Documento de normalización de la estructura del proyecto

—

Lenguaje de Marcas y Sistemas de Gestión de Información

CPIFP Alan Turing

1. Introducción

Este documento establece las normas para la organización de los proyectos Node.js desarrollados en el marco del curso. La finalidad es mantener una estructura clara, modular y bien documentada, facilitando el desarrollo, mantenimiento y escalabilidad del sistema.

---

2. Estructura del proyecto

El proyecto debe seguir la siguiente estructura de directorios:

```
/nombre-del-proyecto
├── /src
│   ├── /controllers
│   ├── /models
│   ├── /routes
│   ├── /services
│   ├── /middlewares
│   ├── /config
│   ├── /utils
│   ├── index.js
│
├── /public
│   ├── /css
│   ├── /js
│   ├── /images
│   ├── index.html
│
├── /docs
│   ├── /manuales
│   ├── /api
│   ├── README.md
│
├── /tests
├── .env
├── .gitignore
├── package.json
├── package-lock.json
```

```
├── README.md
```

Cada directorio cumple una función específica:

- **`/src`**: Contiene el código fuente del backend.
    - **`/controllers`**: Controladores que gestionan la lógica de negocio.
    - **`/models`**: Modelos de datos (ORM o esquemas en caso de usar MongoDB, Sequelize, etc.).
    - **`/routes`**: Definición de las rutas de la API.
    - **`/services`**: Servicios que encapsulan lógica reutilizable (conexiones a bases de datos, servicios externos, etc.).
    - **`/middlewares`**: Middleware para la autenticación, autorización y manejo de errores.
    - **`/config`**: Configuraciones del proyecto (base de datos, variables de entorno, etc.).
    - **`/utils`**: Funciones utilitarias y helpers.
    - **`index.js`**: Punto de entrada de la aplicación.
- **`/public`**: Contiene los archivos estáticos accesibles desde la web (frontend básico).
    - **`/css`**: Archivos CSS.
    - **`/js`**: Archivos JavaScript para la interacción del frontend.
    - **`/images`**: Recursos visuales.
    - **`index.html`**: Archivo principal del frontend.
- **`/docs`**: Documentación del proyecto.
    - **`/manuales`**: Instrucciones de uso y manual de usuario.
    - **`/api`**: Documentación de la API (puede incluir OpenAPI o Postman).
    - **`README.md`**: Introducción y descripción general del proyecto.
- **`.env`**: Archivo para variables de entorno (no debe subirse al repositorio).
- **`.gitignore`**: Archivo para excluir archivos innecesarios en el control de versiones.
- **`package.json`**: Archivo de configuración de dependencias y scripts.
- **`package-lock.json`**: Archivo de bloqueo de versiones de dependencias.

- **`README.md`**: Documentación general del proyecto.

Ejemplo de organización de la carpeta /src

📂 `src/`

El directorio principal donde reside el código fuente del backend.

📂 `controllers/` → Controladores que manejan la lógica de las peticiones HTTP

- **`userController.js`** → Manejo de usuarios (registro, login, perfil).
- **`productController.js`** → Gestión de productos.
- **`orderController.js`** → Procesamiento de órdenes y compras.

📂 `models/` → Modelos de datos (usando ORM como Mongoose o Sequelize)

- **`User.js`** → Modelo de usuario (nombre, email, contraseña).
- **`Product.js`** → Modelo de producto (nombre, precio, stock).
- **`Order.js`** → Modelo de orden (usuario, productos, estado).

📂 `routes/` → Definición de las rutas de la API

- **`userRoutes.js`** → Rutas para usuarios (`/register`, `/login`, `/profile`).
- **`productRoutes.js`** → Rutas para productos (`/products`, `/products/:id`).
- **`orderRoutes.js`** → Rutas para órdenes (`/orders`, `/orders/:id`).

📂 `services/` → Lógica de negocio y conexión con servicios externos

- **`emailService.js`** → Envío de correos electrónicos (nodemailer, SendGrid).
- **`paymentService.js`** → Integración con pasarelas de pago (Stripe, PayPal).
- **`authService.js`** → Generación y verificación de tokens JWT.

📂 `middlewares/` → Funciones intermedias para validaciones y autenticación

- **`authMiddleware.js`** → Protección de rutas con JWT.

- **`errorHandler.js`** → Manejo global de errores.
- **`validateInput.js`** → Validación de datos con `express-validator`.

📂 `config/` → Configuración del proyecto

- **`db.js`** → Conexión a la base de datos (MongoDB, PostgreSQL, MySQL).
- **`serverConfig.js`** → Configuración de puertos, CORS y otros ajustes.

📂 `utils/` → Funciones auxiliares y herramientas reutilizables

- **`generateToken.js`** → Función para generar tokens JWT.
- **`hashPassword.js`** → Encriptación de contraseñas con bcrypt.
- **`logger.js`** → Sistema de logs con `winston` o `morgan`.

📄 `index.js` (fuera de subcarpetas)

El punto de entrada del servidor:

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const db = require('./config/db');
const userRoutes = require('./routes/userRoutes');

const app = express();
app.use(express.json());
app.use(cors());

// Rutas
app.use('/api/users', userRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Servidor corriendo en el puerto
${PORT}`));
```

Resumen de estructura

```
/src
|── /controllers
|    ├── userController.js
|    ├── productController.js
|    ├── orderController.js
|
|── /models
|    ├── User.js
|    ├── Product.js
|    ├── Order.js
|
|── /routes
|    ├── userRoutes.js
|    ├── productRoutes.js
|    ├── orderRoutes.js
|
|── /services
|    ├── emailService.js
|    ├── paymentService.js
|    ├── authService.js
|
|── /middlewares
|    ├── authMiddleware.js
|    ├── errorHandler.js
|    ├── validateInput.js
|
|── /config
|    ├── db.js
|    ├── serverConfig.js
|
|── /utils
|    ├── generateToken.js
|    ├── hashPassword.js
|    ├── logger.js
|
|── index.js
```