**Nodejs, PostgreSQL & Typescript, REST API CRUD**

Pasos para Crear este Proyecto

M3-L1

1. Crear la carpeta del proyecto
2. Crear un README.md en la raíz del proyecto
3. Escribir las userstories
4. Crear el diagrama E/R, guardarlo como imagen y agregarlo al README.md

M3-L2

1. Crear una carpeta back y una front donde se guardarán todos los archivos del proyecto como dos aplicaciones independientes
2. Crear un archivo .gitignore que guarda carpetas y archivos que no se subirán a github  
    node\_modules/ para ignorar todas las carpetas node\_modules sin importar donde se encuentre  
    .env para ignorar todos los archivos que terminen en .env sin importar su nombre  
    dist/ para ignorar todas las carpetas dist sin importar donde se encuentre

BACKEND

1. Crear una carpeta src
2. Crear el archivo .env para guardar todas las variables de entorno del backend
3. Crear el archivo index.ts que va a contener la conexión a express  
   Por ahora contiene un console.log(“Hola Mundo!!! ”)
4. Ir a la carpeta back en la consola usando cd back
5. Incializar el back con npm init para crear el package.json
6. Instalar TS con npm install typescript -D o npm install --save-dev typescript las dos opciones crean dependencias de desarrollo en el package.json
7. Crear el archivo tsconfig.json con npx tsc --init
8. Configurar tsconfig.json

{

"compilerOptions": {

/\* Visit https://aka.ms/tsconfig to read more about this file \*/

/\* Language and Environment \*/

"target": "ES6" /\* Set the JavaScript language version for emitted JavaScript and include compatible library declarations. \*/,

/\* Modules \*/

"module": "commonjs" /\* Specify what module code is generated. \*/,

"rootDir": "./src" /\* Specify the root folder within your source files. \*/,

/\* Emit \*/

"outDir": "./dist" /\* Specify an output folder for all emitted files. \*/,

"removeComments": true,                           /\* Disable emitting comments. \*/

/\* Type Checking \*/

"strict": true /\* Enable all strict type-checking options. \*/,

},

"include": ["./src/\*\*/\*.ts"],

"exclude": ["node\_modules", ".env"]

}

1. Agregar al package.json  
   "scripts": {

"build": "tsc" para compilar ts a js

},

1. Agregar al package.json otro script

"start": "node ./dist/index.js" para ejecutar el index.js

1. Agregar al package.json otro script  
   "dev": "tsc && node ./dist/index.js" para compilar y ejecutar con un solo comando

M3-L3

1. Instalar dependencias

npm install --save-dev @types/express @types/node nodemon ts-node

npm install express morgan dotenv cors  
Para buscar documentación escribir npm librería en el buscador y nos lleva a la documentación correspondiente

1. Crear archivo nodemon.json

{

"watch": ["src"],

"ext": "ts",

"exec": "ts-node ./src/index.ts"

}

1. Ajustar package.json  
   "scripts": {

"build": "tsc",

"start": "node ./dist/index.js",

"build:start": "tsc && node ./dist/index.js",

"dev": "nodemon"

},

1. Crear las carpetas necesarias para el backend del proyecto  
   config, controllers, dtos, entities/schemas, interfaces, middlwares, routes, services
2. Agregar las variables de entorno en el archivo .env  
   PORT=3000
3. Crear un archivo de configuración  
   envs.ts que contiene la llamada a dotenv para exportar las variabes de entorno en cualquier lugar donde las necesite  
   import "dotenv/config";

export const PORT = process.env.PORT;

1. Crear archive server.ts  
   import cors from "cors";

import express from "express";

import morgan from "morgan";  
const server = express();

server.use(cors()); // desde que métodos voy a dejar que se ejecuten, desde qué url, qué header, así es para todo tipo de peticiones

server.use(morgan("dev")); // dev = desarrollo

server.use(express.json()); // para que el servidor entienda los json que le llegan por body

server.get("/", (req, res) => {

res.send("Hola Mundo!")

})

export default server;

Instalar por consola  
npm install --save-dev @types/cors @types/Morgan

1. Llamar al servidor desde index.ts  
   import { PORT } from "./config/envs";

import server from "./server";

server.listen(PORT, () => {

console.log(`Server listening on http://localhost:${PORT}`);

});

1. Crear los controladores usersController, appointmentsController
2. Crear los routers para users y appointments y el router principal o indexRouter.ts
3. Actualizar el archivo server.ts

M3-L4

1. Crear las interfaces IUser, IAppointment, ICredential
2. Crear los servicios usersService, appointmentsService, credentialsService
3. Crear los dtos userDto, appointmentDto, credentialDto

M3-L5

TRABAJANDO CON UNA BASE DE DATOS EN POSTGRESQL

1. Abrir SQL Shell de Postgres  
   Server [localhost]: localhost

Database [postgres]: postgres

Port [5432]: 5432

Username [postgres]: postgres

Contraseña para usuario postgres: Eli09Ate89

1. Crear la base de datos (en el nombre de la DB no se pueden usar mayúsculas, camel case, ni guión medio)

CREATE DATABASE veterinaria (en el ejercicio puso project3\_ft53 como nombre)

1. Conectar a la base de datos con \c y ver si hay tablas \dt
2. Instalar typeORM y sus dependencias  
   Página de la documentación: <https://typeorm.io/>

npm install typeorm --save

npm install reflect-metadata --save

1. Hacer el siguiente import en algún lugar global del proyecto (en el index.ts principal del backend):

import "reflect-metadata"

1. Instalar los types de node:

@types/node

1. Controladores de postgres:

npm install pg --save

1. Habilitar en tsconfig.json:

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"lib": ["ES6"],

1. Si vamos a compartir el proyecto debemos crear una archivo  
   .example.env

Para que guarde un modelo de las variables de entorno sin los datos verdades, para que sean completados por la persona que necesita ejecutar el proyecto

1. Crear archivo data-source.ts con la siguiente Configuración

import { DataSource } from "typeorm";

export const AppDataSource = new DataSource({

type: "postgres",

host: "localhost",

port: 5432,

username: "postgres",

password: "Eli09Ate89",

database: "veterinaria",

synchronize: true,

logging: true,

entities: [],

subscribers: [],

migrations: [],

});

HASTA AQUÍ DONE, SOLO FALTA AGREGAR LA ENTITIES A DATA-SOURCE.TS

1. Hacerla conexión a la base de datos en el archivo index.ts

AppDataSource.initialize()

.then(() => {

console.log(`Database connected on port ${DB\_PORT}`);

server.listen(PORT, () => {

console.log(`Server listening on http://localhost:${PORT}`);

});

})

.catch((error) => console.log(error));

1. Crear las entidades: User, Appointment, Credential usando los decoradores @entity() y @column() así como las relaciones entre las tablas

Al crear las columnas puedo usar   
! es el operador de assertion de no nulo de TypeScript, que permite decirle al compilador que un valor no será null ni undefined en tiempo de ejecución. Debe usarse con cuidado y solo cuando estás seguro de que el valor nunca será nulo o indefinido. En el contexto de TypeORM, es útil para las columnas con NOT NULL. No usarlo sin razón, es una mala práctica, ya que evitas que TypeScript te de las alertas necesarias en caso de errores de este tipo.

Tener en cuenta que:   
- Entre User y Credential hay una relación de 1:1 (en este caso la relación va en la tabla principal User)

  @OneToOne(() => Credential)

  @JoinColumn({ name: "credentialId" })

  credential!: Credential;

- Entre User y Appointment hay una relación de 1:N

Esta relación va en User

  @OneToMany(() => Appointment, (appointment) => appointment.user)

  appointments!: Appointment[];

- Entre Appointment y User hay una relación de N:1

Esta relación va en Appointment

@ManyToOne(() => User, (user) => user.appointments)

  @JoinColumn({ name: "userId" })

  user!: User;

No puede existir un turno sin un usuario que lo pidió appointment tiene que tener el userId

1. Ajustar usersService y appointmentsService para que tomen los datos de la base de datos y no de un array
2. Crear en Thunder Client de forma organizada las peticiones que vamos a ir haciendo
3. Activar y poner en false la siguiente propiedad de tsconfig.json "strictPropertyInitialization": false,
4. Configuraciones de los servicios, aquí es donde voy a establecer la conexión con la base de datos.

export const getUsersService = async () => {

const users = await AppDataSource.getRepository(User).find();

return users;

};

export const createUserService = async (userData: UserDto) => {

const user = await userModel.create(userData); // crea el registro

const result = userModel.save(user); // guarda el registro

return user;

};

export const getUserByIdService = async (id: number) => {

const user = await userModel.findOneBy({ id });

return user;

};

1. Crear todos los archivo necesarios para vehicles
2. Configurar dropschema en true para vaciar las tablas de la base de datos. Es importante al comenzar un proyecto para eliminar datos si necesitamos hacerlo para cambiar los datos
3. Crear la relación entre las tablas users y vehicles agregando a la entity User

@OneToOne(() => Vehicle)

@JoinColumn()

vehicle: Vehicle;

1. Ajustar el Servicio para establecer la relación y hacer la petición de envío a la base de datos y agregar el find relations para que se muestren los datos relacionados a los users
2. Vamos a trabajar relaciones de uno a varios, para hacerlo más real

**1. Configuración del proyecto**

Antes de escribir código, necesitamos configurar el entorno.

1. **Inicializar un proyecto Node.js con TypeScript**

mkdir mi-proyecto && cd mi-proyecto npm init -y npm install typescript ts-node nodemon @types/node --save-dev

1. **Instalar las dependencias principales**

npm install express typeorm pg reflect-metadata class-validator class-transformer npm install @types/express --save-dev

1. **Configurar TypeScript** (tsconfig.json)

json

{ "compilerOptions": { "outDir": "./dist",

"module": "CommonJS",

"target": "ES6",

"strict": true,

"rootDir": "./src" } }

**🔹 2. Estructura del Proyecto**

📁 **mi-proyecto**  
│── 📁 **src**  
│ ├── 📁 **config** → Configuración de la base de datos  
│ ├── 📁 **entities** → Definición de tablas (TypeORM)  
│ ├── 📁 **dtos** → Validaciones de datos  
│ ├── 📁 **repositories** → Métodos de acceso a la base de datos  
│ ├── 📁 **services** → Lógica de negocio  
│ ├── 📁 **controllers** → Recibe y responde a las peticiones HTTP  
│ ├── 📁 **routes** → Define las rutas de la API  
│ ├── 📁 **middleware** → Funciones intermedias  
│ ├── 📄 **server.ts** → Punto de entrada  
│── 📄 **.env** → Variables de entorno  
│── 📄 **ormconfig.ts** → Configuración de TypeORM  
│── 📄 **package.json**

**🔹 3. Configurar la Base de Datos con TypeORM**

📄 **src/config/data-source.ts**

typescript

CopiarEditar

import "reflect-metadata";

import { DataSource } from "typeorm";

import { User } from "../entities/User"; export const AppDataSource = new DataSource({ t

ype: "postgres",

host: "localhost",

port: 5432,

username: "postgres",

password: "123456",

database: "mi\_base\_de\_datos",

synchronize: true, entities: [User], });

**🔹 4. Definir las Entidades (Tablas de la BD)**

📄 **src/entities/User.ts**

typescript

import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

@Entity() export class User {

@PrimaryGeneratedColumn()

id: number;

@Column()

name: string;

@Column({ unique: true })

email: string;

@Column()

password: string; }

**🔹 5. Crear DTOs (Data Transfer Objects)**

📄 **src/dtos/CreateUserDto.ts**

typescript

import { IsEmail, IsString, MinLength } from "class-validator";

export class CreateUserDto {

@IsString()

name: string;

@IsEmail()

email: string;

@IsString() @MinLength(6)

password: string; }

**🔹 6. Crear un Repositorio**

📄 **src/repositories/UserRepository.ts**

typescript

import { AppDataSource } from "../config/data-source";

import { User } from "../entities/User";

export const userRepository = AppDataSource.getRepository(User);

**🔹 7. Crear un Servicio**

📄 **src/services/UserService.ts**

typescript

import { userRepository } from "../repositories/UserRepository";

import { User } from "../entities/User";

import { CreateUserDto } from "../dtos/CreateUserDto";

import { validate } from "class-validator";

export class UserService { async createUser(data: CreateUserDto): Promise<User | string> {

const errors = await validate(data);

if (errors.length > 0) return "Datos no válidos";

const newUser = userRepository.create(data);

return await userRepository.save(newUser);

}

async getAllUsers(): Promise<User[]> {

return await userRepository.find();

} }

**🔹 8. Crear un Controlador**

📄 **src/controllers/UserController.ts**

typescript

import { Request, Response } from "express";

import { UserService } from "../services/UserService";

const userService = new UserService();

export class UserController { static async createUser(req: Request, res: Response) { const result = await userService.createUser(req.body);

res.json(result); } static async getUsers(req: Request, res: Response) {

const users = await userService.getAllUsers(); res.json(users);

} }

**🔹 9. Definir las Rutas**

📄 **src/routes/UserRoutes.ts**

typescript

CopiarEditar

import { Router } from "express"; import { UserController } from "../controllers/UserController"; const router = Router(); router.post("/users", UserController.createUser); router.get("/users", UserController.getUsers); export default router;

**🔹 10. Configurar el Servidor**

📄 **src/server.ts**

typescript

CopiarEditar

import "reflect-metadata"; import express from "express"; import { AppDataSource } from "./config/data-source"; import userRoutes from "./routes/UserRoutes"; const app = express(); app.use(express.json()); app.use("/api", userRoutes); AppDataSource.initialize() .then(() => { console.log("📦 Base de datos conectada"); app.listen(3000, () => console.log("🚀 Servidor en http://localhost:3000")); }) .catch((error) => console.log(error));

**🔹 11. Arrancar el Proyecto**

1. **Iniciar la base de datos** (Asegúrate de que PostgreSQL esté ejecutándose).
2. **Levantar el servidor**

sh

CopiarEditar

npm run dev

**🔹 12. Probar las Rutas**

* Crear un usuario:

http

CopiarEditar

POST http://localhost:3000/api/users Content-Type: application/json { "name": "Juan", "email": "juan@example.com", "password": "123456" }

* Obtener todos los usuarios:

http

CopiarEditar

GET http://localhost:3000/api/users

**🔹 Conclusión**

Este es un esquema **básico pero funcional** de cómo desarrollar un sitio web con **TypeScript, Express, TypeORM y PostgreSQL**. 🚀

Si tienes dudas, dime y te ayudo paso a paso. 😃

**Tú dijiste:**

que debe haber dentro de cada carpeta y en que momento creo cada cosa, supongamos que estoy creando un sitio para una veterinaria, detalla cada archivo que debería crear, cada tabla, todo lo que necesito tenr para que funcione correctamente

**ChatGPT Plus**

ChatGPT

Aquí tienes una **guía detallada** sobre cómo desarrollar un sitio web para una veterinaria usando **TypeScript, Node.js, Express, TypeORM y PostgreSQL**. Te explicaré **qué debes crear, cuándo hacerlo y qué debe contener cada archivo**.

**1️⃣ Configuración del Proyecto**

Antes de escribir código, configuramos el entorno.

**📌 Pasos Iniciales**

1. **Crear la carpeta del proyecto y entrar en ella**

sh

CopiarEditar

mkdir veterinaria && cd veterinaria

1. **Inicializar un proyecto Node.js con TypeScript**

sh

CopiarEditar

npm init -y

1. **Instalar TypeScript y herramientas de desarrollo**

sh

CopiarEditar

npm install typescript ts-node nodemon @types/node --save-dev

1. **Instalar Express, TypeORM y PostgreSQL**

sh

CopiarEditar

npm install express typeorm pg reflect-metadata class-validator class-transformer npm install @types/express --save-dev

1. **Configurar TypeScript** (tsconfig.json)

json

CopiarEditar

{ "compilerOptions": { "outDir": "./dist", "module": "CommonJS", "target": "ES6", "strict": true, "rootDir": "./src" } }

**2️⃣ Estructura del Proyecto**

Este es el esquema de carpetas y archivos que vamos a crear:

📁 **veterinaria**  
│── 📁 **src**  
│ ├── 📁 **config** → Configuración de la base de datos  
│ ├── 📁 **entities** → Definición de tablas (TypeORM)  
│ ├── 📁 **dtos** → Validaciones de datos  
│ ├── 📁 **repositories** → Métodos de acceso a la base de datos  
│ ├── 📁 **services** → Lógica de negocio  
│ ├── 📁 **controllers** → Recibe y responde a las peticiones HTTP  
│ ├── 📁 **routes** → Define las rutas de la API  
│ ├── 📁 **middleware** → Funciones intermedias  
│ ├── 📄 **server.ts** → Punto de entrada  
│── 📄 **.env** → Variables de entorno  
│── 📄 **ormconfig.ts** → Configuración de TypeORM  
│── 📄 **package.json**

**3️⃣ Configurar la Base de Datos**

Creamos la configuración para conectarnos a PostgreSQL.

📄 **src/config/data-source.ts**

typescript

CopiarEditar

import "reflect-metadata"; import { DataSource } from "typeorm"; import { Cliente } from "../entities/Cliente"; import { Mascota } from "../entities/Mascota"; import { Cita } from "../entities/Cita"; export const AppDataSource = new DataSource({ type: "postgres", host: "localhost", port: 5432, username: "postgres", password: "123456", database: "veterinaria", synchronize: true, entities: [Cliente, Mascota, Cita], });

**4️⃣ Crear las Tablas (Entidades)**

Definimos las tablas con **TypeORM**.

**📌 Tabla Clientes**

📄 **src/entities/Cliente.ts**

typescript

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm"; import { Mascota } from "./Mascota"; @Entity() export class Cliente { @PrimaryGeneratedColumn() id: number; @Column() nombre: string; @Column({ unique: true }) email: string; @Column() telefono: string; @OneToMany(() => Mascota, (mascota) => mascota.cliente) mascotas: Mascota[]; }

**📌 Tabla Mascotas**

📄 **src/entities/Mascota.ts**

typescript

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm"; import { Cliente } from "./Cliente"; @Entity() export class Mascota { @PrimaryGeneratedColumn() id: number; @Column() nombre: string; @Column() especie: string; @Column() raza: string; @Column() edad: number; @ManyToOne(() => Cliente, (cliente) => cliente.mascotas) cliente: Cliente; }

**📌 Tabla Citas**

📄 **src/entities/Cita.ts**

typescript

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm"; import { Mascota } from "./Mascota"; @Entity() export class Cita { @PrimaryGeneratedColumn() id: number; @Column() fecha: Date; @Column() motivo: string; @ManyToOne(() => Mascota, (mascota) => mascota.id) mascota: Mascota; }

**5️⃣ Crear los DTOs**

**📄 src/dtos/CreateClienteDto.ts**

typescript

CopiarEditar

import { IsEmail, IsString, IsNotEmpty } from "class-validator"; export class CreateClienteDto { @IsString() @IsNotEmpty() nombre: string; @IsEmail() email: string; @IsString() telefono: string; }

**6️⃣ Crear Repositorios**

📄 **src/repositories/ClienteRepository.ts**

typescript

CopiarEditar

import { AppDataSource } from "../config/data-source"; import { Cliente } from "../entities/Cliente"; export const clienteRepository = AppDataSource.getRepository(Cliente);

**7️⃣ Crear Servicios**

📄 **src/services/ClienteService.ts**

typescript

CopiarEditar

import { clienteRepository } from "../repositories/ClienteRepository"; import { CreateClienteDto } from "../dtos/CreateClienteDto"; import { Cliente } from "../entities/Cliente"; export class ClienteService { async createCliente(data: CreateClienteDto): Promise<Cliente> { const newCliente = clienteRepository.create(data); return await clienteRepository.save(newCliente); } async getAllClientes(): Promise<Cliente[]> { return await clienteRepository.find(); } }

**8️⃣ Crear Controladores**

📄 **src/controllers/ClienteController.ts**

typescript

CopiarEditar

import { Request, Response } from "express"; import { ClienteService } from "../services/ClienteService"; const clienteService = new ClienteService(); export class ClienteController { static async createCliente(req: Request, res: Response) { const result = await clienteService.createCliente(req.body); res.json(result); } static async getClientes(req: Request, res: Response) { const clientes = await clienteService.getAllClientes(); res.json(clientes); } }

**9️⃣ Definir Rutas**

📄 **src/routes/ClienteRoutes.ts**

typescript

CopiarEditar

import { Router } from "express"; import { ClienteController } from "../controllers/ClienteController"; const router = Router(); router.post("/clientes", ClienteController.createCliente); router.get("/clientes", ClienteController.getClientes); export default router;

**🔟 Configurar el Servidor**

📄 **src/server.ts**

typescript

CopiarEditar

import "reflect-metadata"; import express from "express"; import { AppDataSource } from "./config/data-source"; import clienteRoutes from "./routes/ClienteRoutes"; const app = express(); app.use(express.json()); app.use("/api", clienteRoutes); AppDataSource.initialize() .then(() => { console.log("📦 Base de datos conectada"); app.listen(3000, () => console.log("🚀 Servidor en http://localhost:3000")); }) .catch((error) => console.log(error));

**🚀 Arrancar el Proyecto**

sh

CopiarEditar

npm run dev

**📌 Probar con Thunder Client**

* Crear Cliente:

http

CopiarEditar

POST http://localhost:3000/api/clientes Content-Type: application/json { "nombre": "Juan Pérez", "email": "juan@example.com", "telefono": "123456789" }

Este es el esquema detallado para un **sitio web de veterinaria** con **TypeScript, Express, TypeORM y PostgreSQL**. ¿Te gustaría agregar más funcionalidades? 😊

**Tú dijiste:**

Genial, mil gracias por tu ayuda

**ChatGPT Plus**

ChatGPT

¡De nada! 😃 Me alegra saber que te fue útil. Cualquier duda o si quieres agregar más funcionalidades, aquí estoy para ayudarte. 🚀 ¡Éxito con tu proyecto de la veterinaria! 🐶🐱

**Tú dijiste:**

Siguiendo la misma pauta del comienzo ayúdame a ver por qué en el siguiente código no se muesran los appoinments junto con los users, dime que más necesitas para responder si es que no tienes suficiente información con lo que voy a compartirte export const getAllUsersService = async (): Promise<User[]> => {   const allUsers: User[] = await userRepository.find();   relations: {     apointments: true;   } // otra forma de hacer la relación   return allUsers; };

**ChatGPT Plus**

ChatGPT

El problema con tu código es que la forma en que estás intentando incluir la relación **appointments** no es correcta. La opción relations debe pasarse dentro de find() como un objeto de opciones. Ahora mismo, la línea:

typescript

CopiarEditar

relations: { apointments: true; }

está fuera del find() y no tiene ningún efecto.

**✅ Solución**

Corrige el código incluyendo la opción relations correctamente:

typescript

CopiarEditar

export const getAllUsersService = async (): Promise<User[]> => { const allUsers: User[] = await userRepository.find({ relations: { appointments: true, // Asegúrate de que el nombre sea correcto }, }); return allUsers; };

**🔍 Posibles Errores Adicionales**

Si **aún no ves las citas (**appointments**)** junto con los usuarios, revisa lo siguiente:

1. **Verifica el nombre de la relación en la entidad**User  
   En tu **entidad**User, la relación con Appointment (o como la hayas llamado) debe estar bien definida.

📄 **src/entities/User.ts**

typescript

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm"; import { Appointment } from "./Appointment"; // Asegúrate de importar correctamente @Entity() export class User { @PrimaryGeneratedColumn() id: number; @Column() name: string; @OneToMany(() => Appointment, (appointment) => appointment.user) appointments: Appointment[]; // Asegúrate de que el nombre coincide con "relations" }

1. **Verifica la relación en**Appointment.ts  
   📄 **src/entities/Appointment.ts**

typescript

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm"; import { User } from "./User"; @Entity() export class Appointment { @PrimaryGeneratedColumn() id: number; @Column() date: Date; @ManyToOne(() => User, (user) => user.appointments) user: User; }

1. **Verifica que la base de datos tenga datos relacionados**  
   Ejecuta una consulta en PostgreSQL para asegurarte de que los appointments tienen un userId válido.

sql

CopiarEditar

SELECT \* FROM appointments WHERE userId IS NOT NULL;

Si después de esto sigues teniendo problemas, dime si TypeORM está cargando correctamente la relación o si hay algún error en la consola. 🚀

**Tú dijiste:**

Listo, muchas gracias

**ChatGPT Plus**

ChatGPT

¡De nada! 😃 Si tienes más dudas, aquí estoy para ayudarte. ¡Éxito con tu proyecto! 🚀🐶🐱

**Tú dijiste:**

Una pregunta por qué cuando doy enter en el cuadro de conversación tuyo no cambia de línea, estoy trabajando con google Chrome

**ChatGPT Plus**

ChatGPT

En el cuadro de conversación de esta plataforma, al presionar **Enter** se envía el mensaje en lugar de crear una nueva línea. Para crear una nueva línea sin enviar el mensaje, puedes usar la combinación de teclas:

**Shift + Enter**

Esto funciona tanto en **Google Chrome** como en otros navegadores. 😊

**2/2**

**Tú dijiste:**

Estoy usando shift+enter yno cambia de linea

**ChatGPT Plus**

ChatGPT

Parece un problema del navegador o de la configuración del campo de texto. Prueba lo siguiente en **Google Chrome**:

**Opciones para solucionar el problema**

1️⃣ **Prueba con otro navegador**

* Intenta abrir la conversación en **Firefox** o **Edge** para ver si el problema persiste.

2️⃣ **Revisa extensiones**

* Algunas extensiones pueden interferir con el campo de texto. Prueba desactivarlas temporalmente:
  1. Abre Chrome y ve a chrome://extensions/
  2. Desactiva las extensiones una por una
  3. Recarga la página y prueba de nuevo

3️⃣ **Usa otro método para hacer salto de línea**

* Prueba con:
  + Ctrl + Enter
  + Alt + Enter
  + Shift + Espacio antes de Enter

4️⃣ **Prueba en una ventana de incógnito**

* Abre una nueva ventana en **modo incógnito** (Ctrl + Shift + N) y accede a la página.

5️⃣ **Limpia caché y cookies**

* Ve a chrome://settings/clearBrowserData, selecciona **Cookies y datos de otros sitios** y prueba de nuevo.

Si nada de esto funciona, dime si notas algún otro comportamiento extraño en la página o en otros sitios web. 🚀

**Tú dijiste:**

me pasa lo mismo cuando hago búsquedas en el buscador de google cuando doy enter en lugar de ir a traer la respuesta cambia de linea

**ChatGPT Plus**

ChatGPT

Si **Enter** no está funcionando como debería en Google y en otros sitios, puede ser un problema del teclado, la configuración del navegador o algún software en tu PC.

**🔍 Posibles causas y soluciones**

**1️⃣ Prueba en otro navegador**

* Si el problema ocurre **solo en Chrome**, es probable que sea una configuración o una extensión.
* Si también pasa en **Firefox, Edge u otro**, el problema podría estar en el teclado o en la configuración del sistema.

**2️⃣ Revisa tu teclado**

* ¿Estás usando un teclado externo? **Prueba con otro teclado o cambia de puerto USB**.
* Usa un **teclado en pantalla** para probar si el problema es físico:
  + Presiona Windows + R, escribe osk y presiona **Enter**.
  + Usa el teclado en pantalla y prueba si el problema sigue.

**3️⃣ Revisa la configuración del idioma y distribución del teclado**

* Ve a **Configuración > Hora e idioma > Idioma y región** en Windows.
* Asegúrate de que el **idioma del teclado** sea **Español (Latinoamérica) o Español (España)** según el que uses.

**4️⃣ Desactiva extensiones en Chrome**

* Abre chrome://extensions/ y desactiva todas las extensiones.
* Recarga la página y prueba si **Enter** vuelve a funcionar normalmente.

**5️⃣ Restablece la configuración de Chrome**

Si el problema es solo en Chrome:

* Ve a chrome://settings/reset
* Haz clic en **"Restablecer configuración a los valores predeterminados"**

**6️⃣ Verifica si un programa en segundo plano está interfiriendo**

* Abre el **Administrador de tareas** (Ctrl + Shift + Esc) y revisa si hay algún programa sospechoso.
* Si tienes un software de personalización de teclado o macros, intenta cerrarlo y prueba de nuevo.

Si el problema sigue, dime **qué pasa cuando presionas Enter en un bloc de notas o en otro programa** para ver si afecta solo al navegador o a todo el sistema. 🚀

**Tú dijiste:**

No sucede lo mismo con brave y usa el mismo motor que google

**ChatGPT Plus**

ChatGPT

Si en **Brave** funciona bien pero en **Google Chrome** no, el problema seguramente está en la configuración de Chrome o en una extensión. Prueba lo siguiente:

**🔹 1. Desactiva todas las extensiones**

Algunas extensiones pueden modificar el comportamiento del teclado.

🔹 **Prueba esto:**

1. Ve a chrome://extensions/
2. Desactiva **todas** las extensiones
3. Cierra Chrome y vuelve a abrirlo
4. Prueba si **Enter** funciona correctamente

Si funciona bien después de esto, activa las extensiones **una por una** para identificar cuál está causando el problema.

**🔹 2. Restablece la configuración de Chrome**

Si el problema persiste, restablecer la configuración puede ayudar.

🔹 **Pasos:**

1. Ve a chrome://settings/reset
2. Haz clic en **"Restablecer configuración a sus valores predeterminados"**
3. Confirma y reinicia el navegador

⚠️ Esto **no borra tus marcadores ni contraseñas**, pero sí restablece la página de inicio, el motor de búsqueda y desactiva todas las extensiones.

**🔹 3. Verifica si hay una actualización de Chrome**

Si Chrome está desactualizado, pueden ocurrir errores extraños.

🔹 **Cómo actualizar Chrome:**

1. Ve a chrome://settings/help
2. Si hay una actualización disponible, instálala
3. Reinicia el navegador

**🔹 4. Prueba con un nuevo perfil de usuario**

A veces, la configuración del perfil de usuario puede estar dañada.

🔹 **Pasos:**

1. Entra a chrome://settings/people
2. Crea un nuevo perfil de usuario
3. Abre una nueva ventana con ese perfil y prueba si **Enter** funciona bien

Si en el **nuevo perfil** funciona bien, el problema está en tu perfil actual.

**🔹 5. Si nada funciona, reinstala Chrome**

Si ninguna solución anterior funciona:

1. Desinstala Chrome
2. Borra la carpeta de datos (C:\Users\TuUsuario\AppData\Local\Google\Chrome)
3. Descarga e instala Chrome nuevamente desde aquí

Con estas pruebas deberíamos encontrar la causa del problema. 🚀  
Dime si alguna de estas soluciones te funcionó o si necesitas más ayuda. 😊

**Tú dijiste:**

Puedes hacerme una secuencia detallada de como crear un sitio web con typescript, nodejs, expressjs, typeorm, postgresql, que incluya todas las carpetas y archivos que se necesitan.

**ChatGPT Plus**

ChatGPT

Aquí tienes una guía detallada, paso a paso, para crear un sitio web usando **TypeScript, Node.js, Express.js, TypeORM y PostgreSQL**. La estructura del proyecto estará organizada en capas (rutas, controladores, servicios, repositorios, entidades y DTOs).

**1️⃣ Crear el proyecto y configurar TypeScript**

**1.1 Inicializar el proyecto**

Ejecuta en la terminal:

sh

CopiarEditar

mkdir veterinaria-api cd veterinaria-api npm init -y

**1.2 Instalar dependencias**

Ejecuta:

sh

CopiarEditar

# Dependencias principales npm install express pg typeorm reflect-metadata dotenv # Dependencias de desarrollo npm install --save-dev typescript ts-node @types/express @types/node nodemon

**1.3 Configurar TypeScript**

Crea el archivo tsconfig.json:

json

CopiarEditar

{ "compilerOptions": { "target": "ES6", "module": "CommonJS", "outDir": "./dist", "rootDir": "./src", "strict": true, "esModuleInterop": true } }

**2️⃣ Estructura del Proyecto**

pgsql

CopiarEditar

veterinaria-api/

│── node\_modules/

│── src/

│ ├── config/

│ │ ├── database.ts

│ ├── entities/

│ │ ├── User.ts

│ │ ├── Appointment.ts

│ ├── dtos/

│ │ ├── CreateUserDto.ts

│ │ ├── CreateAppointmentDto.ts

│ ├── repositories/

│ │ ├── UserRepository.ts

│ │ ├── AppointmentRepository.ts

│ ├── services/

│ │ ├── UserService.ts

│ │ ├── AppointmentService.ts

│ ├── controllers/

│ │ ├── UserController.ts

│ │ ├── AppointmentController.ts

│ ├── routes/

│ │ ├── userRoutes.ts

│ │ ├── appointmentRoutes.ts

│ ├── app.ts

│ ├── server.ts

│── .env

│── ormconfig.json

│── package.json

│── tsconfig.json

│── README.md

**3️⃣ Configurar TypeORM con PostgreSQL**

**3.1 Configurar la conexión a la base de datos**

**📌 Crear archivo**.env

env

CopiarEditar

DB\_HOST=localhost DB\_PORT=5432 DB\_USER=postgres DB\_PASSWORD=1234 DB\_NAME=veterinaria

**📌 Configurar TypeORM en**src/config/database.ts

ts

CopiarEditar

import "reflect-metadata"; import { DataSource } from "typeorm"; import { User } from "../entities/User"; import { Appointment } from "../entities/Appointment"; import dotenv from "dotenv"; dotenv.config(); export const AppDataSource = new DataSource({ type: "postgres", host: process.env.DB\_HOST, port: Number(process.env.DB\_PORT), username: process.env.DB\_USER, password: process.env.DB\_PASSWORD, database: process.env.DB\_NAME, synchronize: true, logging: false, entities: [User, Appointment], migrations: [], subscribers: [], }); AppDataSource.initialize() .then(() => console.log("📦 Base de datos conectada")) .catch((error) => console.log("❌ Error al conectar DB:", error));

**4️⃣ Crear Entidades (Tablas de la BD)**

**4.1**src/entities/User.ts

ts

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from "typeorm"; import { Appointment } from "./Appointment"; @Entity() export class User { @PrimaryGeneratedColumn() id: number; @Column() name: string; @Column() email: string; @OneToMany(() => Appointment, (appointment) => appointment.user) appointments: Appointment[]; }

**4.2**src/entities/Appointment.ts

ts

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm"; import { User } from "./User"; @Entity() export class Appointment { @PrimaryGeneratedColumn() id: number; @Column() date: string; @Column() description: string; @ManyToOne(() => User, (user) => user.appointments) user: User; }

**5️⃣ Crear DTOs (Data Transfer Objects)**

**5.1**src/dtos/CreateUserDto.ts

ts

CopiarEditar

export class CreateUserDto { name: string; email: string; }

**5.2**src/dtos/CreateAppointmentDto.ts

ts

CopiarEditar

export class CreateAppointmentDto { date: string; description: string; userId: number; }

**6️⃣ Crear Repositorios**

**6.1**src/repositories/UserRepository.ts

ts

CopiarEditar

import { AppDataSource } from "../config/database"; import { User } from "../entities/User"; export const userRepository = AppDataSource.getRepository(User);

**6.2**src/repositories/AppointmentRepository.ts

ts

CopiarEditar

import { AppDataSource } from "../config/database"; import { Appointment } from "../entities/Appointment"; export const appointmentRepository = AppDataSource.getRepository(Appointment);

**7️⃣ Crear Servicios**

**7.1**src/services/UserService.ts

ts

CopiarEditar

import { userRepository } from "../repositories/UserRepository"; import { CreateUserDto } from "../dtos/CreateUserDto"; import { User } from "../entities/User"; export const getAllUsers = async (): Promise<User[]> => { return await userRepository.find({ relations: ["appointments"] }); }; export const createUser = async (userData: CreateUserDto): Promise<User> => { const user = userRepository.create(userData); return await userRepository.save(user); };

**8️⃣ Crear Controladores**

**8.1**src/controllers/UserController.ts

ts

CopiarEditar

import { Request, Response } from "express"; import { getAllUsers, createUser } from "../services/UserService"; export const getUsers = async (req: Request, res: Response) => { const users = await getAllUsers(); res.json(users); }; export const addUser = async (req: Request, res: Response) => { const user = await createUser(req.body); res.status(201).json(user); };

**9️⃣ Configurar las Rutas**

**9.1**src/routes/userRoutes.ts

ts

CopiarEditar

import { Router } from "express"; import { getUsers, addUser } from "../controllers/UserController"; const router = Router(); router.get("/", getUsers); router.post("/", addUser); export default router;

**🔟 Configurar Express y el Servidor**

**10.1**src/app.ts

ts

CopiarEditar

import express from "express"; import userRoutes from "./routes/userRoutes"; const app = express(); app.use(express.json()); app.use("/users", userRoutes); export default app;

**10.2**src/server.ts

ts

CopiarEditar

import app from "./app"; import { AppDataSource } from "./config/database"; AppDataSource.initialize().then(() => { app.listen(3000, () => console.log("🚀 Servidor corriendo en http://localhost:3000")); });

Con esto ya tienes la estructura completa de un sitio web con **TypeScript, Node.js, Express, TypeORM y PostgreSQL**. 🚀

Si tienes dudas o necesitas ajustes, dime. 😃

**Tú dijiste:**

para que sirven las interfaces y por qué se dejan de usar en el proyecto?

**ChatGPT Plus**

ChatGPT

Las **interfaces en TypeScript** se utilizan para definir la estructura de un objeto sin generar código en tiempo de ejecución. Sirven para garantizar que los objetos sigan una forma específica y ayudan a mantener el código más seguro y tipado.

**🔹 Para qué sirven las interfaces en TypeScript**

1. **Definir la forma de un objeto**

ts

CopiarEditar

interface User { id: number; name: string; email: string; } const user: User = { id: 1, name: "Juan", email: "juan@example.com" };

1. **Enforcing typing en funciones y clases**

ts

CopiarEditar

function printUser(user: User): void { console.log(`Usuario: ${user.name}, Email: ${user.email}`); }

1. **Extender estructuras**

ts

CopiarEditar

interface Admin extends User { role: string; }

**🔹 ¿Por qué se dejan de usar interfaces en el proyecto con TypeORM?**

En proyectos con **TypeORM**, en lugar de interfaces, se utilizan **clases de entidad** (@Entity()), porque TypeORM necesita trabajar con objetos reales en tiempo de ejecución.

**🔹 Diferencias clave entre interfaces y clases en este caso**

| **Característica** | **Interface** | **Clase (**@Entity**)** |
| --- | --- | --- |
| **Solo tipo, sin implementación** | ✅ Sí | ❌ No |
| **Genera código en tiempo de ejecución** | ❌ No | ✅ Sí |
| **Funciona con TypeORM** | ❌ No | ✅ Sí |
| **Permite métodos y lógica** | ❌ No | ✅ Sí |

**Ejemplo de por qué usamos clases en lugar de interfaces**

**👎 Usar interface (no funcionará con TypeORM)**

ts

CopiarEditar

interface User { id: number; name: string; email: string; }

Esto **no funciona** con TypeORM porque las interfaces desaparecen en tiempo de ejecución y no pueden ser interpretadas por TypeORM para crear la tabla en la base de datos.

**✅ Usar clase con TypeORM (funciona correctamente)**

ts

CopiarEditar

import { Entity, PrimaryGeneratedColumn, Column } from "typeorm"; @Entity() export class User { @PrimaryGeneratedColumn() id: number; @Column() name: string; @Column() email: string; }

Aquí **TypeORM sí puede trabajar con la clase** porque es un objeto en tiempo de ejecución.

**🔹 Entonces, ¿cuándo usar interfaces en este proyecto?**

1. **Para definir DTOs** (Data Transfer Objects), donde no necesitamos lógica de base de datos:

ts

CopiarEditar

export interface CreateUserDto { name: string; email: string; }

1. **Para definir estructuras en servicios externos** (como respuestas de APIs).

Pero para trabajar con la base de datos y TypeORM, **usamos clases en lugar de interfaces**.

Si necesitas más detalles o ejemplos, dime. 🚀

FRONTEND

M3-L7

1. Generar el proyecto de frontent utilizando vite  
   npm create vite@latest

Nombre: turnosveterinaria

Select a framework: React

Select a variant: JavaScript

1. Reorganizar las carpetas para que quede la estructura que yo quiero
2. Ejecutar

npm install

1. Ejecutar

npm run dev

1. Abrir app.jsx y eliminar todo lo que no vamos a necesitar en nuestra página principal
2. Abrir main.jsx y eliminar el import del archivo index.css
3. Crear dos carpetas muy importantes, ambas van a tener componentes  
   views: componentes más grandes que representan las distintas vistas de a aplicación  
   components
4. Modularizar las views, separar en carpetas individuales cada uno de esto elementos para que guarde el componente y su css.
5. Modularizar los components, separar en carpetas individuales cada uno de esto elementos para que guarde el componente y su css.
6. Crear la vista Home.jsx en la carpeta views
7. Crear la vista Appointments.jsx
8. Agregamos Home a App.jsx  
   function App() {

return (

    <>

    <Home />

    </>

  );

}

export default App;

1. Crear un componente NavBar.jsx en la carpeta componentes
2. Ubicar el componente NavBar dentro de la vista Home
3. Crear los componentes necesarios para crear los componentes principales en este caso  
   LogoEmpres.jsx, NavLinks.jsx
4. Agregar los componentes menores a los que los contendrán
5. Completar los componentes y las views con la información que queremos presentar
6. Normalizar los estilos con un archivo normalize, agregamos la información de este archivo a index.css
7. Agregamos css legacy en index.css para los estilos globales
8. Agregar module.css para cada componente con los estilos específicos

M3-L8

1. Crear los estados necesarios en los archivos correspondientes
2. Crear lo que se quiere renderizar

**Guía para Crear un Frontend con React y Vite**

Esta guía te llevará paso a paso a través del proceso de creación del frontend de una aplicación web con React y Vite, desde la configuración inicial del proyecto hasta la estructura de archivos y las mejores prácticas de desarrollo.

**Fase 1: Configuración Inicial del Proyecto**

1. **Requisitos previos:**
   * Asegúrate de tener Node.js y npm (o yarn) instalados en tu sistema. Puedes descargarlos desde [https://nodejs.org/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fnodejs.org%2F).
2. **Crear un nuevo proyecto Vite:**
   * Abre tu terminal y ejecuta el siguiente comando para crear un nuevo proyecto Vite con la plantilla de React:
   * npm create vite@latest mi-proyecto --template react
   * # o con yarn:

yarn create vite mi-proyecto --template react

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

Reemplaza mi-proyecto con el nombre que quieras darle a tu proyecto.

1. **Navegar al directorio del proyecto:**

cd mi-proyecto

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Instalar dependencias:**
2. npm install
3. # o con yarn:

yarn install

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Iniciar el servidor de desarrollo:**
2. npm run dev
3. # o con yarn:

yarn dev

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

Esto iniciará el servidor de desarrollo de Vite y abrirá tu aplicación en el navegador.

**Fase 2: Estructura de Archivos y Carpetas**

Una estructura de archivos bien organizada es fundamental para mantener tu proyecto limpio y escalable. Aquí te presento una estructura recomendada:

mi-proyecto/

├── public/ # Archivos estáticos (imágenes, fuentes, etc.)

├── src/ # Código fuente de la aplicación

│ ├── components/ # Componentes reutilizables

│ │ ├── Button/ # Ejemplo de componente

│ │ │ ├── Button.jsx # Componente Button

│ │ │ └── Button.module.css # Estilos del componente (CSS Modules)

│ │ ├── Card/ # Otro ejemplo de componente

│ │ │ ├── Card.jsx # Componente Card

│ │ │ └── Card.module.css # Estilos del componente (CSS Modules)

│ │ └── ...

│ ├── pages/ # Páginas de la aplicación (vistas principales)

│ │ ├── Home/ # Ejemplo de página

│ │ │ ├── Home.jsx # Componente Home

│ │ │ └── Home.module.css # Estilos de la página (CSS Modules)

│ │ ├── About/ # Otra ejemplo de página

│ │ │ ├── About.jsx # Componente About

│ │ │ └── About.module.css # Estilos de la página (CSS Modules)

│ │ └── ...

│ ├── services/ # Servicios para interactuar con APIs o bases de datos

│ │ └── api.js # Funciones para llamadas a la API

│ ├── utils/ # Funciones de utilidad

│ │ └── helpers.js # Funciones auxiliares

│ ├── context/ # Contextos para gestionar el estado global

│ │ └── AuthContext.jsx # Ejemplo de contexto para autenticación

│ ├── App.jsx # Componente principal de la aplicación

│ ├── main.jsx # Punto de entrada de la aplicación

│ └── index.css # Estilos globales (opcional)

├── vite.config.js # Configuración de Vite

├── package.json # Dependencias y scripts del proyecto

├── README.md # Información sobre el proyecto

└── .gitignore # Archivos a ignorar por Git

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).

**Explicación de las carpetas:**

* **public/:** Contiene archivos estáticos como imágenes, fuentes o archivos HTML que no necesitan ser procesados por Vite.
* **src/:** Contiene todo el código fuente de tu aplicación React.
* **components/:** Almacena componentes reutilizables que se pueden usar en diferentes partes de la aplicación. Cada componente debe tener su propia carpeta con un archivo .jsx para el componente y un archivo .module.css (o .module.scss, etc.) para sus estilos (usando CSS Modules).
* **pages/:** Almacena los componentes que representan las diferentes páginas (vistas principales) de tu aplicación. Similar a la carpeta components/, cada página debe tener su propia carpeta con archivos .jsx y .module.css.
* **services/:** Contiene funciones para interactuar con APIs o bases de datos. Por ejemplo, podrías tener un archivo api.js con funciones para obtener datos de una API REST.
* **utils/:** Almacena funciones de utilidad que se pueden usar en diferentes partes de la aplicación.
* **context/:** Contiene contextos de React para gestionar el estado global de la aplicación. Por ejemplo, podrías tener un contexto para la autenticación del usuario.
* **App.jsx:** Es el componente principal de la aplicación, que renderiza la estructura general de la interfaz de usuario.
* **main.jsx:** Es el punto de entrada de la aplicación. Aquí es donde se renderiza el componente App en el DOM.
* **index.css:** Contiene estilos CSS globales que se aplican a toda la aplicación (opcional).
* **vite.config.js:** Contiene la configuración de Vite para tu proyecto.
* **package.json:** Define las dependencias del proyecto y los scripts para ejecutar tareas como iniciar el servidor de desarrollo, construir la aplicación, etc.
* **README.md:** Contiene información sobre el proyecto, como una descripción, instrucciones de instalación, etc.
* **.gitignore:** Especifica los archivos y carpetas que deben ser ignorados por Git al realizar el seguimiento de los cambios en el código.

**Fase 3: Componentes y Páginas**

1. **Crear componentes:**
   * Cada componente debe ser responsable de una parte específica de la interfaz de usuario.
   * Utiliza componentes funcionales con hooks para gestionar el estado y los efectos secundarios.
   * Utiliza CSS Modules para estilizar tus componentes y evitar colisiones de nombres de clases.

**Ejemplo de componente Button:**

**src/components/Button/Button.jsx:**

import styles from "./Button.module.css";

function Button({ onClick, children }) {

return (

<button className={styles.button} onClick={onClick}>

{children}

</button>

);

}

export default Button;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**src/components/Button/Button.module.css:**

.button {

background-color: #4caf50;

color: white;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Css

1. **Crear páginas:**
   * Cada página debe representar una vista principal de tu aplicación.
   * Utiliza componentes para construir las diferentes secciones de cada página.
   * Utiliza React Router para gestionar la navegación entre las páginas.

**Ejemplo de página Home:**

**src/pages/Home/Home.jsx:**

import styles from "./Home.module.css";

import Button from "../../components/Button/Button";

function Home() {

return (

<div className={styles.home}>

<h1>Página de Inicio</h1>

<Button onClick={() => alert("¡Hola!")}>Haz clic aquí</Button>

</div>

);

}

export default Home;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**src/pages/Home/Home.module.css:**

.home {

text-align: center;

padding: 20px;

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Css

**Fase 4: Gestión del Estado**

1. **Estado local:**
   * Utiliza el hook useState para gestionar el estado local de tus componentes.
   * El estado local es útil para datos que solo son relevantes para un componente específico.
2. **Estado global:**
   * Para datos que necesitan ser compartidos entre varios componentes, utiliza un sistema de gestión de estado global. Algunas opciones son:
     + **Context API:** Proporciona una forma sencilla de compartir datos entre componentes sin necesidad de pasar props manualmente a través de cada nivel del árbol de componentes.
     + **Redux:** Es una librería popular para gestionar el estado de aplicaciones complejas. Requiere más configuración que Context API, pero ofrece herramientas para depuración y gestión de la complejidad.
     + **Zustand:** Es una librería más ligera y sencilla que Redux, pero también ofrece una buena solución para gestionar el estado global.

**Ejemplo de Context API:**

**src/context/AuthContext.jsx:**

import { createContext, useState, useContext } from "react";

const AuthContext = createContext();

export function AuthProvider({ children }) {

const [user, setUser] = useState(null);

const login = (userData) => {

setUser(userData);

};

const logout = () => {

setUser(null);

};

const value = {

user,

login,

logout,

};

return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;

}

export function useAuth() {

return useContext(AuthContext);

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**src/App.jsx:**

import { AuthProvider } from "./context/AuthContext";

import Home from "./pages/Home/Home";

function App() {

return (

<AuthProvider>

<Home />

</AuthProvider>

);

}

export default App;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Fase 5: Enrutamiento**

1. **Instalar React Router:**
2. npm install react-router-dom
3. # o con yarn:

yarn add react-router-dom

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Configurar las rutas:**
   * Utiliza los componentes BrowserRouter, Route, y Link de react-router-dom para configurar las rutas de tu aplicación.

**src/App.jsx:**

import { BrowserRouter, Route, Routes } from "react-router-dom";

import Home from "./pages/Home/Home";

import About from "./pages/About/About";

function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</BrowserRouter>

);

}

export default App;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

* + Para que el enlace funcione, es necesario usar la etiqueta Link de la siguiente manera:
  + import { Link} from "react-router-dom";

<Link to="/about">A</Link>

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Fase 6: Estilos**

1. **CSS Modules:**
   * Utiliza CSS Modules para estilizar tus componentes y evitar colisiones de nombres de clases.
   * Los archivos CSS Modules tienen la extensión .module.css (o .module.scss, etc.).
   * Importa el archivo CSS Module en tu componente como un objeto y accede a las clases como propiedades de ese objeto.

**Ejemplo:**

**src/components/Button/Button.module.css:**

.button {

background-color: #4caf50;

color: white;

padding: 10px 20px;

border: none;

border-radius: 5px;

cursor: pointer;

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Css

**src/components/Button/Button.jsx:**

import styles from "./Button.module.css";

function Button({ onClick, children }) {

return (

<button className={styles.button} onClick={onClick}>

{children}

</button>

);

}

export default Button;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Fase 7: Llamadas a APIs**

1. **Crear funciones de servicio:**
   * Crea funciones en la carpeta src/services/ para encapsular la lógica de las llamadas a la API.
   * Utiliza fetch o una librería como Axios para realizar las llamadas a la API.

**Ejemplo:**

**src/services/api.js:**

const API\_URL = "https://ejemplo.com/api";

export async function getData() {

const response = await fetch(`${API\_URL}/data`);

const data = await response.json();

return data;

}

export async function createData(newData) {

const response = await fetch(`${API\_URL}/data`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(newData),

});

const data = await response.json();

return data;

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

1. **Usar las funciones de servicio en tus componentes:**
   * Importa las funciones de servicio en tus componentes y utilízalas dentro de los hooks useEffect o en los controladores de eventos.

**src/pages/Home/Home.jsx:**

import { useState, useEffect } from "react";

import { getData } from "../../services/api";

function Home() {

const [data, setData] = useState([]);

useEffect(() => {

getData().then((data) => setData(data));

}, []);

return (

<div>

<h1>Página de Inicio</h1>

{/\* Renderizar los datos \*/}

</div>

);

}

export default Home;

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).JavaScript

**Fase 8: Pruebas**

1. **Configurar un entorno de pruebas:**
   * Instala Jest y React Testing Library:
   * npm install --save-dev jest @testing-library/react @testing-library/jest-dom
   * # o con yarn:

yarn add --dev jest @testing-library/react @testing-library/jest-dom

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Escribir pruebas:**
   * Crea archivos de prueba para tus componentes y funciones de servicio.
   * Utiliza React Testing Library para simular la interacción del usuario y verificar que tus componentes se comporten como se espera.

**Fase 9: Despliegue**

1. **Construir la aplicación:**
2. npm run build
3. # o con yarn:

yarn build

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

Esto creará una carpeta dist/ con los archivos estáticos de tu aplicación.

1. **Desplegar los archivos estáticos:**
   * Puedes desplegar los archivos estáticos en un servicio de hosting como Netlify, Vercel, Firebase Hosting, o GitHub Pages.

**Consideraciones Adicionales y Mejores Prácticas:**

* **Convenciones de nombres:** Sigue convenciones de nombres consistentes para archivos, carpetas y variables.
* **Comentarios:** Escribe comentarios claros y concisos para documentar tu código.
* **Control de versiones:** Utiliza Git para realizar el seguimiento de los cambios en tu código.
* **Diseño responsivo:** Asegúrate de que tu aplicación se vea bien en diferentes tamaños de pantalla utilizando media queries y unidades relativas (em, rem, %).
* **Accesibilidad:** Diseña tu aplicación teniendo en cuenta la accesibilidad para que sea utilizable por personas con discapacidades.