




Arquitectura

 2025-01   20 min.

Arquitectura del Backend

1. Monolito
2. Microservicios
3. Serverless

Monolito

Todo el código de la aplicación está en un solo proyecto, una sola app, un solo repo.

La app se despliega completa, y si se necesita mayor capacidad, se despliega una mayor cantidad.

Microservicios

Lo que sería una aplicación "monolito" está separada en secciones, en servicios pequeños e independientes que se comunican entre si con una API interna.

Esta arquitectura permite desplegar los servicios independientes de cada uno, por lo que podrías, por ejemplo, desplegar 3 veces el servicio de usuario, pero 1 sola vez el servicio de autenticación.

Serverless

Similar a microservicios, con la diferencia que le dejas el control de la infraestructura y despliegue al proveedor del cloud hosting.

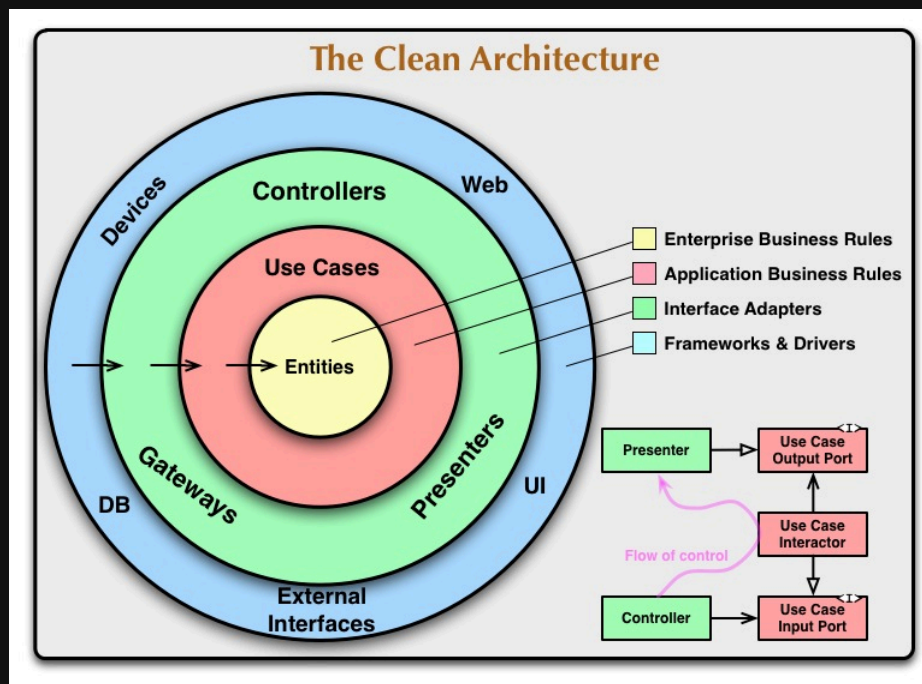
Arquitectura de la aplicación Backend

1. Clean Architecture
2. Hexagonal Architecture
3. Onion Arquitectura
4. Screaming Architecture
5. DCI
6. BCE

Arquitectura de la aplicación Backend

1. Clean Architecture
2. Hexagonal Architecture
3. Onion Arquitectura
4. Screaming Architecture
5. DCI
6. BCE

Clean Architecture



Capas Clean Architecture

1. Capa 1 > Frontend
2. Capa 2 > Rutas
3. Capa 3 > Controladores
4. Capa 4 > Casos de Uso / Acciones
5. Capa 5 > Base de Datos

Esto es un poco distinto al diagrama porque el diagrama ubica la DB en las afueras.

Capa 1 - Frontend

Esta capa es literalmente el frontend.

El frontend realiza requests al backend. La **capa 2** recibe esos requests.

Capa 2 - Rutas

Esta capa recibe los requests por parte del frontend.

Es donde se define la ruta del endpoint y que función de la **capa 3** se ejecuta.

Adicionalmente, podría ser responsable de extraer los datos del request (body, query, params).

Existe un archivo para cada modelo.

Capa 3 - Controladores

Esta capa es ejecutada por la **capa 2** con todos los datos necesarios para su ejecución pasado por parametros.

Esta capa ejecuta todas las funciones de la **capa 4** que considere necesario para realizar su labor.

Existe un archivo para cada modelo.

Capa 4 - Casos de Uso / Acciones

Esta capa es ejecutada por la **capa 3** con todos los datos necesarios para su ejecución pasado por parametros.

Esta capa realiza llamados a la **capa 5** y retorna el resultado.

Existe un archivo para cada accion de cada modelo.

Capa 5 - Base de Datos

Esta capa es literalmente la base de datos.

Pero su representación en el código es el archivo del modelo.

Estructura de proyecto

Cada capa está representada en un archivo distinto, siguiendo esta estructura:

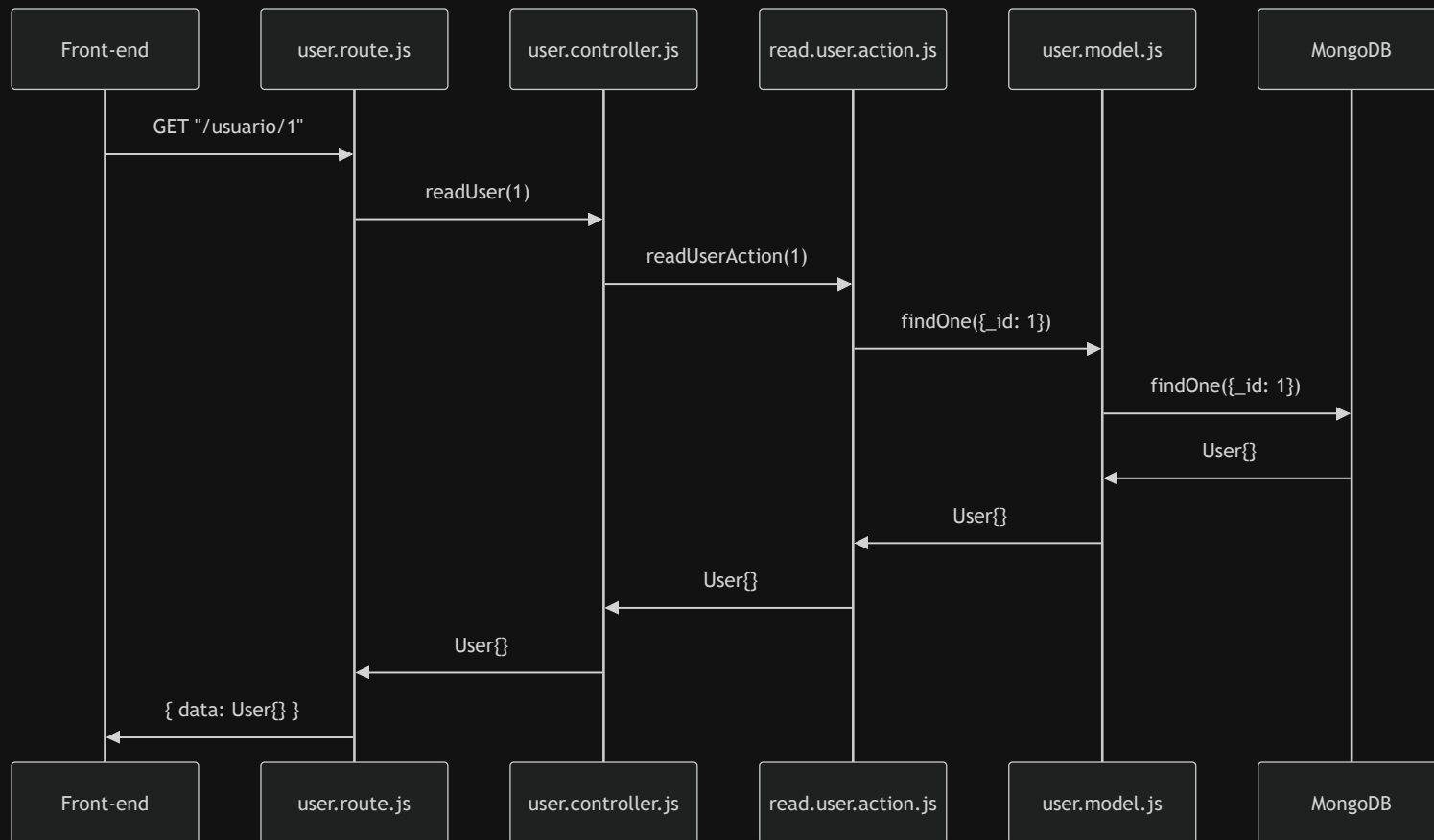
```
[modelo].[capa].js
```

Servicio de Usuarios

Entonces, para el servicio de users, tendríamos los siguientes archivos:

Archivo	Capa
user.route.js	Ruta
user.controller.js	Controlador
create.user.action.js	Acción
read.user.action.js	Acción
update.user.action.js	Acción
delete.user.action.js	Acción

Servicio de Usuarios



Estructura de proyecto

Juntamos todas la rutas en una carpeta de rutas, todos los controladores en una carpeta de controladores, etc?

O

Juntamos todo lo involucrado a un servicio en su propia carpeta?

Estructura de proyecto

Capas juntas o modelo junto?

Eso ya es preferencia propia.

Yo prefiero modelo junto.

Ejemplo Archivo de Rutas

```
1  // INIT ROUTES
2  const userRoutes = Router();
3
4  // DECLARE ENDPOINT FUNCTIONS
5  async function GetUsers(request: Request, response: Response) {
6    const users = await readUsers();
7
8    response.status(200).json({
9      message: "Success.",
10     users: users,
11   });
12 }
13
14 // DECLARE ENDPOINTS
15 userRoutes.get("/", GetUsers);
```

Ejemplo Archivo Controlador

```
1 // DECLARE CONTROLLER FUNCTIONS
2 async function readUsers(): Promise<UserType[]> {
3   const users = await readUserAction();
4
5   return users;
6 }
7
8 // EXPORT CONTROLLER FUNCTIONS
9 export { readUsers };
```

Ejemplo Archivo Acción Read

```
1  import { UserModel, UserType } from "./user.model";
2
3  // DECLARE ACTION FUNCTION
4  async function readUserAction(): Promise<UserType[]> {
5      const results = await UserModel.find();
6
7      return results;
8  }
9
10 // EXPORT ACTION FUNCTION
11 export default readUserAction;
```

Ejemplo Archivo Modelo

```
1  // DECLARE MODEL TYPE
2  type UserType = {
3    // USER FIELDS
4  };
5
6  // DECLARE MONGOOSE SCHEMA
7  const UserSchema =
8    new Schema(<UserType>{
9      // USER FIELDS
10    });
11
12  // DECLARE MONGO MODEL
13  const UserModel = model < UserType > ("User", UserSchema);
14
15  // EXPORT ALL
16  export { UserModel, UserSchema, UserType };
```



'tamos claros