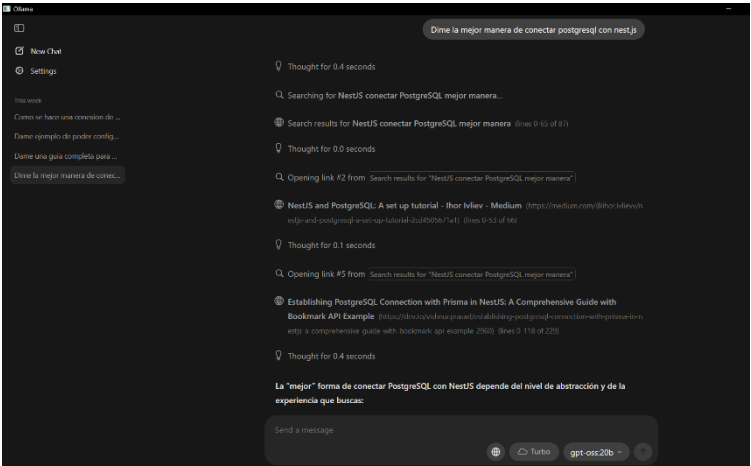


Utilización de OLLAMA, la IA para tu ordenador

Dime la mejor manera de conectar postgresql con nest.js



La "mejor" forma de conectar PostgreSQL con NestJS depende del nivel de abstracción y de la experiencia que busques:

Necesidad	Tecnologías recomendadas	Ventajas	Cuando usarla
ORM clásico (recomendado si quieres definir entidades y usar relaciones)	<code>@nestjs/typeorm</code> o <code>@nestjs/sequelize</code>	<ul style="list-style-type: none">• Genera tablas automáticamente.• Permite <code>Repository</code> y <code>@InjectRepository</code>.• Facilita transacciones y migraciones con TypeORM	Cuando trabajas con datos complejos y quieres una capa de abstracción de la base de datos.
Query-builder con tipos (para un enfoque orientado a tipos y a menos código boilerplate)	Prisma (<code>@prisma/client</code> + <code>@prisma/integrations/nestjs</code>)	<ul style="list-style-type: none">• Genera tipos TypeScript exactos.• API de consulta declarativa.• Enfoque "schema-first" con migraciones automáticas.	Al querer la mayor seguridad de tipos sin escribir SQL manual.
SQL puro / <code>nodemon</code> (simplicidad y control total)	<code>pg</code> + <code>pg-promise</code> o <code>node-postgres</code>	<ul style="list-style-type: none">• No hay abstracción; ejecutas SQL directamente.• Muy útil en micro-servicios pequeños.	Cuando la lógica de negocio es sencilla y el rendimiento de la capa de objetos no importa.

1. Configuración con TypeORM—La ruta más "NestJS-orgánica"

Send a message

Turbo

gpt-oss-20b

1. Configuración con TypeORM—La ruta más "NestJS-orgánica"

Paso 1 – Instalar dependencias

```
bash
npm i --save @nestjs/typeorm typeorm pg @nestjs/config
```

Paso 2 – Crear el archivo `.env`

```
bash
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=nestjs_demo
```

Paso 3 – Modificar `app.module.ts`

```
ts
import { Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersModule } from './users/users.module';

@Module({
  imports: [
    ConfigModule.forRoot(),
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.POSTGRES_HOST,
      port: +process.env.POSTGRES_PORT,
      username: process.env.POSTGRES_USER,
      password: process.env.POSTGRES_PASSWORD,
      database: process.env.POSTGRES_DB,
      entities: [__dirname + '/../**/*.entity{.ts,.js}'],
      synchronize: true, // prod: false + migr
      autloadEntities: true,
    }),
    inject: [ConfigService],
  ],
  UsersModule,
})
export class AppModule {}
```

Send a message

Turbo

gpt-oss-20b

```
typeormModule.forRootAsync({
  imports: [ConfigModule],
  useFactory: (config: ConfigService) => ({
    type: 'postgres',
    host: config.getString('POSTGRES_HOST'),
    port: +config.getString('POSTGRES_PORT'),
    username: config.getString('POSTGRES_USER'),
    password: config.getString('POSTGRES_PASSWORD'),
    database: config.getString('POSTGRES_DB'),
    entities: [__dirname + '/../**/*.entity{.ts,.js}'],
    synchronize: true, // prod: false + migr
    autloadEntities: true,
  }),
  inject: [ConfigService],
}),
UsersModule,
],
})
export class AppModule {}
```

"Tip: En producción, evita `synchronize: true`; utiliza migraciones (`npx typeorm migration:generate`)."

Paso 4 – Definir entidad

```
ts
// src/users/entities/user.entity.ts
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ length: 50 })
  name: string;

  @Column({ unique: true })
  email: string;
}
```

Send a message

Turbo

gpt-oss-20b

"Tip: En producción, evita `synchronize: true`; utiliza migraciones (`npx typeorm migration:generate`)."

Paso 4 – Definir entidad

```
ts
// src/users/entities/user.entity.ts
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ length: 50 })
  name: string;

  @Column({ unique: true })
  email: string;
}
```

Paso 5 – Usar el repositorio

```
ts
// src/users/users.service.ts
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from '../entities/user.entity';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User) private repository: Repository<User>,
  ) {}

  async create(createUserDto: CreateUserDto): Promise<User> {
    const user = this.repository.create(createUserDto);
    return this.repository.save(user);
  }
}
```

Send a message

Turbo

gpt-oss-20b

2.Configuración con Prisma—La opción “tipo-first”

Paso 1 – Instalar dependencias

```
bash
npm i prisma @prisma/client
npm i --save @prisma/integrations/nestjs
npm i --save-dev prisma @prisma/migrate
```

Paso 2 – Inicializar Prisma

```
npx prisma init
```

Aparecerá /prisma/schema.prisma .

Paso 3 – Configurar conexión en schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
}
```

Send a message

Paso 4 – Definir modelo

```
model User {
  id Int @id @default(autoincrement())
  name String
  email String @unique
}
```

Ejecuta migración:

```
bash
npx prisma migrate dev --name init
```

Paso 5 – Configurar módulo en NestJS

```
ts
// src/app.module.ts
import { Module } from '@nestjs/common';
import { PrismaModule } from '@prisma/integrations/nestjs';
import { ConfigModule } from '@nestjs/config';
import { UsersModule } from './users/users.module';

@Module({
  imports: [
    ConfigModule.forRoot(),
    PrismaModule.forRoot({ prisma: '@prisma/client' }),
    UsersModule,
  ],
})
```

Send a message

“Recomendación final

Para la mayoría de las aplicaciones NestJS que comienzan en 2024, Prisma es la opción “más moderna” y proporciona un flujo de trabajo sin complicaciones (autogeneración de tipos, migraciones simples). Si ya tienes experiencia con TypeORM o quieres un control más “ORM-tradicional”, @nestjs/typeorm sigue siendo una elección sólida y oficial. En ambos casos, aprovecha @nestjs/config para manejar las variables de entorno y mantén synchronize: false en producción.”

Que debo de tener en cuenta para crear mi entity, controller y service con TypeORM

Thought for 0.4 seconds

Guía práctica: Como crear Entity – Controller – Service con TypeORM en NestJS

(revisión 2024 – pensado en proyectos medianos-grandes)

1. Arquitectura recomendada

```
src
├── modules
│   └── users
│       ├── users.controller.ts
│       ├── users.service.ts
│       ├── users.repository.ts (opcional)
│       ├── users.module.ts
│       └── entities
│           └── user.entity.ts
```

Send a message

1. Arquitectura recomendada

```
src
├── modules
│   └── users
│       ├── users.controller.ts
│       ├── users.service.ts
│       ├── users.repository.ts (opcional)
│       ├── users.module.ts
│       └── entities
│           └── user.entity.ts
├── shared
│   ├── dtos
│   │   ├── create-user.dto.ts
│   │   └── update-user.dto.ts
│   ├── common
│   │   └── pipes
│   │       └── validate-id.pipe.ts
```

“¿Por qué esta separación?”

- Entity: solo lógica de datos.
- DTO: transacciones de entrada.
- Service: lógica de negocio.
- Controller: capa de HTTP.
- Module: encapsula dependencias y lo que se exporta.

2. Creando la Entity – user.entity.ts

Concepto	Cómo hacerlo	Por qué
Decorador	@Entity({ name: 'users' })	Define la tabla y su nombre explícito.
Tipos y validaciones	@Column({ type: 'varchar', length: 50 })	Asegura la columna y el tamaño.
Llaves primarias	@PrimaryGeneratedColumn()	Auto-incremento automático (PostgreSQL).
Índices y unicidad	@Index({ unique: true })	Mejora rendimiento y evita duplicados.
Relaciones	@OneToMany(...) / @ManyToOne(...)	Mapea ForeignKey, habilita carga perezosa o eager.
Hooks (listeners)	@BeforeInsert() / @Afterload()	Automatiza hashes, timestamps, logging.
Soft delete	@DeleteDateColumn()	Permite “borrar” sin remover registro.
DTO mapping	No usar en la entity, crea DTOs separados.	

```
ts
import { Entity, PrimaryGeneratedColumn, column, Unique, Index, BeforeInsert } from 'typeorm'
import * as bcrypt from 'bcrypt';

@Entity('users')
@Unique(['email'])
export class User {
  @PrimaryGeneratedColumn()
  id: number;
```

Send a message

3. DTOs – Validaciones con class-validator

```
ts
// create-user.dto.ts
import { IsString, IsEmail, MinLength, MaxLength } from 'class-validator';

export class CreateUserDto {
  @IsString()
  @MaxLength(50)
  readonly name: string;

  @IsEmail()
  readonly email: string;

  @IsString()
  @MinLength(8)
  readonly password: string;
}
```

```
ts
// update-user.dto.ts (partial update)
import { PartialType } from '@nestjs/mapped-types';
import { CreateUserDto } from './create-user.dto';

export class UpdateUserDto extends PartialType(CreateUserDto) {}
```

“Útil: los DTOs se usan en el controller para que Nest se encargue de validar antes de llegar al servicio”

Send a message

4. Service – Logica de negocio

```
ts
import { Injectable, NotFoundException, BadRequestException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository, DeleteResult, UpdateResult } from 'typeorm';
import { User } from '../entities/user.entity';
import { CreateUserDto, UpdateUserDto } from '../dtos';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private readonly usersRepo: Repository<User>,
  ) {}

  /* ----- CRUD ----- */

  async findAll(): Promise<User[]> {
    return this.usersRepo.find();
  }

  async findOne(id: number): Promise<User> {
    const user = await this.usersRepo.findOne({ where: { id } });
    if (!user) throw new NotFoundException('User not found');
    return user;
  }

  async create(dto: CreateUserDto): Promise<User> {
    const user = this.usersRepo.create(dto);
    return this.usersRepo.save(user);
  }
}
```

5. Controller – Manejando HTTP y Pipes

```
ts
import { Controller, Get, Post, Body, Param, Delete, Put, UsePipes, ValidationPipe, ParseIntPipe } from '@nestjs/common';
import { UsersService } from '../users.service';
import { CreateUserDto, UpdateUserDto } from '../dtos';

@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @Get()
  async findAll() {
    return this.usersService.findAll();
  }

  @Get('/:id')
  async findOne(@Param('id', ParseIntPipe) id: number) {
    return this.usersService.findOne(id);
  }

  @Post()
  @UsePipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true }))
  async create(@Body() dto: CreateUserDto) {
    return this.usersService.create(dto);
  }

  @Put('/:id')
```

6. Module – Encapsulando todo

```
ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersController } from '../users.controller';
import { UsersService } from '../users.service';
import { User } from '../entities/user.entity';

@Module({
  imports: [TypeOrmModule.forFeature([User])],
  controllers: [UsersController],
  providers: [UsersService],
  exports: [UsersService], // si quieres usarlo desde otros módulos
})
export class UsersModule {}
```

"Consejo: si usas repositories con lógica especializada, puedes crear un custom UsersRepository extendiendo Repository<User> y registrar mediante @EntityRepository."

7. Problemas comunes y sus soluciones

Problema	Diagnóstico	Solución
Errores de tipo "Cannot find module"	typeorm no instalado o tsconfig.json no incluye	npm i typeorm pg + tsconfig.json typeRoots o skipLibCheck

8. Resumen rápido (Checklist)

1. Entity

- @Entity(), @PrimaryGeneratedColumn, @Column con tipos y length.
- Índices, unicidad, soft delete.
- Hooks (BeforeInsert , AfterLoad).

2. DTO

- class-validator + class-transformer .
- PartialType para updates.

3. Service

- @InjectRepository .
- Métodos CRUD con NotFoundException .
- Transacciones cuando sea necesario.

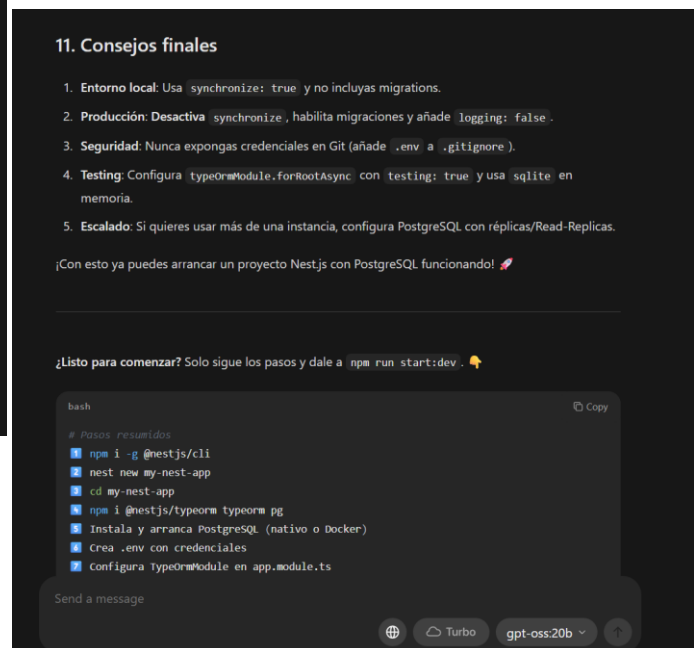
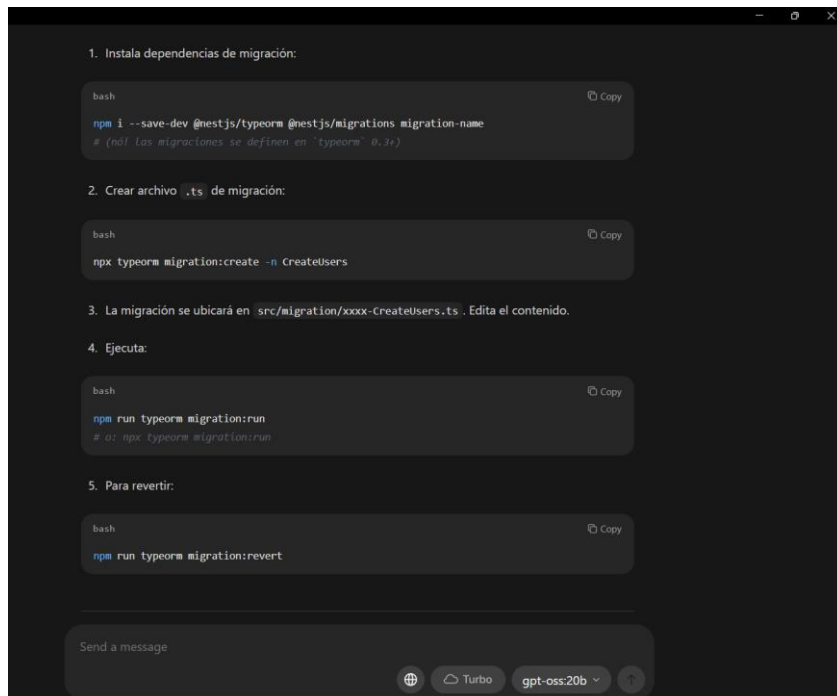
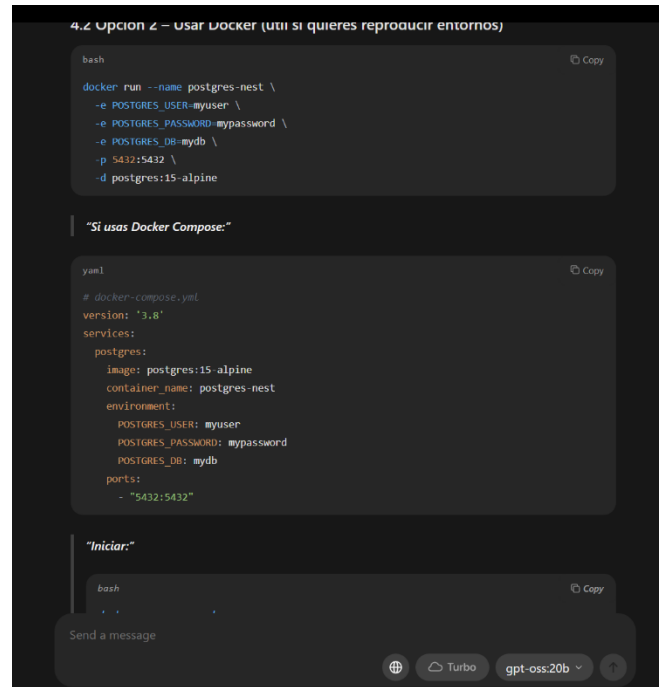
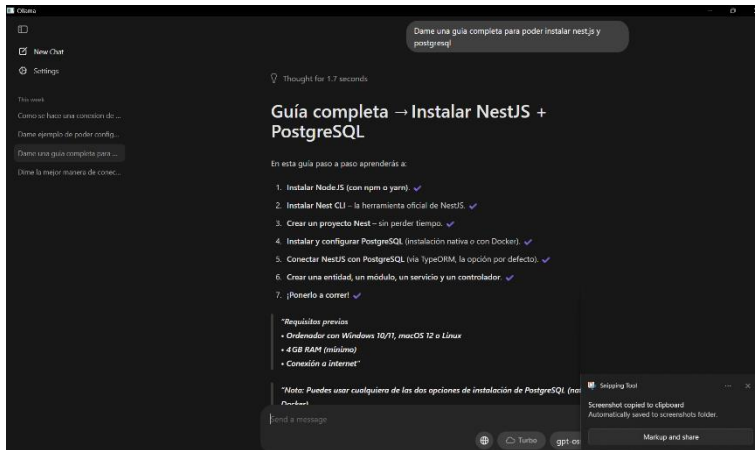
4. Controller

- @UsePipes(ValidationPipe) .
- ParseIntPipe para parámetros.
- Respuestas limpias (status, body).

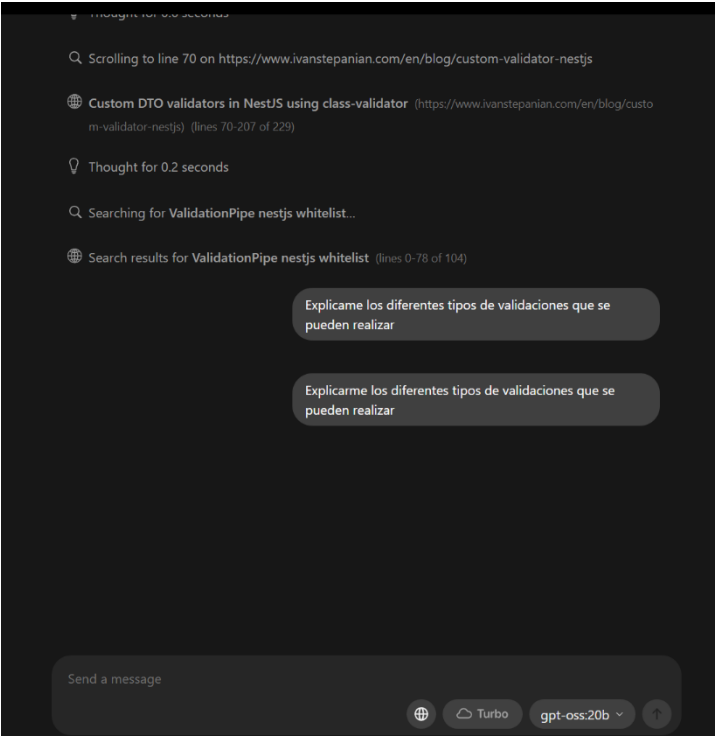
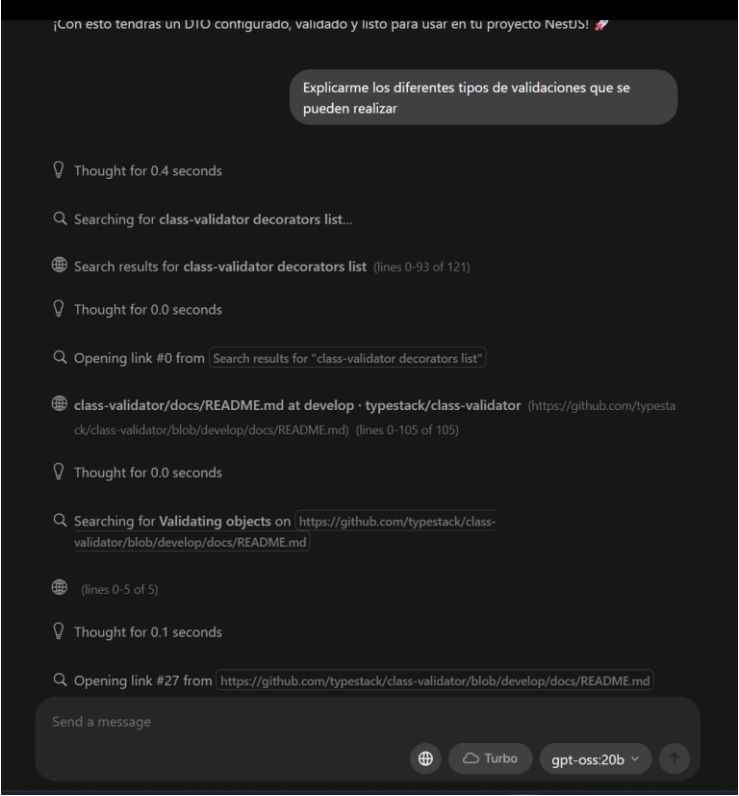
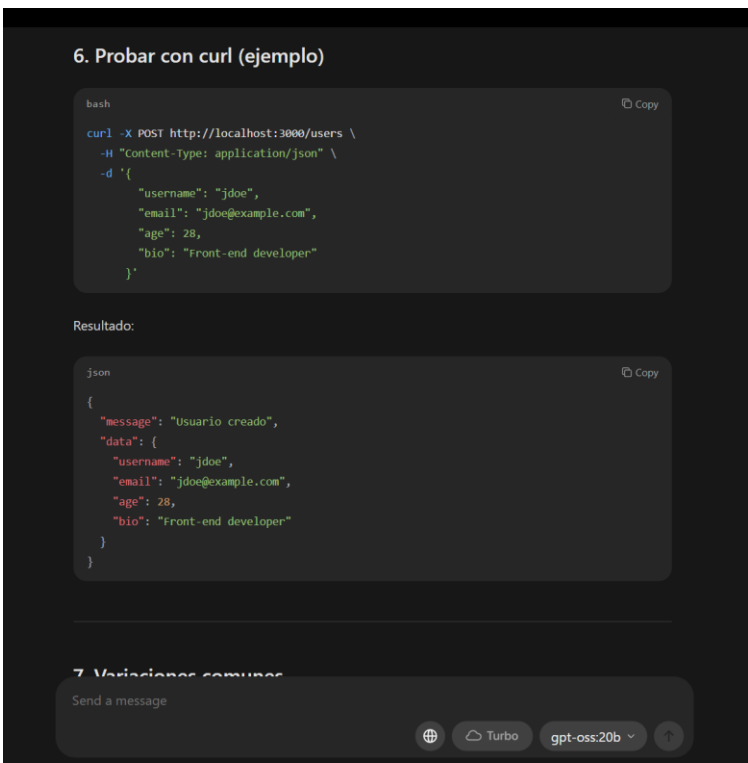
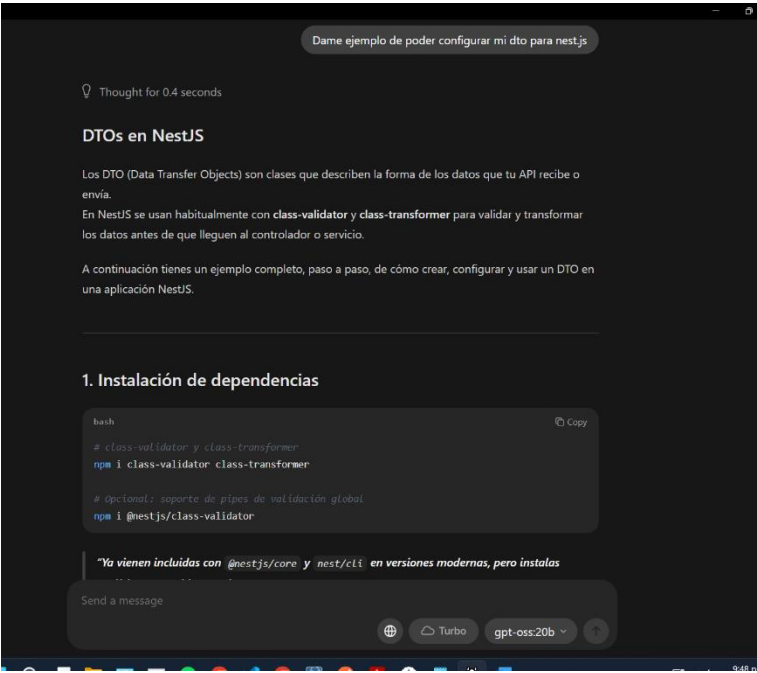
5. Module

- forFeature([User]) .

Dame una guia completa para poder instalar nest.js y postgresql



Dame ejemplo de poder configurar mi dto para nest.js



Y desde aquí no me dejo seguir preguntándole más, porque me salió un error (Ver la siguiente imagen)



New Chat

Settings

This week

Como se hace una conexion de ...

Dame ejemplo de poder config...

Dame una guia completa para ...

Dime la mejor manera de conec...

Como se hace una conexion de postgresql

Como hacer una conexion

Error

500 Internal Server Error: model requires more system memory (12.2 GiB) than is available (5.1 GiB)

Chats que se utilizaron con ChatGPT (Creo que estan todos)

1. **Implementacion de nestjs y flutter para el sharedPreferences:**
<https://chatgpt.com/share/6896b2fa-8ed4-8008-a18a-06efaafec066>
2. **Como crear una implementación para guardado que permita saber si el dispositivo y no carguen todas las tareas:**
<https://chatgpt.com/share/6896b2ca-f308-8008-be33-a45507f0de82>
3. **Solución de error que tenia con temas de espaciado en el backend:**
<https://chatgpt.com/share/6896b211-202c-8008-852b-b77ce48d7619>
4. **Nombran algo que era screen y queria saber si se podría poner views:**
<https://chatgpt.com/share/6896b00a-e02c-800b-bbea-16758c1a6a07>
5. **Problemas de ejecución con la ruta 'localhost...' en el emulador de android:** <https://chatgpt.com/share/6896b32a-aa54-8008-ac77-10a65a4473be>
6. **Metodos HTTP con flutter y dart:** <https://chatgpt.com/share/6896b01a-2670-800b-bca7-06cc319d2dc1>
7. **Problemas con la importación de http en flutter:**
<https://chatgpt.com/share/6896b273-fe74-8008-9ef6-fafc8232be24>
8. **Manejo de errores con flutter para vista al usuario:**
<https://chatgpt.com/share/6896b3e2-9884-8008-8e95-2f9dfc7cc42c>
9. **Dto y sus validaciones:** <https://chatgpt.com/share/6896b237-f800-8008-8e97-a2c8e1d25300>
10. **Error de importación:** <https://chatgpt.com/share/6896b063-a224-800b-a26f-11f77765d929>
11. **Campos en un modal:** <https://chatgpt.com/share/6896b4ad-bb28-8008-ac3c-53ebcf36c83c>
12. **Como hacer para poder cargar mi 'screen' de primerazo y modal en flutter:** <https://chatgpt.com/share/6896b29e-3e18-8008-9f69-2deb91444af7>
13. **Manejo de validaciones con los modals en frontend:**
<https://chatgpt.com/share/6896b44a-f580-8008-b88b-10e2132327dc>
14. **Concatenación y como se utiliza en flutter:**
<https://chatgpt.com/share/6896b25f-2d5c-8008-bc6a-2a5784cdc2b0>
15. **Validación del modelo de tareas:** <https://chatgpt.com/share/689519cd-7c58-800b-827d-33f6d96e899d>
16. **Instalar flutter con dart y problemas con la sincronización de git:**
<https://chatgpt.com/share/6896b145-0f70-800b-9329-a9c263866b76>
17. **Algunas dudas con la instalación de flutter con dart:**
<https://chatgpt.com/share/6896b1f5-afb0-8008-bff8-10f6844c996e>
18. **Estructura de las diferentes carpetas y diferentes dudas que tenia:**
<https://chatgpt.com/share/6896b198-c2b0-8008-86f2-582a0ed689fe>

19. Despliegue y maneras de hacerlo (Gratis):

<https://chatgpt.com/share/68978670-1e70-800b-b924-29b0ae3b7777>