**✅ Updated src/ Directory Structure**

src/

├── app.module.ts # Root module to bootstrap everything

├── main.ts # Entry point of the application

│

├── config/

│ ├── app.config.ts # Application-wide configs (port, name, environment)

│ ├── auth.config.ts # Auth & JWT configuration

│ ├── cache.config.ts # Cache (e.g., Redis) configuration

│ ├── mail.config.ts # Email configuration

│ ├── sms-gateway.config.ts # SMS Gateway configurations

│ ├── notification.config.ts # Notification gateway configuration for push notification

│ ├── file.config.ts # File storage configuration (local, S3, etc.)

│ ├── kafka.config.ts # Kafka integration configuration

│ ├── queue.config.ts # Queue configuration (e.g., BullMQ / RabbitMQ)

│ ├── rate-limit.config.ts # Rate-limiting & throttling configuration

│ ├── websocket.config.ts # WebSocket configuration (for real-time updates)

│ ├── external-api.config.ts # External API communication configuration

│ └── config.module.ts # Exports all configs for global injection

│

├── constants/ # Application-wide constants, enums, tokens

│ ├── app.constants.ts

│ ├── cache.constants.ts

│ └── roles.enum.ts

│

├── database/ # Centralized DB logic

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

│ │ ├── user.entity.ts

│ │ ├── post.entity.ts

│ │ └── ...

│ ├── migrations/ # TypeORM / Prisma migrations

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

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

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

│

├── modules/ # All core feature modules grouped here

│ ├── auth/ # Authentication & Authorization

│ ├── user/ # User management

│ ├── post/ # Example blog/post module

│ └── ... # More modules (notifications, uploads, etc.)

│

├── common/ # Shared logic across modules

│ ├── decorators/ # Custom decorators (e.g., @Public(), @Roles())

│ ├── filters/ # Global Exception filters

│ ├── guards/ # RBAC, Auth guards

│ ├── interceptors/ # Logging, Tracing, Response wrapping

│ ├── middleware/ # Express-style middleware (e.g., Request ID)

│ ├── pipes/ # Global validation pipes

│ ├── services/ # Shared services (e.g., LoggerService)

│ └── utils/ # Helper functions, utilities

│

├── interfaces/ # App-level types and interfaces

│ ├── response.interface.ts

│ ├── pagination.interface.ts

│ └── ...

│

├── helpers/ # Utility classes and shared logic

│ ├── file.helper.ts # File path manipulation, storage helper

│ ├── string.helper.ts

│ └── ...

│

├── websockets/ # Real-time gateway layer

│ ├── events/ # Event constants or handlers

│ ├── dashboard.gateway.ts # Real-time dashboard updates

│ ├── notification.gateway.ts # Real-time user notifications

│ └── websocket.module.ts # Combines all gateways

│

├── integrations/ # Third-party or external API integration

│ ├── payment/ # Example integration (e.g., Stripe)

│ ├── email/ # Email service provider

│ └── ...

│

├── jobs/ # Background jobs / queues

│ ├── consumers/ # Bull consumers or event handlers

│ ├── producers/ # Job producers

│ ├── queues.module.ts # Register queues

│ └── ...

│

├── templates/ # Email or system templates

│ ├── email/ # Email HTML/Text templates

│ ├── notifications/ # Push or system notification templates

│ └── ...

│

├── uploads/ # Uploaded files if using local storage

│ └── (optional) folder structure based on feature

│

├── audit/

│ ├── audit-log.service.ts # Service handling audit log logic

│ ├── audit-log.interceptor.ts # Interceptor capturing audit logs

│ ├── audit-log.decorator.ts # Custom decorator for audit logging

│ └── audit-log.module.ts # Module encapsulating audit logging components

│

└── logging/

├── logger.service.ts # Custom logger service integrating with Winston or similar

└── logger.module.ts # Module for logging

**🧠 Purpose and Reasoning for Key Decisions**

**✅ Centralized database/ Directory**

* **Why?** Avoid circular dependencies between modules when entities are reused.
* **Use Case:** You have relationships across many entities — ex: User ↔ Post ↔ AuditLog.
* **With Migrations:** TypeORM/Prisma migrations become easier when all entities are in one place.
* **Separation of Concerns:** Keeps DB logic, migrations, seeding, providers in one domain.

Example: user.entity.ts in database/entities/

@Entity('users')

export class User {

@PrimaryGeneratedColumn()

id: number;

@Column({ unique: true })

email: string;

@Column()

password: string;

@CreateDateColumn()

createdAt: Date;

}

Example: database.module.ts

@Module({

imports: [

TypeOrmModule.forRootAsync({

imports: [ConfigModule],

useFactory: (config: ConfigService) => ({

type: 'postgres',

host: config.get('database.host'),

port: config.get('database.port'),

username: config.get('database.user'),

password: config.get('database.pass'),

database: config.get('database.name'),

entities: [\_\_dirname + '/entities/\*.entity{.ts,.js}'],

synchronize: false,

}),

inject: [ConfigService],

}),

],

})

export class DatabaseModule {}

**✅ Modular & Scalable**

* Every domain (user, auth, post, file-upload, notification) stays in modules/.
* Shared features like decorators, guards, interceptors live in common/.

**✅ Real-Time Ready**

* Real-time dashboards and notifications go in websockets/.
* Gateways use @WebSocketGateway() from NestJS with custom event emitters.

**✅ API & Queue Integration**

* integrations/ is for external service handling.
* jobs/ holds queue logic with consumer/producer patterns for background tasks.