Diseño de Microservicios Basado en el Dominio

MitoCode Network

Por: Andres Gonzales

MitoCode

Agenda

- Entender por qué **no partimos del código**, sino del **modelo del negocio**
- Aplicar **DDD** para diseñar microservicios relevantes
- Introducir el concepto de **Bounded Context**
- Empezar a diseñar la arquitectura del sistema base del curso

MitoCode

¿Por qué diseñar bien importa?

Mal diseño = servicios acoplados, duplicación de lógica, cambios lentos

Buen diseño = independencia, escalabilidad, claridad de responsabilidades

“**No es sobre microservicios pequeños, sino bien delimitados.**”

El monolito de “Usuarios”

Una startup crea su sistema con un único microservicio llamado `UserService`. Este servicio tiene muchas responsabilidades relacionadas con los usuarios:

Funciones dentro de `UserService`:

Envío de emails:
-Email de bienvenida, recuperación de contraseña, verificación de correo

Gestión de permisos:
-Roles de usuario (admin, moderador, cliente)
-Control de acceso a funcionalidades de la plataforma

Autenticación:
-Validación de credenciales
-Generación y renovación de tokens JWT

Gestión de Perfil:
-Datos personales (nombre, edad, dirección, etc.)
-Preferencias del usuario

Subida de fotos de perfil:
-Manejo de archivos, almacenamiento en S3 o disco

MitoCode

Consecuencias de este diseño

Cambios lentos

Al cambiar el proceso de autenticación (po se ve afectada toda la lógica de usuarios.

Los cambios en los emails o las fotos requ el servicio.

Errores frecuentes

Un bug en el envío de correos (ej. una libr login funcione correctamente.

Un error en la carga de imágenes podría t afectando la autenticación también.

Despliegues riesgosos

Como todo está junto, cualquier cambio ob idad del servicio.

Se hace más difícil escalar solo una parte (ej. subida de fotos), porque todo está acoplado.

MitoCode

Solución con microservicios por contexto

Separar el `UserService` en microservicios basados en **Bounded Contexts**, cada uno con su propio modelo de datos y despliegue independiente:

AuthService:
Login, tokens, refresh, logout

ProfileService:
Datos del usuario, nombre, dirección

EmailService:
Envío de correos con templates y lógica propia

PhotoService:
Almacenamiento, validación y compresión de imágenes

PermissionService:
Roles, permisos, control de acceso

MitoCode

Conclusión

Un microservicio no debería ser una agrupación de funcionalidades “parecidas” técnicamente (todo lo de usuarios), sino una unidad coherente de negocio. Dividir por contexto de dominio reduce errores, mejora el tiempo de respuesta al cambio y permite escalar cada servicio según sus necesidades.

Estrategias de diseño de microservicios

Dividir por tecnología

`FrontendService`

`BackendService`

`DatabaseService`

`NotificationService`

`IntegrationService`

Dividir por entidad

`UserService`

`OrderService`

`ClientService`

`ProductService`

Dividir por dominio

`CatalogoService`: maneja productos, sus precios y disponibilidad.

`CarritoService`: gestiona el carrito de compra de cada usuario.

`OrdenesService`: maneja la creación y seguimiento de pedidos.

`PagosService`: se encarga del procesamiento de pagos.

`EnviosService`: maneja el despacho y seguimiento de entregas.

MitoCode

¿Qué es DDD (Domain-Driven Design)?

DDD (Domain-Driven Design) es un enfoque de diseño de software propuesto por **Eric Evans** que pone el foco en el **modelo del dominio** y en la **colaboración con expertos del negocio** para construir software basado en una comprensión profunda del problema que se quiere resolver.

Principales objetivos de DDD:

- Modelar el **núcleo del negocio** de forma precisa y expresiva.
- Alinear el software con el **lenguaje y reglas del dominio real**.
- Dividir el sistema en **Bounded Contexts** claros y bien definidos.
- Favorecer un diseño basado en **entidades, value objects, aggregates, servicios y eventos** del dominio.

¿Qué es DDD (Domain-Driven Design)?

Conceptos clave:

- **Ubiquitous Language** (lenguaje común entre devs y negocio)
- **Entidades, Value Objects, Aggregates**
- **Bounded Context** (contexto delimitado)
- **Context Mapping** (relaciones entre contextos)

¿Qué NO es un buen microservicio?

Ejemplos comunes a evitar:

- "UtilsService", "NotificationService" como contenedor genérico
- Microservicios que solo exponen una tabla CRUD
- Servicios sin reglas de negocio claras

Conclusiones y siguientes pasos

Un microservicio nace de una **necesidad del negocio**, no de una tabla o un archivo.

DDD es útil para entender y modelar correctamente esos dominios.

Vamos a usar estos conceptos para construir la arquitectura real del curso.

Caso de negocio del curso (introducción)

Plataforma de pedidos en línea (tipo Rappi o PedidosYa)

MitoCode

MitoCode Network

www.mitocode.com