**NestJS Customer API** with:

* **URI versioning** → /api/v1/... and /api/v2/...
* **PostgreSQL + TypeORM (>=0.3)**
* **Swagger docs per version** → /docs/v1 and /docs/v2
* **Auto table create (dev)** and **migrations (prod)**

I’ll keep it step-by-step with file blocks you can drop into a fresh project.

**0) New project & dependencies**

npm i -g @nestjs/cli

nest new customer-api

cd customer-api

# DB + config + validation

npm i @nestjs/typeorm typeorm pg @nestjs/config class-validator class-transformer

# Swagger

npm i @nestjs/swagger swagger-ui-express

# TypeORM CLI & TS runtime

npm i -D ts-node

**1) (Optional) Postgres via Docker for local dev**

# docker-compose.yml (project root)

version: "3.9"

services:

pg:

image: postgres:16

environment:

POSTGRES\_USER: nest

POSTGRES\_PASSWORD: nest

POSTGRES\_DB: customerdb

ports: ["5432:5432"]

volumes: [pgdata:/var/lib/postgresql/data]

volumes:

pgdata:

docker compose up -d

**2) Environment**

# .env

NODE\_ENV=development

DB\_HOST=localhost

DB\_PORT=5432

DB\_USER=nest

DB\_PASS=nest

DB\_NAME=customerdb

# dev only: let TypeORM auto-create tables

TYPEORM\_SYNC=true

**3) Folder layout**

src/

├─ app.module.ts

├─ main.ts

└─ customers/

├─ customer.entity.ts

├─ address.entity.ts

├─ customer.service.ts

├─ dto/

│ ├─ address.dto.ts

│ ├─ create-customer.dto.ts

│ └─ update-customer.dto.ts

├─ v1/customer.controller.ts

├─ v2/customer.controller.ts

├─ customer.v1.module.ts

└─ customer.v2.module.ts

typeorm.config.ts

**4) Bootstrap with API versioning + Swagger per version**

// src/main.ts

import { NestFactory } from '@nestjs/core';

import { AppModule } from './app.module';

import { ValidationPipe, VersioningType } from '@nestjs/common';

import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';

import { CustomerV1Module } from './customers/customer.v1.module';

import { CustomerV2Module } from './customers/customer.v2.module';

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.setGlobalPrefix('api');

app.enableVersioning({ type: VersioningType.URI }); // => /api/v1, /api/v2

app.useGlobalPipes(new ValidationPipe({

whitelist: true,

transform: true,

forbidNonWhitelisted: true,

transformOptions: { enableImplicitConversion: true },

}));

// Swagger: V1

const v1 = new DocumentBuilder()

.setTitle('Customer API - V1')

.setDescription('Endpoints for /api/v1')

.setVersion('1.0')

.build();

const docV1 = SwaggerModule.createDocument(app, v1, {

include: [CustomerV1Module],

deepScanRoutes: true,

});

SwaggerModule.setup('docs/v1', app, docV1);

// Swagger: V2

const v2 = new DocumentBuilder()

.setTitle('Customer API - V2')

.setDescription('Endpoints for /api/v2')

.setVersion('2.0')

.build();

const docV2 = SwaggerModule.createDocument(app, v2, {

include: [CustomerV2Module],

deepScanRoutes: true,

});

SwaggerModule.setup('docs/v2', app, docV2);

await app.listen(3000);

console.log('Docs:', 'http://localhost:3000/docs/v1', '|', 'http://localhost:3000/docs/v2');

}

bootstrap();

**5) AppModule + DB config**

// src/app.module.ts

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

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

import { ConfigModule, ConfigService } from '@nestjs/config';

import { CustomerV1Module } from './customers/customer.v1.module';

import { CustomerV2Module } from './customers/customer.v2.module';

@Module({

imports: [

ConfigModule.forRoot({ isGlobal: true }),

TypeOrmModule.forRootAsync({

inject: [ConfigService],

useFactory: (cfg: ConfigService) => ({

type: 'postgres',

host: cfg.get<string>('DB\_HOST'),

port: cfg.get<number>('DB\_PORT'),

username: cfg.get<string>('DB\_USER'),

password: cfg.get<string>('DB\_PASS'),

database: cfg.get<string>('DB\_NAME'),

autoLoadEntities: true, // picks up @Entity classes

synchronize: cfg.get('TYPEORM\_SYNC') === 'true', // dev only!

// logging: ['schema', 'error'], // handy in dev

}),

}),

CustomerV1Module,

CustomerV2Module,

],

})

export class AppModule {}

**6) Entities (Customer ⇄ Address)**

// src/customers/customer.entity.ts

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

import { Address } from './address.entity';

@Entity('customers')

export class Customer {

@PrimaryGeneratedColumn()

id!: number;

@Column({ length: 100 })

@Index()

firstName!: string;

@Column({ length: 100 })

lastName!: string;

@Column({ length: 150, unique: true })

email!: string;

@Column({ length: 20, nullable: true })

phone?: string;

@OneToMany(() => Address, a => a.customer, {

cascade: ['insert', 'update'],

eager: true, // include addresses in queries automatically

})

addresses!: Address[];

}

// src/customers/address.entity.ts

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

import { Customer } from './customer.entity';

@Entity('addresses')

export class Address {

@PrimaryGeneratedColumn()

id!: number;

@Column({ length: 30 })

doorNumber!: string;

@Column({ length: 120 })

@Index()

street!: string;

@Column({ length: 80 })

city!: string;

@Column({ length: 80 })

state!: string;

@Column({ length: 12 })

zip!: string;

@Column({ length: 80, default: 'India' })

country!: string;

@ManyToOne(() => Customer, c => c.addresses, { onDelete: 'CASCADE' })

customer!: Customer;

}

**7) DTOs**

// src/customers/dto/address.dto.ts

import { IsString, MaxLength } from 'class-validator';

export class AddressDto {

@IsString() @MaxLength(30) doorNumber!: string;

@IsString() @MaxLength(120) street!: string;

@IsString() @MaxLength(80) city!: string;

@IsString() @MaxLength(80) state!: string;

@IsString() @MaxLength(12) zip!: string;

@IsString() @MaxLength(80) country?: string;

}

// src/customers/dto/create-customer.dto.ts

import { IsEmail, IsOptional, IsString, MaxLength, ValidateNested, ArrayMinSize, IsArray } from 'class-validator';

import { Type } from 'class-transformer';

import { AddressDto } from './address.dto';

export class CreateCustomerDto {

@IsString() @MaxLength(100) firstName!: string;

@IsString() @MaxLength(100) lastName!: string;

@IsEmail() @MaxLength(150) email!: string;

@IsOptional() @IsString() @MaxLength(20) phone?: string;

@IsArray()

@ValidateNested({ each: true })

@Type(() => AddressDto)

@ArrayMinSize(1)

addresses!: AddressDto[];

}

// src/customers/dto/update-customer.dto.ts

import { PartialType } from '@nestjs/mapped-types';

import { CreateCustomerDto } from './create-customer.dto';

export class UpdateCustomerDto extends PartialType(CreateCustomerDto) {}

**8) Service**

// src/customers/customer.service.ts

import { Injectable, NotFoundException } from '@nestjs/common';

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

import { Repository } from 'typeorm';

import { Customer } from './customer.entity';

import { CreateCustomerDto } from './dto/create-customer.dto';

import { UpdateCustomerDto } from './dto/update-customer.dto';

@Injectable()

export class CustomerService {

constructor(@InjectRepository(Customer) private readonly repo: Repository<Customer>) {}

create(dto: CreateCustomerDto) {

return this.repo.save(this.repo.create(dto));

}

findAll() {

return this.repo.find({ order: { id: 'DESC' } }); // addresses eager-loaded

}

async findOne(id: number) {

const c = await this.repo.findOne({ where: { id } });

if (!c) throw new NotFoundException('Customer not found');

return c;

}

async update(id: number, dto: UpdateCustomerDto) {

await this.findOne(id);

await this.repo.update({ id }, dto as any);

return this.findOne(id);

}

async remove(id: number) {

await this.findOne(id);

await this.repo.delete({ id });

return { deleted: true };

}

}

**9) Versioned controllers (v1 full, v2 lite)**

// src/customers/v1/customer.controller.ts

import { Controller, Get, Post, Body, Param, Patch, Delete, ParseIntPipe } from '@nestjs/common';

import { ApiTags, ApiOperation } from '@nestjs/swagger';

import { CustomerService } from '../customer.service';

import { CreateCustomerDto } from '../dto/create-customer.dto';

import { UpdateCustomerDto } from '../dto/update-customer.dto';

@ApiTags('customers')

@Controller({ path: 'customers', version: '1' }) // /api/v1/customers

export class CustomerV1Controller {

constructor(private readonly svc: CustomerService) {}

@Post()

@ApiOperation({ summary: 'Create customer with addresses (v1)' })

create(@Body() dto: CreateCustomerDto) {

return this.svc.create(dto);

}

@Get()

@ApiOperation({ summary: 'List customers (v1)' })

findAll() {

return this.svc.findAll();

}

@Get(':id')

@ApiOperation({ summary: 'Get customer by id (v1)' })

findOne(@Param('id', ParseIntPipe) id: number) {

return this.svc.findOne(id);

}

@Patch(':id')

@ApiOperation({ summary: 'Update customer (v1)' })

update(@Param('id', ParseIntPipe) id: number, @Body() dto: UpdateCustomerDto) {

return this.svc.update(id, dto);

}

@Delete(':id')

@ApiOperation({ summary: 'Delete customer (v1)' })

remove(@Param('id', ParseIntPipe) id: number) {

return this.svc.remove(id);

}

}

// src/customers/v2/customer.controller.ts

import { Controller, Get, Param, ParseIntPipe } from '@nestjs/common';

import { ApiTags, ApiOperation } from '@nestjs/swagger';

import { CustomerService } from '../customer.service';

@ApiTags('customers')

@Controller({ path: 'customers', version: '2' }) // /api/v2/customers

export class CustomerV2Controller {

constructor(private readonly svc: CustomerService) {}

@Get()

@ApiOperation({ summary: 'List customers (v2 lite projection)' })

async findAllLite() {

const rows = await this.svc.findAll();

return rows.map(r => ({

id: r.id,

name: `${r.firstName} ${r.lastName}`,

addressCount: r.addresses?.length ?? 0,

}));

}

@Get(':id')

@ApiOperation({ summary: 'Get single customer (v2 lite)' })

async findOneLite(@Param('id', ParseIntPipe) id: number) {

const r = await this.svc.findOne(id);

return { id: r.id, name: `${r.firstName} ${r.lastName}` };

}

}

**10) Version-specific modules (lets Swagger isolate v1/v2)**

// src/customers/customer.v1.module.ts

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

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

import { Customer } from './customer.entity';

import { Address } from './address.entity';

import { CustomerService } from './customer.service';

import { CustomerV1Controller } from './v1/customer.controller';

@Module({

imports: [TypeOrmModule.forFeature([Customer, Address])],

providers: [CustomerService],

controllers: [CustomerV1Controller],

})

export class CustomerV1Module {}

// src/customers/customer.v2.module.ts

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

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

import { Customer } from './customer.entity';

import { Address } from './address.entity';

import { CustomerService } from './customer.service';

import { CustomerV2Controller } from './v2/customer.controller';

@Module({

imports: [TypeOrmModule.forFeature([Customer, Address])],

providers: [CustomerService],

controllers: [CustomerV2Controller],

})

export class CustomerV2Module {}

**11) TypeORM CLI config (for migrations)**

// typeorm.config.ts (project root)

import 'dotenv/config';

import { DataSource } from 'typeorm';

import { Customer } from './src/customers/customer.entity';

import { Address } from './src/customers/address.entity';

export default new DataSource({

type: 'postgres',

host: process.env.DB\_HOST,

port: +(process.env.DB\_PORT || 5432),

username: process.env.DB\_USER,

password: process.env.DB\_PASS,

database: process.env.DB\_NAME,

entities: [Customer, Address],

migrations: ['dist/migrations/\*.js'],

});

Add NPM scripts:

// package.json (only the "scripts" section shown)

{

"scripts": {

"build": "tsc -p tsconfig.build.json",

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

"start:dev": "nest start --watch",

"mig:gen": "npm run build && node --loader ts-node/esm ./node\_modules/typeorm/cli.js -d ./typeorm.config.ts migration:generate ./src/migrations/auto",

"mig:run": "npm run build && node ./node\_modules/typeorm/cli.js -d dist/typeorm.config.js migration:run",

"mig:revert": "npm run build && node ./node\_modules/typeorm/cli.js -d dist/typeorm.config.js migration:revert"

}

}

**Dev**: keep TYPEORM\_SYNC=true and let TypeORM make tables.  
**Prod**: set TYPEORM\_SYNC=false, then npm run mig:gen → npm run mig:run.

**12) Run & test**

Start:

npm run start:dev

# Swagger:

# http://localhost:3000/docs/v1

# http://localhost:3000/docs/v2

Create (v1):

curl -X POST http://localhost:3000/api/v1/customers \

-H "Content-Type: application/json" \

-d '{

"firstName":"Ravi",

"lastName":"Kumar",

"email":"ravi.kumar@example.com",

"phone":"9876543210",

"addresses":[

{"doorNumber":"12A","street":"Anna Salai","city":"Chennai","state":"TN","zip":"600002","country":"India"}

]

}'

List:

curl http://localhost:3000/api/v1/customers

curl http://localhost:3000/api/v2/customers

Get by id:

curl http://localhost:3000/api/v1/customers/1