**📁 database/ Directory Overview**

The database/ directory centralizes all database-related logic, promoting a clean separation of concerns and enhancing maintainability

**Suggested Structure:**



src/

├── database/

│ ├── entities/ # All entities (ORM models) go here

│ │ ├── user.entity.ts

│ │ ├── post.entity.ts

│ │ └── ...

│ ├── migrations/ # TypeORM migrations

│ ├── seeders/ # Optional: DB seeding logic for initial data

│ ├── database.module.ts # Provides TypeORM setup

│ └── database.providers.ts # Reusable DB connection/provider logic

```

---

## 📄 `entities/` – ORM Models

This folder contains all your TypeORM entity definitions, representing the database table.

\*\*Use Case:\* Define the structure of your database tables and their relationship.

\*\*Example: `user.entity.ts`\*\*



```typescript

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

@Entity()

export class User {

@PrimaryGeneratedColumn()

id: number;

@Column({ length: 100 })

name: string;

@Column({ unique: true })

email: string;

@Column()

password: string;

}

``



---

## 📄 `migrations/` – Database Migration

This directory stores migration files that track changes to your database schema over tie.

\*\*Use Case:\* Manage schema changes in a version-controlled manner, ensuring consistency across environmens.

\*\*Generating a Migration:\*



```bash

npx typeorm migration:generate -n CreateUserTable

``



\*\*Running Migrations:\*



```bash

npx typeorm migration:run

``



---

## 📄 `seeders/` – Database Seedig

Seeders populate your database with initial or test data. This is particularly useful during development or testing phaes.

\*\*Use Case\*\* Initialize the database with default or test dta.

\*\*Example: `user.seeder.ts`\*



```typescript

import { DataSource } from 'typeorm';

import { User } from '../entities/user.entity';

export const seedUsers = async (dataSource: DataSource) => {

const userRepository = dataSource.getRepository(User);

const users = [

{ name: 'Alice', email: 'alice@example.com', password: 'securepass' },

{ name: 'Bob', email: 'bob@example.com', password: 'securepass' },

];

await userRepository.save(users);

};

``



\*\*Running the Seeder:\*



```typescript

import { DataSource } from 'typeorm';

import { seedUsers } from './seeders/user.seeder';

import { AppDataSource } from './data-source';

AppDataSource.initialize()

.then(async () => {

await seedUsers(AppDataSource);

console.log('Seeding completed.');

})

.catch((error) => console.error('Seeding failed:', error));

``



---

## 📄 `database.providers.ts` – Database Connection Providrs

This file defines and exports the database connection providers, facilitating dependency injection throughout your applicaion.

\*\*Use Cas:\*\* Centralize and manage database connection configuratons.

\*\*Example\*\*



```typescript

import { DataSource } from 'typeorm';

import { User } from './entities/user.entity';

export const AppDataSource = new DataSource({

type: 'postgres',

host: process.env.DB\_HOST,

port: parseInt(process.env.DB\_PORT || '5432', 10),

username: process.env.DB\_USERNAME,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_NAME,

entities: [User],

migrations: [\_\_dirname + '/migrations/\*.ts'],

synchronize: false, // Set to false in production

});```



---

## 📄 `database.module.ts` – Database Moule

This module integrates the database connection into your NestJS appliction.

\*\*Use Cae:\*\* Provide a centralized module for database interactions, making it easily importable across the appliction.

\*\*Exampl:\*\*



```typescript

import { Module } from '@nestjs/common';

import { TypeOrmModule } from '@nestjs/typeorm';

import { AppDataSource } from './database.providers';

@Module({

imports: [TypeOrmModule.forRoot(AppDataSource.options)],

})

export class DatabaseModule {



**🔄 Integration with Other Moules**

To utilize the database entities and services in other modules, import the TypeOrmModule and specify the entities yo need.

**Use Cse:** Enable feature modules to interact with specific entities and reposiories.

**Exampe:**



import { Module } from '@nestjs/common';

import { TypeOrmModule } from '@nestjs/typeorm';

import { User } from '../database/entities/user.entity';

import { UserService } from './user.service';

@Module({

imports: [TypeOrmModule.forFeature([User])],

providers: [UserService],

})

export class UserModule }



**✅ Best Practices**

* **Environment Variabes:** Store sensitive information like database credentials in environment variables and access them using a configuration srvice.
* **Migratins:** Use migrations to manage schema changes, ensuring consistency across different enviroments.
* **Seedng:** Implement seeders to populate the database with initial data, which is especially useful for testing and develpment.
* **Modularty:** Keep your database logic modular to enhance maintainability and scalaility.

By structuring your database/ directory as outlined above, you create a robust foundation for managing database interactions within your NestJS application.