentales distribuidos en cuatro microservicios y un API Gateway:

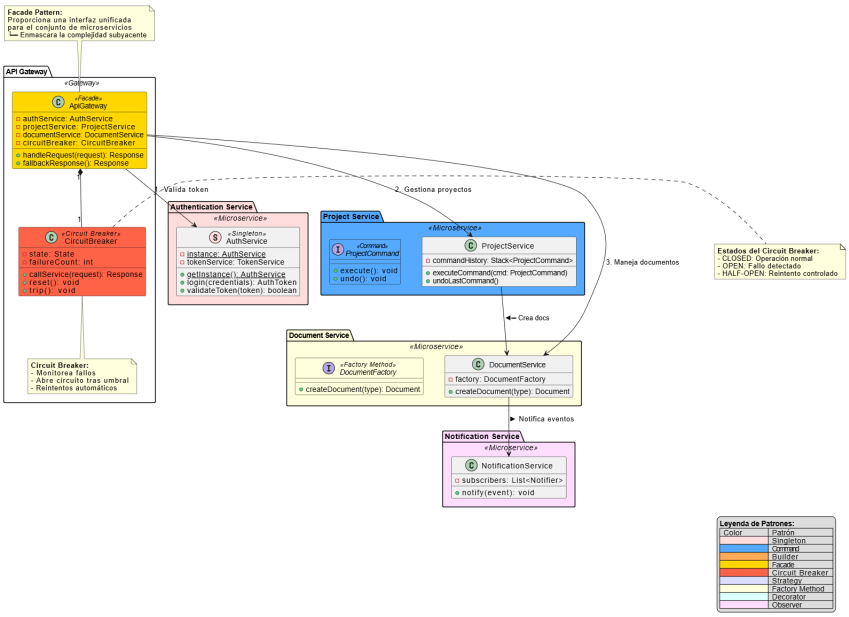

Figura 3: Diagrama de clases de TaskHub con Patrones de diseño

4.2. Patrones Implementados

4.2.1. API Gateway

- **Facade**: Unifica el acceso a los microservicios
- **Circuit Breaker**: Gestiona fallos con tres estados (abierto/cerrado/semi-abierto)

4.2.2. Servicios Principales

Servicio	Patrones
Autenticación	Singleton (instancia global)
Proyectos	Command (operaciones undo/redo)
Documentos	Factory Method + Decorator
Notificaciones	Observer (suscripciones)

4.3. Interacciones Clave

- El API Gateway valida tokens con AuthService (Singleton)
- ProjectService usa comandos para gestionar operaciones reversibles
- DocumentService decora documentos con funcionalidad adicional
- Notificaciones se distribuyen a observadores registrados

4.4. Ventajas Arquitectónicas

- **Resiliencia**: Circuit Breaker previene fallos en cascada
- **Flexibilidad**: Patrones Command y Decorator permiten extensibilidad
- **Coherencia**: Singleton garantiza un único punto de autenticación
- **Acoplamiento débil**: Observer mantiene servicios independientes

5. Flujos de Comunicación

- El `APIGateway` valida siempre los tokens con `AuthService` antes de procesar solicitudes.
- `ProjectService` interactúa con `DocumentService` para gestionar documentos asociados a proyectos.
- `DocumentService` notifica a `NotificationService` cuando se completan operaciones importantes.

6. Ventajas de la Arquitectura

- **Escalabilidad**: Cada servicio puede escalar independientemente
- **Resiliencia**: Fallos en un servicio no afectan a los demás
- **Despliegue independiente**: Cada servicio puede actualizarse sin afectar al sistema completo
- **Tecnologías heterogéneas**: Cada servicio puede usar la tecnología más adecuada