**1) Install deps**

npm i @nestjs/passport @nestjs/jwt passport passport-local passport-jwt bcrypt

npm i -D @types/passport-local @types/passport-jwt

**2) Env (development)**

.env

JWT\_SECRET=superlongrandomsecret

JWT\_EXPIRES=3600s

**3) Users module (Mongo + password hash)**

src/users/schemas/user.schema.ts

import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';

import { HydratedDocument } from 'mongoose';

export type UserDocument = HydratedDocument<User>;

@Schema({ collection: 'users', timestamps: true })

export class User {

@Prop({ required: true, unique: true, lowercase: true, trim: true })

email: string;

@Prop({ required: true })

passwordHash: string;

@Prop({ type: [String], default: ['USER'] })

roles: string[];

}

export const UserSchema = SchemaFactory.createForClass(User);

src/users/dto/create-user.dto.ts

import { IsEmail, MinLength, IsOptional, IsArray } from 'class-validator';

export class CreateUserDto {

@IsEmail() email: string;

@MinLength(8) password: string;

@IsOptional() @IsArray() roles?: string[];

}

src/users/users.service.ts

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

import { InjectModel } from '@nestjs/mongoose';

import { Model } from 'mongoose';

import \* as bcrypt from 'bcrypt';

import { User, UserDocument } from './schemas/user.schema';

import { CreateUserDto } from './dto/create-user.dto';

@Injectable()

export class UsersService {

constructor(@InjectModel(User.name) private model: Model<UserDocument>) {}

async create(dto: CreateUserDto) {

const exists = await this.model.findOne({ email: dto.email }).lean();

if (exists) throw new ConflictException('Email already registered');

const passwordHash = await bcrypt.hash(dto.password, 12);

const user = await this.model.create({ email: dto.email, passwordHash, roles: dto.roles ?? ['USER'] });

return { id: user.\_id.toString(), email: user.email, roles: user.roles };

}

async findByEmail(email: string) {

const u = await this.model.findOne({ email }).lean();

if (!u) throw new NotFoundException('User not found');

return u;

}

async validateUser(email: string, password: string) {

const u = await this.model.findOne({ email });

if (!u) return null;

const ok = await bcrypt.compare(password, u.passwordHash);

if (!ok) return null;

return { id: u.\_id.toString(), email: u.email, roles: u.roles };

}

}

src/users/users.module.ts

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

import { MongooseModule } from '@nestjs/mongoose';

import { UsersService } from './users.service';

import { User, UserSchema } from './schemas/user.schema';

@Module({

imports: [MongooseModule.forFeature([{ name: User.name, schema: UserSchema }])],

providers: [UsersService],

exports: [UsersService],

})

export class UsersModule {}

**4) Auth module (Local login → JWT)**

src/auth/strategies/local.strategy.ts

import { Strategy } from 'passport-local';

import { PassportStrategy } from '@nestjs/passport';

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

import { AuthService } from '../auth.service';

@Injectable()

export class LocalStrategy extends PassportStrategy(Strategy) {

constructor(private authService: AuthService) {

super({ usernameField: 'email' }); // POST body {email, password}

}

async validate(email: string, password: string) {

const user = await this.authService.validateUser(email, password);

if (!user) throw new UnauthorizedException('Invalid credentials');

return user;

}

}

src/auth/strategies/jwt.strategy.ts

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

import { PassportStrategy } from '@nestjs/passport';

import { ExtractJwt, Strategy } from 'passport-jwt';

@Injectable()

export class JwtStrategy extends PassportStrategy(Strategy) {

constructor() {

super({

jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

ignoreExpiration: false,

secretOrKey: process.env.JWT\_SECRET!,

});

}

async validate(payload: { sub: string; email: string; roles: string[] }) {

// attaches to request.user

return { userId: payload.sub, email: payload.email, roles: payload.roles };

}

}

src/auth/guards/local-auth.guard.ts

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

import { AuthGuard } from '@nestjs/passport';

@Injectable()

export class LocalAuthGuard extends AuthGuard('local') {}

src/auth/guards/jwt-auth.guard.ts

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

import { AuthGuard } from '@nestjs/passport';

@Injectable()

export class JwtAuthGuard extends AuthGuard('jwt') {}

src/auth/roles.decorator.ts

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

export const ROLES\_KEY = 'roles';

export const Roles = (...roles: string[]) => SetMetadata(ROLES\_KEY, roles);

src/auth/roles.guard.ts

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

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

import { ROLES\_KEY } from './roles.decorator';

@Injectable()

export class RolesGuard implements CanActivate {

constructor(private reflector: Reflector) {}

canActivate(ctx: ExecutionContext): boolean {

const required = this.reflector.getAllAndOverride<string[]>(ROLES\_KEY, [

ctx.getHandler(),

ctx.getClass(),

]);

if (!required?.length) return true;

const { user } = ctx.switchToHttp().getRequest();

return required.some((r) => user?.roles?.includes(r));

}

}

src/auth/auth.service.ts

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

import { JwtService } from '@nestjs/jwt';

import { UsersService } from '../users/users.service';

@Injectable()

export class AuthService {

constructor(private users: UsersService, private jwt: JwtService) {}

async validateUser(email: string, password: string) {

return this.users.validateUser(email, password);

}

async login(user: { id: string; email: string; roles: string[] }) {

const payload = { sub: user.id, email: user.email, roles: user.roles };

return {

access\_token: await this.jwt.signAsync(payload),

token\_type: 'Bearer',

expires\_in: process.env.JWT\_EXPIRES ?? '3600s',

};

}

}

src/auth/auth.controller.ts

import { Controller, Post, Body, UseGuards, Request } from '@nestjs/common';

import { AuthService } from './auth.service';

import { LocalAuthGuard } from './guards/local-auth.guard';

@Controller('auth')

export class AuthController {

constructor(private auth: AuthService) {}

@Post('login')

@UseGuards(LocalAuthGuard)

async login(@Request() req) {

// req.user is from LocalStrategy.validate()

return this.auth.login(req.user);

}

}

src/auth/auth.module.ts

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

import { JwtModule } from '@nestjs/jwt';

import { PassportModule } from '@nestjs/passport';

import { UsersModule } from '../users/users.module';

import { AuthService } from './auth.service';

import { LocalStrategy } from './strategies/local.strategy';

import { JwtStrategy } from './strategies/jwt.strategy';

import { AuthController } from './auth.controller';

@Module({

imports: [

UsersModule,

PassportModule,

JwtModule.register({

secret: process.env.JWT\_SECRET,

signOptions: { expiresIn: process.env.JWT\_EXPIRES || '3600s' },

}),

],

providers: [AuthService, LocalStrategy, JwtStrategy],

controllers: [AuthController],

exports: [JwtModule],

})

export class AuthModule {}

**5) Registration endpoint (optional but handy)**

src/users/users.controller.ts

import { Body, Controller, Post } from '@nestjs/common';

import { UsersService } from './users.service';

import { CreateUserDto } from './dto/create-user.dto';

@Controller('users')

export class UsersController {

constructor(private users: UsersService) {}

@Post('register')

register(@Body() dto: CreateUserDto) {

return this.users.create(dto);

}

}

Update module:

// users.module.ts

import { UsersController } from './users.controller';

...

controllers: [UsersController],

**6) Account API protected by JWT (+ role-based)**

Example Account schema & controller.

src/accounts/schemas/account.schema.ts

import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';

import { HydratedDocument } from 'mongoose';

export type AccountDocument = HydratedDocument<Account>;

@Schema({ collection: 'accounts', timestamps: false, versionKey: false })

export class Account {

@Prop({ unique: true, required: true })

accountNo: number;

@Prop({ required: true })

runningTotal: number;

@Prop({ required: true })

openingDate: Date;

@Prop({ required: true })

ownerUserId: string; // link to req.user.userId

}

export const AccountSchema = SchemaFactory.createForClass(Account);

AccountSchema.index({ accountNo: 1 }, { unique: true });

src/accounts/dto/create-account.dto.ts

import { IsInt, IsISO8601, IsNumber, Min } from 'class-validator';

export class CreateAccountDto {

@IsInt() accountNo: number;

@IsNumber() @Min(0) runningTotal: number;

@IsISO8601() openingDate: string; // ISO string in request

}

src/accounts/accounts.service.ts

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

import { InjectModel } from '@nestjs/mongoose';

import { Model } from 'mongoose';

import { Account, AccountDocument } from './schemas/account.schema';

import { CreateAccountDto } from './dto/create-account.dto';

@Injectable()

export class AccountsService {

constructor(@InjectModel(Account.name) private model: Model<AccountDocument>) {}

async create(dto: CreateAccountDto, ownerUserId: string) {

const acc = await this.model.create({

accountNo: dto.accountNo,

runningTotal: dto.runningTotal,

openingDate: new Date(dto.openingDate),

ownerUserId,

});

return { id: acc.\_id.toString(), ...acc.toObject() };

}

async findMine(ownerUserId: string) {

return this.model.find({ ownerUserId }).lean();

}

async findOneAuthorized(id: string, requester: { userId: string; roles: string[] }) {

const acc = await this.model.findById(id).lean();

if (!acc) throw new NotFoundException('Account not found');

const isOwner = acc.ownerUserId === requester.userId;

const isAdmin = requester.roles?.includes('ADMIN');

if (!isOwner && !isAdmin) throw new ForbiddenException('Not allowed');

return acc;

}

}

src/accounts/accounts.controller.ts

import { Body, Controller, Get, Param, Post, UseGuards, Request } from '@nestjs/common';

import { AccountsService } from './accounts.service';

import { CreateAccountDto } from './dto/create-account.dto';

import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';

import { RolesGuard } from '../auth/roles.guard';

import { Roles } from '../auth/roles.decorator';

@Controller('accounts')

@UseGuards(JwtAuthGuard, RolesGuard)

export class AccountsController {

constructor(private accounts: AccountsService) {}

@Post()

@Roles('USER', 'ADMIN') // must be authenticated and have one of these roles

create(@Body() dto: CreateAccountDto, @Request() req) {

return this.accounts.create(dto, req.user.userId);

}

@Get('me')

@Roles('USER', 'ADMIN')

myAccounts(@Request() req) {

return this.accounts.findMine(req.user.userId);

}

@Get(':id')

@Roles('USER', 'ADMIN')

getOne(@Param('id') id: string, @Request() req) {

return this.accounts.findOneAuthorized(id, req.user);

}

}

src/accounts/accounts.module.ts

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

import { MongooseModule } from '@nestjs/mongoose';

import { AccountsController } from './accounts.controller';

import { AccountsService } from './accounts.service';

import { Account, AccountSchema } from './schemas/account.schema';

@Module({

imports: [MongooseModule.forFeature([{ name: Account.name, schema: AccountSchema }])],

controllers: [AccountsController],

providers: [AccountsService],

})

export class AccountsModule {}

**7) App module wiring**

src/app.module.ts

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

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

import { MongooseModule } from '@nestjs/mongoose';

import { AuthModule } from './auth/auth.module';

import { UsersModule } from './users/users.module';

import { AccountsModule } from './accounts/accounts.module';

@Module({

imports: [

ConfigModule.forRoot({ isGlobal: true }),

MongooseModule.forRoot(process.env.MONGO\_URL ?? 'mongodb://localhost:27017/boeing-banking'),

UsersModule,

AuthModule,

AccountsModule,

],

})

export class AppModule {}

src/main.ts

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

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

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

async function bootstrap() {

const app = await NestFactory.create(AppModule);

app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true, transform: true }));

await app.listen(process.env.PORT ?? 3000);

}

bootstrap();

**8) Quick test flow (cURL)**

1. Register:

curl -s POST http://localhost:3000/users/register \

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

-d '{"email":"a@b.com","password":"P@ssw0rd!","roles":["ADMIN"]}'

1. Login:

curl -s POST http://localhost:3000/auth/login \

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

-d '{"email":"a@b.com","password":"P@ssw0rd!"}'

# => { "access\_token": "..." }

1. Create account (authorized):

TOKEN=... # paste the JWT from login

curl -s POST http://localhost:3000/accounts \

-H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \

-d '{"accountNo":9000000001,"runningTotal":5000,"openingDate":"2023-01-01T00:00:00.000Z"}'

1. List my accounts:

curl -s GET http://localhost:3000/accounts/me -H "Authorization: Bearer $TOKEN"