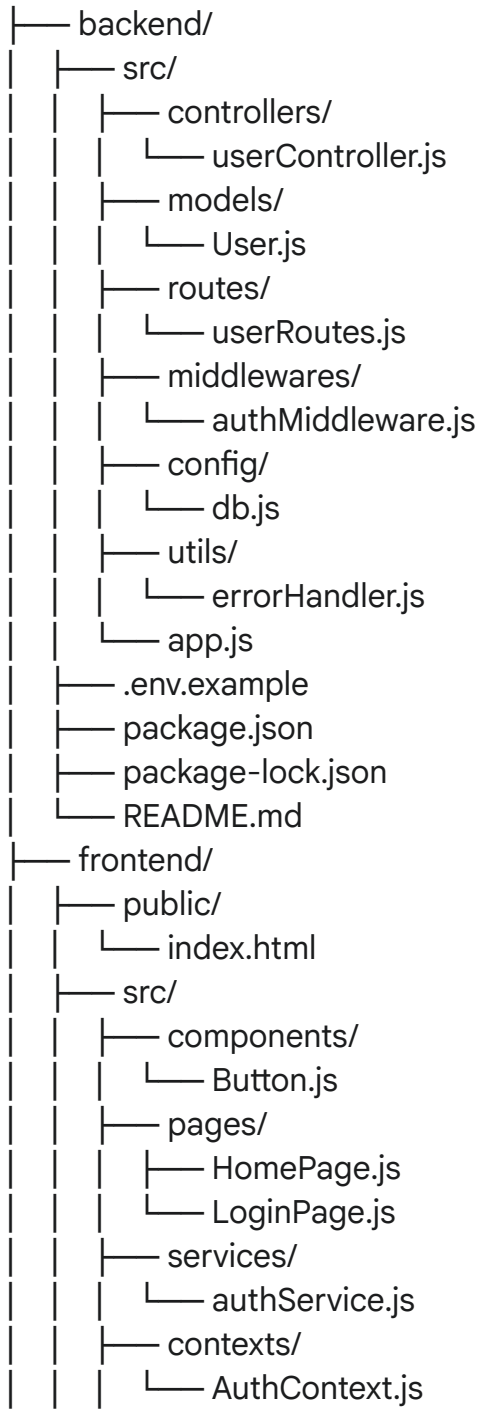
Estructura del Repositorio Full-Stack

Esta estructura está diseñada para separar lógicamente las responsabilidades del frontend y el backend, lo que mejora la claridad, la mantenibilidad y la escalabilidad del proyecto.



```
│   │   ├── hooks/
│   │   │   └── useAuth.js
│   │   ├── assets/
│   │   │   └── images/
│   │   ├── styles/
│   │   │   └── main.css
│   │   └── App.js
│   ├── .env.example
│   ├── package.json
│   ├── package-lock.json
│   └── README.md
├── .gitignore
├── README.md
├── package.json (opcional, para scripts a nivel de root)
└── docker-compose.yml (opcional, para contenerización)
```

Descripción Detallada de las Carpetas

backend/

Este directorio encapsula toda la lógica del servidor, la API RESTful y la interacción con la base de datos.

- **src/**: Contiene el código fuente principal del backend.
  - **controllers/**: Manejan la lógica de negocio y procesan las solicitudes HTTP. Son responsables de interactuar con los modelos y enviar las respuestas adecuadas. Por ejemplo, userController.js contendría funciones para el registro, login, obtención de perfiles de usuario, etc.
  - **models/**: Definen la estructura de los datos y la lógica de interacción con la base de datos (si usas una ORM/ODM como Mongoose para MongoDB o Sequelize para SQL). User.js sería el esquema o modelo para la entidad de usuario.
  - **routes/**: Definen los endpoints de la API y mapean las URL a las funciones de los controladores. userRoutes.js podría manejar rutas como /api/users, /api/users/:id, /api/auth/register, etc.
  - **middlewares/**: Funciones que se ejecutan en la cadena de solicitud/respuesta de Express. Son ideales para autenticación (authMiddleware.js), autorización, validación de datos, logging, manejo de errores, etc.
  - **config/**: Archivos de configuración específicos del backend, como la cadena de conexión a la base de datos (db.js), configuraciones de seguridad, etc.
  - **utils/**: Funciones de utilidad y helpers reutilizables que no encajan directamente en las categorías anteriores. Por ejemplo, un manejador de errores centralizado (errorHandler.js), funciones para generar tokens, etc.
  - **app.js** (o server.js): El archivo principal que inicializa la aplicación Express, configura los middlewares globales, carga las rutas y arranca el servidor.
- **.env.example**: Un archivo de plantilla que lista las variables de entorno necesarias para el backend (ej. PORT, DB_URI, JWT_SECRET). Los valores sensibles se guardan en un archivo .env real que no se versiona.
- **package.json**: Define las dependencias de Node.js del backend y los scripts para ejecutar, probar y construir la aplicación del servidor.
- **package-lock.json**: Registra las versiones exactas de las dependencias para asegurar instalaciones consistentes.
- **README.md**: Proporciona instrucciones detalladas sobre cómo configurar, instalar dependencias, ejecutar y probar el backend.

frontend/

Este directorio contiene toda la aplicación de React, incluyendo componentes, páginas, estilos y lógica de interacción con la API.

- **public/**: Contiene archivos estáticos que se sirven directamente, como index.html (el punto de entrada de la aplicación React), el favicon y otros assets públicos.
- **src/**: Contiene el código fuente principal de la aplicación React.
  - **components/**: Pequeños componentes reutilizables y sin estado (o con estado mínimo) que se utilizan en múltiples páginas. Ejemplos: Button.js, Navbar.js, Card.js, Modal.js.
  - **pages/**: Componentes de React que representan vistas completas o "páginas" de la aplicación. Aquí se orquesta la composición de varios componentes para formar una vista completa. Ejemplos: HomePage.js, LoginPage.js, DashboardPage.js.
  - **services/**: Módulos que contienen funciones para realizar llamadas HTTP a la API del backend. Esto centraliza la lógica de las solicitudes y respuestas de la API. Ejemplos: authService.js (para login/registro), userService.js (para operaciones CRUD de usuarios).
  - **contexts/**: Si utilizas la API de Context de React para la gestión de estados globales que necesitan ser accesibles por muchos componentes sin pasar props manualmente. Ejemplo: AuthContext.js para el estado de autenticación del usuario.
  - **hooks/**: Custom Hooks de React para encapsular lógica reutilizable y con estado. Ayudan a abstraer la lógica compleja de los componentes. Ejemplo: useAuth.js para manejar el estado de autenticación.
  - **assets/**: Archivos estáticos como imágenes, iconos, fuentes, videos, etc., que son utilizados por el frontend.
  - **styles/**: Archivos CSS, SASS, LESS, o módulos CSS para estilizar la aplicación. Puedes organizar por componentes, páginas o globales.
  - **App.js**: El componente raíz de tu aplicación React, donde generalmente se configura el enrutamiento principal (ej. con react-router-dom) y se establecen los layouts globales.
- **.env.example**: Archivo de plantilla para las variables de entorno del frontend (ej. REACT_APP_API_URL para la URL del backend).
- **package.json**: Define las dependencias de React del frontend y los scripts para iniciar el servidor de desarrollo, construir la aplicación para producción, etc.
- **package-lock.json**: Asegura la consistencia de las dependencias del frontend.
- **README.md**: Instrucciones específicas sobre cómo configurar, ejecutar y desplegar la aplicación React.

Archivos en la Raíz del Repositorio

- **.gitignore**: Un archivo crucial que especifica qué archivos y directorios Git debe ignorar al versionar el proyecto (ej. node_modules/, .env, archivos de compilación).
- **README.md**: Un README general para todo el proyecto. Debería incluir una descripción del proyecto, tecnologías utilizadas, requisitos previos, y cómo configurar y ejecutar tanto el frontend como el backend.
- **package.json (opcional)**: Puedes tener un package.json en la raíz para definir scripts "monorepo" que te permitan ejecutar comandos en ambos subproyectos (ej. npm run dev podría iniciar tanto el servidor de Express como el servidor de desarrollo de React simultáneamente). Esto es útil para el desarrollo local.
- **docker-compose.yml (opcional)**: Si utilizas Docker para la contenerización, este archivo te permitiría definir y ejecutar servicios multi-contenedor (backend, frontend, base de datos) de forma orquestada.

Esta estructura proporciona una base sólida para la mayoría de los proyectos full-stack. A medida que tu aplicación crezca en complejidad, podrías considerar patrones de diseño más avanzados o herramientas de monorepo si la necesidad lo justifica.