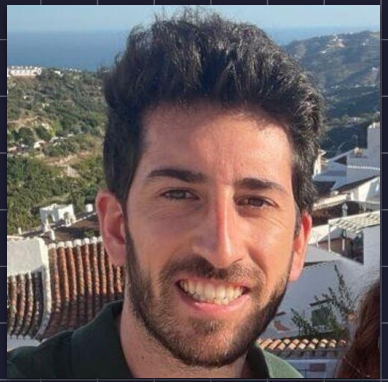
keep coding

Desarrollo web Full Stack

Backend Avanzado



Software Engineer at Cobee

linkedin.com/in/eduaguilarweb

Creación proyecto base

Wallapop backend

Creación de proyecto usando ExpressJS y Mongodb
Eslint y Prettier
Live-reload
Autenticación via JWT
Securización endpoints
Gestión de roles
Validación de parámetros usando Joi
Gestión de variables de entorno
y mucho más...

ExpressJS y Mongodb

ExpressJS

Framework ligero y flexible que simplifica la creación de APIs REST mediante un sistema claro de rutas, middlewares y controladores.

Mongodb

Base de datos NoSQL orientada a documentos, adecuada para aplicaciones modernas por su esquema flexible, fácil integración con JSON y buena escalabilidad horizontal.

ESLint y Prettier

keep coding

ESLint

Herramienta de análisis estático de código que detecta errores, malas prácticas y problemas de calidad en el código JavaScript.

Prettier

Herramienta de formateo automático de código.

keep coding

Let's Code

1. Creación de proyecto usando ExpressJS, Mongodb y Docker.
2. Configuración ESLint y Prettier.
3. Live-reload.
4. CRUD de productos.

keep coding

Let's Think

1. Analizar si estamos cómodos con nuestro código

Typescript

keep coding

¿Qué es?

TypeScript es un superset de JavaScript. Esto significa que:

- Todo código JavaScript válido es compatible con TypeScript
- TypeScript añade funcionalidades adicionales sobre JavaScript, siendo la principal el tipado estático.
- El código TypeScript no se ejecuta directamente, sino que se transpila a JavaScript

Typescript

¿Por qué?

Básicamente nos permite una detección temprana de errores. JavaScript es un lenguaje de tipado dinámico, lo que permite flexibilidad, pero también hace que muchos errores solo se detecten en tiempo de ejecución.

```
let age: number = 25;
age = "25";          // Error de tipo
```

Typescript

keep coding

Beneficios

- Código más explícito y legible.
- Mejora la experiencia de desarrollo (autocompletado, detección de errores)
- Escalabilidad y mantenimiento.
  - reduce errores derivados de cambios de código
  - Facilita refactorizaciones
  - Define contratos claros entre distintas partes del sistema
- Se puede usar de forma progresiva
- Inferencia de tipos
- Es un estándar en la industria

Typescript

keep coding

Inconvenientes

- Mayor complejidad inicial
  - tipado
  - configuración + traspilación
- Mayores restricciones

keep coding

Let's Code

1. Incluyamos typescript en nuestro proyecto

Testing e2e

¿Qué es?

Son pruebas que validan el flujo completo de una aplicación, simulando la experiencia real del usuario desde el inicio hasta el final para asegurar que todos los componentes y sistemas integrados funcionen correctamente juntos.

Testing e2e

¿Por qué?

Cuando desarrollamos una API, queremos estar seguros de:

- Los endpoints funcionan correctamente.
- Los datos se guardan bien en la base de datos.
- Las reglas de negocio se cumplen correctamente.
- Los cambios futuros no rompen funcionalidades existentes.

El testing E2E (End-to-End) nos ayuda a comprobar todo esto de principio a fin, como lo haría un usuario real.

Testing e2e

¿En qué consiste?

Simular una petición real a nuestra aplicación y comprobar la respuesta final. En una API como la nuestra, esto significa:

- Realizar peticiones HTTP reales.
- Pasar por: rutas, controladores, lógica de negocio y base de datos.
- Recibir una respuesta HTTP real.

No mockeamos nada.
Probamos el sistema en su totalidad.

Testing e2e

keep coding

Beneficios

1. Si un test E2E pasa, hay muchas probabilidades de que funcione en producción.
2. Identifica errores en cualquier punto de nuestro código.
   a. Definición de rutas.
   b. Lógica de negocio no válida.
   c. Errores al interactuar con la base de datos.
3. Evita regresiones. Cuando cambiamos nuestro código, estamos protegidos por los tests.

Testing e2e

Inconvenientes

1. Son más lentos.
2. Son algo más complejos de escribir.

Testing e2e

¿Cómo?

1. Jest - ejecutor de tests, aserciones, etc
2. Supertest - haremos las peticiones http
3. mongodb-memory-server - base de datos para tests

keep coding

Let's Code

1. Incluyamos tests e2e para nuestra gestión de productos.

Refactorización del proyecto

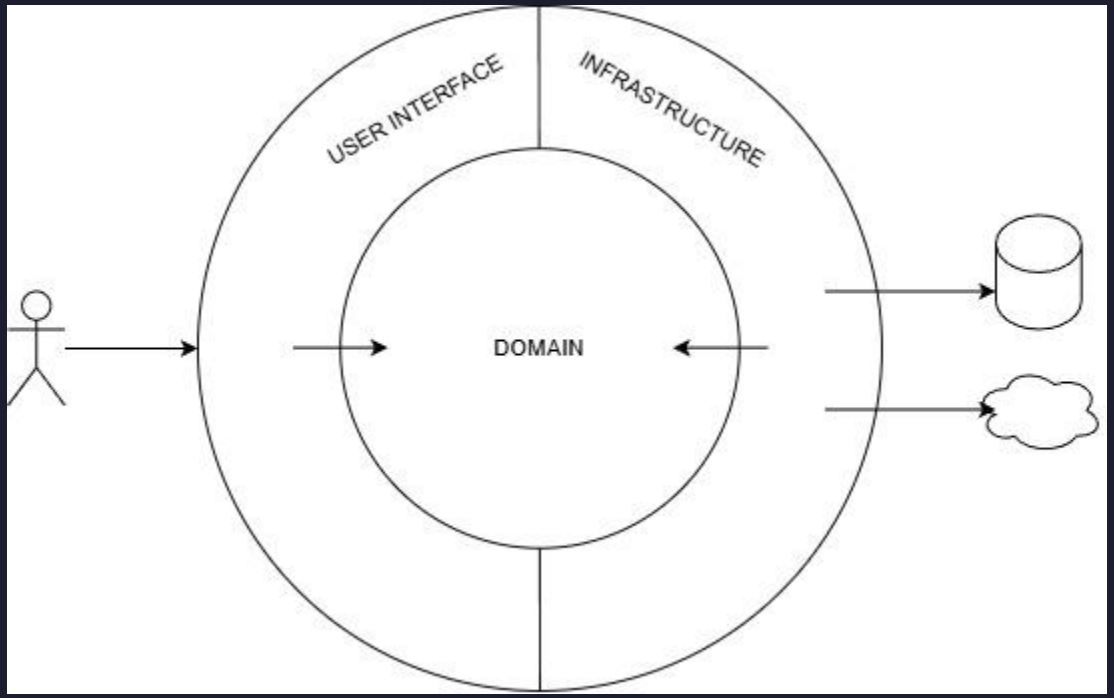
keep coding

Actualmente

- Controladores con lógica.
- Modelos de Mongoose usados en el controlador.
- Ficheros muy largos.
- Testing unitario imposible.

El problema no es Mongo ni Express, es **mezclar responsabilidades.**

Arquitectura hexagonal

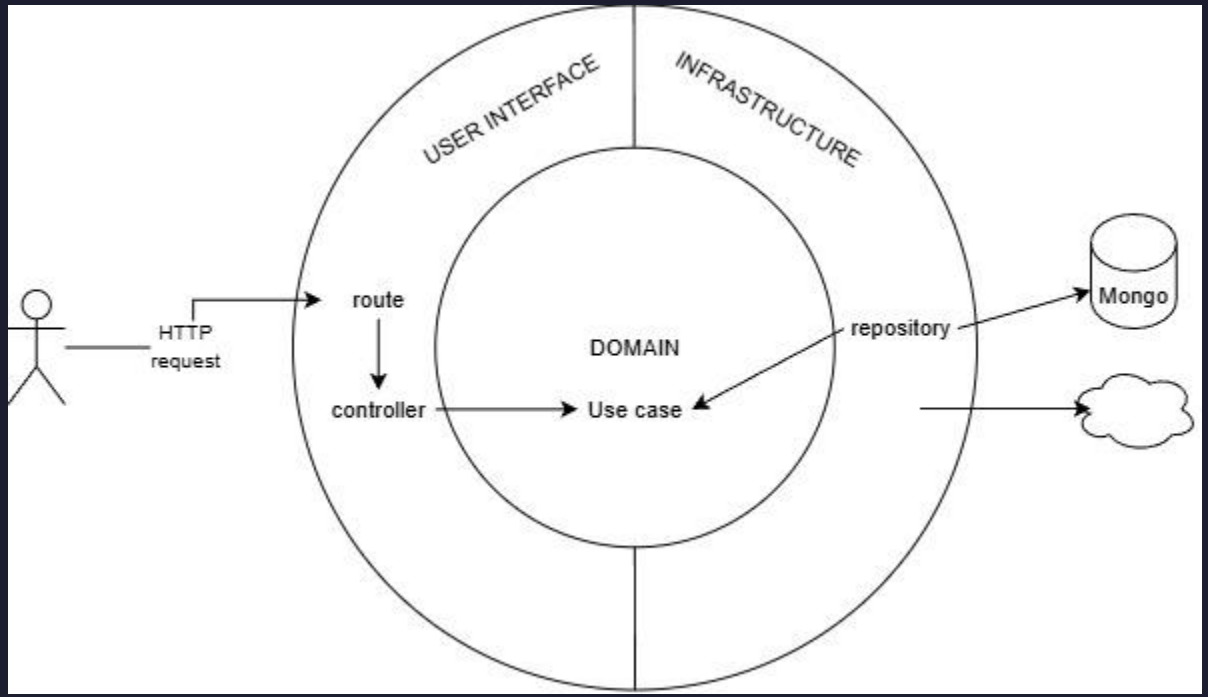


Arquitectura hexagonal

keep coding

Regla de oro
El dominio **no depende** de nada externo

Arquitectura hexagonal



Arquitectura hexagonal

keep coding

❌ Esto no:

- Lógica en el controller.
- ProductModel.create() dentro del caso de uso.
- req y res en el dominio.

✅ Esto sí:

- Controller → prepara y valida datos.
- Caso de uso → toma decisiones.
- Repositorio → guarda y lee datos.

Arquitectura hexagonal

keep coding

¿ Por qué ?

No usamos arquitectura hexagonal porque esté de moda. La usamos porque hace el código más:

- Mantenible
- Testeable
- Fácil de cambiar

Refactorización del proyecto

Nueva estructura de carpetas

```
src/
├── api/          → routes, controllers, middlewares.
├── domain/       → casos de uso y lógica de negocio.
└── infrastructure/ → bases de datos, Mongo, servicios externos
```

keep coding

Let's Refactor

1. Apliquemos lo aprendido de arquitectura hexagonal a la creación de productos

keep coding

Let's Challenge

1. Refactorizar el borrado de producto usando hexagonal.

Autenticación

JWT

Un JWT (JSON Web Token) es un estándar que define una forma segura y compacta de transmitir información entre dos partes en formato JSON.

Se usa principalmente para autenticación y autorización en APIs REST.

HEADER.PAYLOAD.SIGNATURE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30

Autenticación

keep coding

Header
Describe el tipo de token y el algoritmo de la firma

Payload
Contiene los datos que identifican al usuario

Signature
Firma criptográfica del token, creada con: header, payload y una clave secreta.

Autenticación

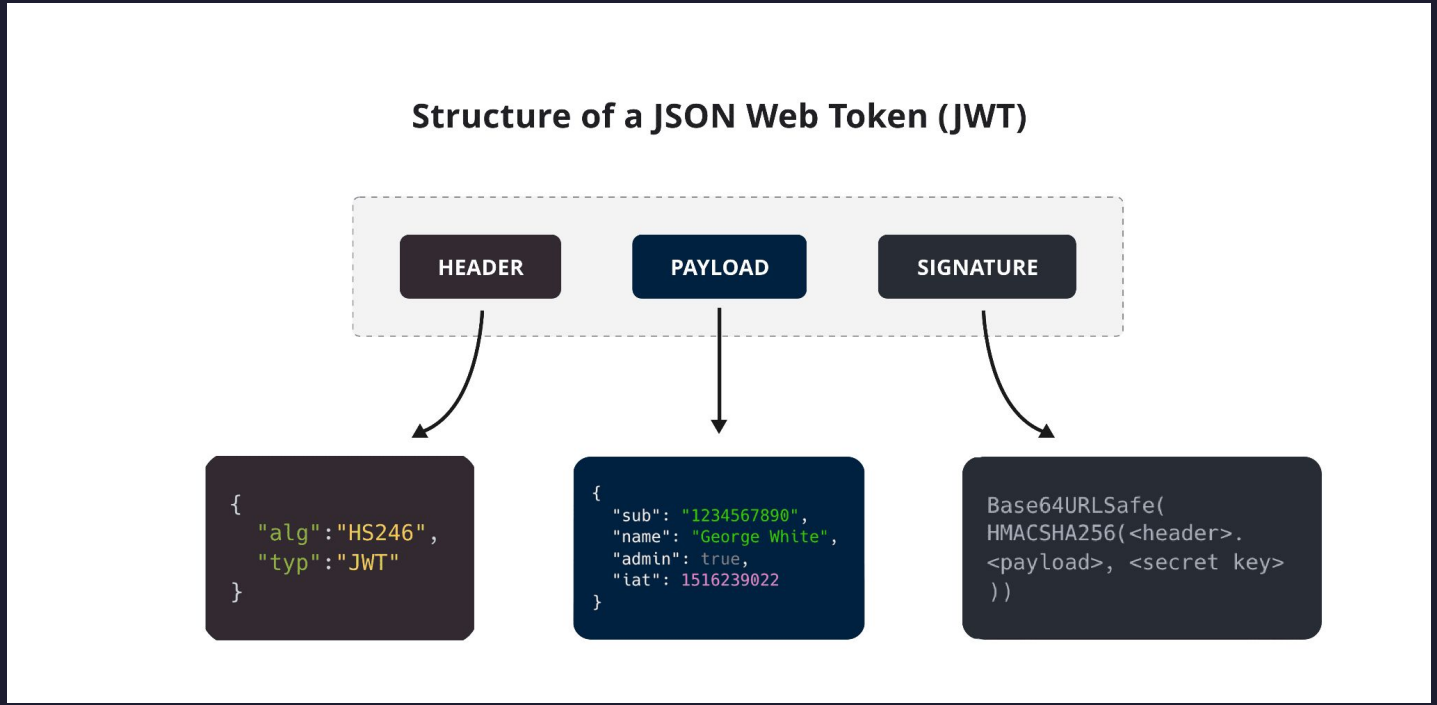
Usos de JWT

1. Identificar a un usuario autenticado.
2. Proteger rutas de una API.
3. Compartir información de forma segura entre servicios.
4. Evitar el uso de sesiones almacenadas en el servidor (stateless).

Características clave

1. Stateless: el servidor no guarda información de sesión.
2. Autocontenido: el token incluye toda la información necesaria.
3. Portable: se envía fácilmente por HTTP.
4. Escalable: ideal para sistemas distribuidos.

Autenticación



Autenticación

keep coding

JWT !== encriptación

Un JWT no cifra los datos, FIRMA los datos.

Esto significa que cualquier puede leer el contenido, pero nadie puede alterarlo sin la clave secreta

Autenticación

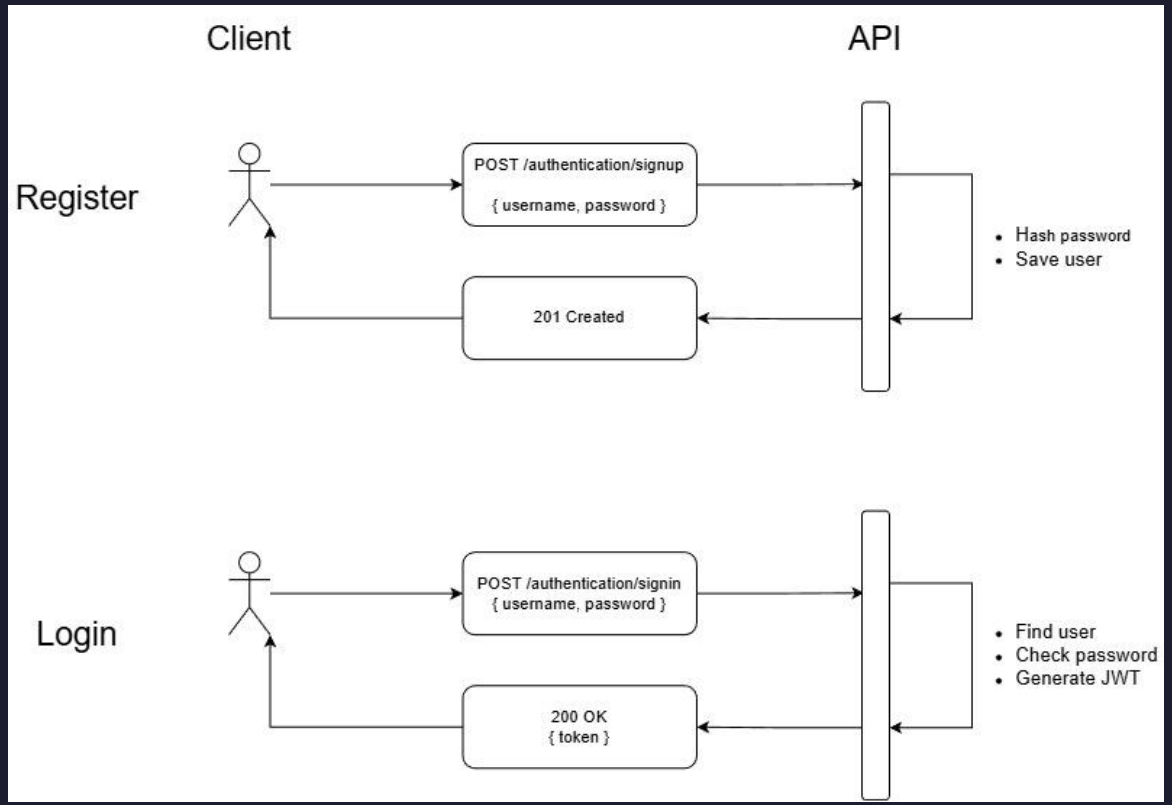
keep coding

Autenticación vs Autorización

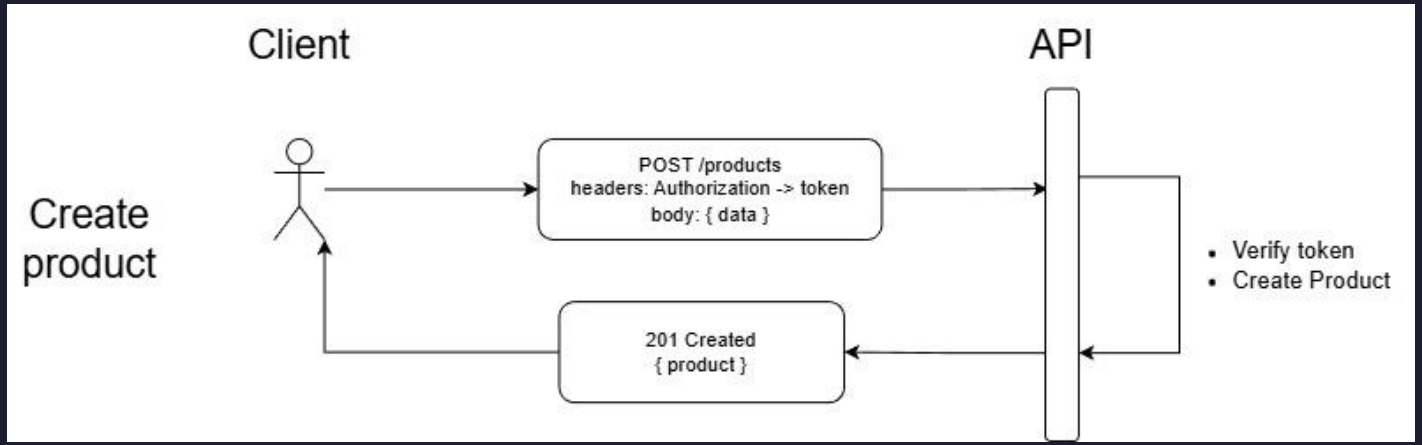
Autenticación -> ¿Quién eres?

Autorización -> ¿Qué puedes hacer?

Autenticación



Autenticación



keep coding

Let's Code

1. Añadir endpoints para registro y login.
2. Relacionar modelo User y Product.
3. Uso de middleware de autenticación.
4. Implementar reglas de seguridad sobre los productos.

Validación de datos

keep coding

Joi

Permite definir esquemas que describen cómo debería ser la información que recibe tu aplicación y validar objetos contra esos esquemas.

1. Evitamos errores de formato
2. Centralizamos y simplificamos la validación
3. Mensajes de error concretos
4. Robustez

Autorización

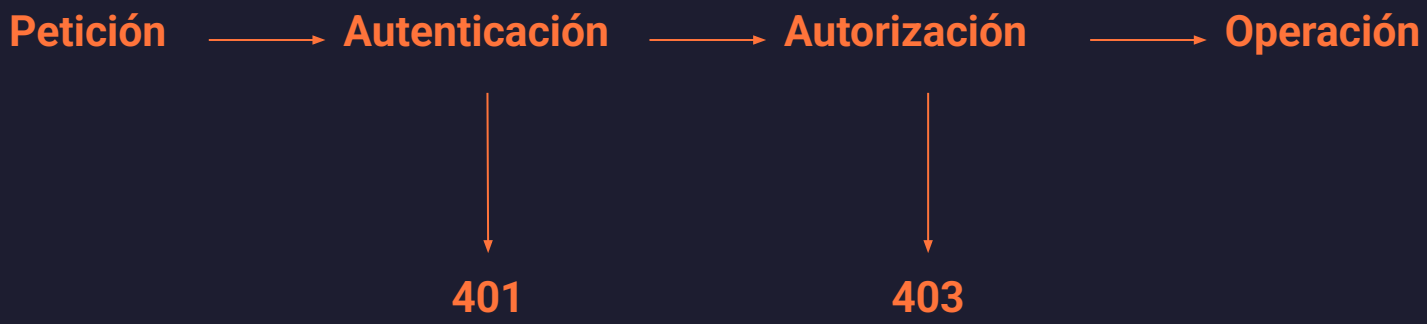
Roles

La autorización basada en roles (RBAC – Role-Based Access Control) es un mecanismo para controlar qué acciones puede realizar un usuario en un sistema, según su rol.

Conceptos

1. Rol: un conjunto de permisos que define qué puede y qué no puede hacer un usuario.
2. Usuario: se le asigna uno o varios roles.
3. Permiso: acción concreta que se puede permitir o denegar (ej. “crear usuario”, “eliminar artículo”, “acceder a /admin”).

Autorización



keep coding

Let's Code

1. Añadir validadores para la creación de productos.
2. Incluir perfilado de usuarios.
3. Borrado de usuarios.

keep coding

Let's Challenge

1. Permitir a los admins borrar cualquier producto.

Variables de entorno

Nuestra aplicación no se ejecutará siempre en nuestra máquina. Ahora mismo, código como:

```
await mongoose.connect('mongodb://admin:admin123@localhost:27017/db?authSource=admin');
```

Es algo que hará que nuestra aplicación no pueda ser accesible por otros usuarios.

Variables de entorno

Por otra parte, código como este:

```
const token = jwt.sign({ id: user.id, role: user.role }, 'your_jwt_secret', { expiresIn: '1h' });
```

Supone un problema de seguridad enorme.

Variables de entorno

❌ Problemas de seguridad

- El secreto del JWT queda en el código
- Si subimos el repo a GitHub, el secreto queda expuesto
- Cualquiera con acceso al repo puede generar tokens válidos

❌ Problemas de despliegue

- En local usamos una base de datos
- En producción usamos otra
- En staging usamos otra distinta

Variables de entorno

¿Qué es?

Una variable de entorno es un valor que:

- No vive en el código
- Vive en el entorno donde se ejecuta la aplicación
- Se inyecta en tiempo de ejecución

El código no cambia entre entornos, solo cambia la configuración

Variables de entorno

keep coding

¿Cómo las identifico?

Si algún dato cambia según el entorno en el cual es ejecutado, entonces no debe estar en el código, sino en una variable de entorno. Ejemplos claros:

- credenciales de MongoDB
- Secreto de JWT
- Puerto del servidor
- Credenciales de admin

Variables de entorno

¿Qué es un entorno?

Es el contexto donde se ejecuta una aplicación, que define cómo se comporta y con qué recursos trabaja. El código es el mismo, lo que cambia es:

- Configuración
- Servicios externos
- Nivel de seguridad y observabilidad

Variables de entorno

¿Para qué sirven los entornos?

Principalmente sirven para:

- Separar responsabilidades
- Reducir riesgos
- Probar funcionalidades sin afectar a usuarios reales

Variables de entorno

keep coding

Entornos habituales - Development

Sirve para:

- Desarrollar nuevas funcionalidades
- Debuggear errores
- Iterar rápido

Aquí el objetivo es comodidad del desarrollador, no seguridad ni rendimiento.

Variables de entorno

Entornos habituales - Test / Staging / Preproduction

Sirve para:

- Ejecutar tests unitarios
- Ejecutar tests e2e
- Garantizar que el sistema funciona antes de desplegar

Entorno fundamentalmente orientado a pruebas, ya sean técnicas o funcionales.

Variables de entorno

keep coding

Entornos habituales - Production

Sirve para:

- Atender tráfico real
- Ejecutar la aplicación en casos reales

Cualquier error aquí afecta a usuarios reales

keep coding

Let's Code

1. Instalar dotenv
2. Crear servicio de carga de variables de entorno
3. Eliminar todos los datos sensibles que estén hardcodeados

Gestión de errores

keep coding

Gestión de errores

Ahora mismo, en la mayoría de nuestros casos:

```
export const anyController = async (req: Request, res: Response) => {
  try {
    // input validation
    // business logic
    // api response
  } catch (error) {
    res.status(400).json({ message: error instanceof Error ? error.message : 'Error doing something' });
  }
};
```

Gestión de errores

Gestión de errores

Objetivos

- ❌ El dominio no conoce HTTP ni ExpressJS
- ✅ El dominio lanza errores expresivos
- ✅ La capa API traduce errores a HTTP
- ✅ Hay un único punto de gestión de errores

Los casos de uso solo lanzan errores de dominio. La capa API decide cómo se convierten en respuestas HTTP

Gestión de errores

Gestión de errores

Posibles errores ahora mismo:

- Errores de dominio: reglas de negocio, entidades inexistentes, duplicadas...
- Errores de validación
- Errores técnicos: MongoDb, JWT
- Cualquier error no contemplado

Gestión de errores

Implementación en dominio

- Crearemos un DomainError base, del que todos extenderán
- Errores específicos que dependen de DomainError: EntityNotFoundError, BusinessConflictError, UnauthorizedOperationError

El error describe el problema de negocio, no el HTTP status.

Gestión de errores

keep coding

Gestión de errores
Ejemplos

```
if (!product) {
  throw new EntityNotFoundError('Product', productId);
}

if (product.ownerId !== userId) {
  throw new UnauthorizedOperationError();
}
```

Gestión de errores

Implementación en API:

- Los controladores dejarán de gestionar errores (bye bye try/catch)
- Incluiremos un middleware global para nuestra API que se encargará de traducir los errores a errores HTTP

keep coding

Let's Code

1. Diseño de errores
2. Limpieza de controladores
3. Implementación de middleware de errores

Observabilidad

keep coding

¿Y qué pasa con los errores?

Pensemos sobre lo siguiente:

- ¿Qué pasa si sucede un error gestionado? ¿Es grave?
- ¿Qué pasa si sucede un error NO gestionado?

Tenemos que ser capaces de entender qué pasa en nuestro software cuando no estamos delante del ordenador

Observabilidad

keep coding

¿Qué queremos observar?

Limitaremos el scope a:

- Errores no controlados (500)
- Errores de dominio relevantes

Siendo de mucha ayuda datos relacionados como:

- endpoint y método HTTP
- usuario
- datos relacionados con la acción

Observabilidad

keep coding

¿Cómo?

Nos apoyaremos en plataformas externas para registrar estos errores.

- Nuestro middleware de errores se encargará de enviarlos
- No dependeremos de un log en consola
- Podremos configurar avisos
- Tendremos la suficiente información como para poder solucionarlos

Usaremos una plataforma llamada **Sentry**

Observabilidad

Sentry

Sentry proporciona un seguimiento distribuido de extremo a extremo, lo que permite a los desarrolladores **identificar** y **depurar** problemas de rendimiento y errores en sus sistemas y servicios.
@sentry/node

keep coding

Let's Code

1. Integraremos Sentry en nuestro sistema
2. Probaremos cómo funciona

Rendimiento

Paginación

Ahora mismo, nuestro endpoint de búsqueda de productos devuelve **todos** los documentos que haya en la colección. Esto significa:

- Puede consumir mucha memoria en el servidor
- Generar respuestas lentas
- Devolver más datos de los necesarios
- Empeorar la experiencia del usuario

Rendimiento

Paginación

La paginación no es una optimización o mejora, es parte del diseño del endpoint. Un api profesional:

- Nunca devuelve listas infinitas
- Obliga al cliente a pedir los datos por partes
- Expone metadatos para entender los resultados

GET /products?page=1&limit=10

Rendimiento

Búsquedas

Permitir a los clientes de api hacer búsquedas es imprescindible, ya que lo necesitarán para poder ofrecer resultados concretos de manera eficiente. Olvidémonos de:

- Traernos todos los resultados del API
- hacer un .filter() de esos resultados en el front

GET /products?search=iphone

Rendimiento

Búsqueda parcial vs exacta

En nuestro caso, buscaremos por nombre, de forma parcial y case-insensitive.

- “iphone”, “iPho” o “PHONE” deberían dar los mismos resultados
- Necesitaremos un índice que facilite la búsqueda por nombre

Rendimiento

Índices

Los índices en MongoDB son estructuras de datos basadas en árboles, que almacenan un subconjunto pequeño y ordenado de los datos de una colección.

Funcionan como el índice de un libro, permitiendo a MongoDB localizar rápidamente documentos específicos sin escanear cada documento de una colección, mejorando drásticamente el rendimiento de las consultas.

Rendimiento

Índices

No deberíamos realizar búsquedas sobre campos que no tengan índices por los siguientes motivos:

- Mongo haría un *scan* sobre todos los documentos de la colección, haciendo crecer el tiempo de las búsquedas linealmente con los datos
- Cuantos más datos y más usuarios, peor experiencia

Rendimiento

Índices - tipos

En mongo existen 3 tipos de índices:

1. Índice Simple
2. Índice compuesto
3. Índice de texto

Rendimiento

Índice simple

Índice creado sobre **un solo campo** del documento. Mongo guardará los valores del campo ordenados y con referencias a los documentos. Sirve para:

- Búsquedas por igualdad
- Búsquedas por rango
- Ordenaciones

Rendimiento

Índice simple

Ventajas

1. Muy rápido
2. Poco coste de mantenimiento
3. Ideal para campos muy consultados

Inconvenientes

1. No sirve para búsquedas semánticas
2. No combina varios campos

Rendimiento

Índice compuesto

Índice que incluye más de un campo, en un orden concreto. Sirve para:

- Búsquedas que filtran por varios campos
- Optimización de consultas combinadas

Rendimiento

Índice compuesto

Ventajas

1. Muy eficiente para consultas complejas
2. Reduce aún más el número de documentos escaneados

Inconvenientes

1. Más complejo de diseñar
2. El orden de los campos importa
3. Más coste de escritura

Rendimiento

Índice de texto

Un índice especial para búsqueda textual. Sirve para:

- Búsqueda por texto avanzada
- Ideal para catálogos grandes

Rendimiento

Índice de texto

Ventajas

1. Búsqueda mucho más avanzada
2. Ideal para catálogos grandes (+50k)

Inconvenientes

1. Más pesado
2. Limitaciones(sólo 1 por colección)

keep coding

Let's Code

1. Implementemos la búsqueda de productos paginada

Tareas programadas

Cron job

Un cron job es una tarea que se ejecuta automáticamente en un momento o intervalo de tiempo definido, sin intervención del usuario.

Los cron jobs se usan cuando una acción:

1. No debe ejecutarse en tiempo real
2. No depende de una request del cliente
3. Puede ejecutarse de forma periódica o diferida

Tareas programadas

keep coding

Casos prácticos

- Limpieza de datos
  - Eliminar usuarios no verificados tras X días
  - Borrar productos soft-deleted antiguos
- Procesos automáticos
  - Cerrar pedidos caducados
  - Recalcular estadísticas

Tareas programadas

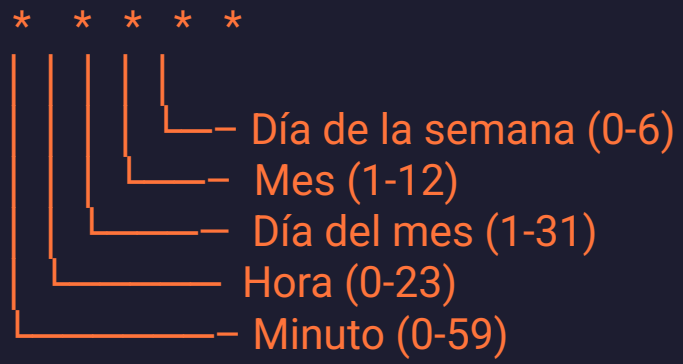
keep coding

Casos prácticos

- Mantenimiento
  - Limpieza de imágenes de productos vendidos/eliminados
  - Compactación o normalización de datos
- Notificaciones
  - Enviar emails programados
  - Recordatorios
  - Resúmenes diarios

Tareas programadas

Configuración



Ejemplos

Cada minuto → * * * * *

Cada día a medianoche → 0 0 * * *

Cada lunes a las 9 → 0 9 * * 1

crontab.guru

Tareas programadas

Configuración

Ojo con esto:

*/5 4 * * *

Cada 5 minutos durante la hora 4

Minuto: */5 → 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55
Hora: 4 → solo a las 04:00–04:59
Días/meses/día semana: cualquiera

12 veces al día

5 4 * * *

Una vez al día, a las 04:05

Minuto: 5
Hora: 4
Resto: cualquiera

1 vez al día

keep coding

Let's Code

1. Enviar un resumen semanal al usuario sobre sus productos.

keep coding

¡Muchas gracias!

www.keepcoding.io cursos@keepcoding.io (+34) 916 33 1779