**1) Schemas (Mongoose discriminators)**

// src/accounts/schemas/account.schema.ts

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

import { HydratedDocument } from 'mongoose';

export type AccountDocument = HydratedDocument<Account>;

export enum AccountType {

SAVINGS = 'savings',

CURRENT = 'current',

}

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

export class Account {

// Discriminator key lives here as a simple prop

@Prop({ required: true, enum: Object.values(AccountType) })

type: AccountType;

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

accountNo: number;

@Prop({ required: true, default: 0 })

balance: number;

@Prop({ required: true })

holderName: string;

}

export const AccountSchema = SchemaFactory.createForClass(Account);

// Important: tell Mongoose which field is the discriminator key

AccountSchema.set('discriminatorKey', 'type');

// -------- Savings --------

@Schema()

export class SavingsAccount extends Account {

@Prop({ required: true })

interestRate: number; // e.g., 4.5 means 4.5%

}

export const SavingsAccountSchema =

SchemaFactory.createForClass(SavingsAccount);

// -------- Current --------

@Schema()

export class CurrentAccount extends Account {

@Prop({ required: true })

overdraftLimit: number; // e.g., 50000

}

export const CurrentAccountSchema =

SchemaFactory.createForClass(CurrentAccount);

**2) DTOs**

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

import { IsEnum, IsNotEmpty, IsNumber, IsOptional, Min, ValidateIf } from 'class-validator';

import { ApiProperty, ApiPropertyOptional } from '@nestjs/swagger';

import { AccountType } from '../schemas/account.schema';

export class CreateAccountDto {

@ApiProperty({ enum: AccountType })

@IsEnum(AccountType)

type: AccountType;

@ApiProperty({ example: 1234567890 })

@IsNumber()

@Min(1)

accountNo: number;

@ApiProperty({ example: 1000 })

@IsNumber()

balance: number;

@ApiProperty({ example: 'John Doe' })

@IsNotEmpty()

holderName: string;

// Savings-only field

@ApiPropertyOptional({ example: 4.5, description: 'Required when type = savings' })

@ValidateIf(o => o.type === AccountType.SAVINGS)

@IsNumber()

@Min(0)

interestRate?: number;

// Current-only field

@ApiPropertyOptional({ example: 50000, description: 'Required when type = current' })

@ValidateIf(o => o.type === AccountType.CURRENT)

@IsNumber()

@Min(0)

overdraftLimit?: number;

}

// src/accounts/dtos/update-account.dto.ts

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

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

import { ApiPropertyOptional } from '@nestjs/swagger';

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

export class UpdateAccountDto extends PartialType(CreateAccountDto) {

@ApiPropertyOptional({ example: 1500 })

@IsOptional()

@IsNumber()

@Min(0)

balance?: number;

}

// src/accounts/dtos/account-response.dto.ts

import { ApiProperty, ApiPropertyOptional } from '@nestjs/swagger';

import { AccountType } from '../schemas/account.schema';

export class AccountResponseDto {

@ApiProperty() id: string;

@ApiProperty({ enum: AccountType }) type: AccountType;

@ApiProperty() accountNo: number;

@ApiProperty() balance: number;

@ApiProperty() holderName: string;

@ApiPropertyOptional() interestRate?: number; // for savings

@ApiPropertyOptional() overdraftLimit?: number; // for current

@ApiProperty() createdAt: string;

@ApiProperty() updatedAt: string;

}

**3) Module (register discriminators)**

// src/accounts/accounts.module.ts

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

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

import {

Account, AccountSchema,

SavingsAccount, SavingsAccountSchema,

CurrentAccount, CurrentAccountSchema,

} from './schemas/account.schema';

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

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

@Module({

imports: [

MongooseModule.forFeatureAsync([

{

name: Account.name,

useFactory: () => {

const schema = AccountSchema;

schema.set('discriminatorKey', 'type');

return schema;

},

discriminators: [

{ name: SavingsAccount.name, schema: SavingsAccountSchema },

{ name: CurrentAccount.name, schema: CurrentAccountSchema },

],

},

]),

],

controllers: [AccountsController],

providers: [AccountsService],

exports: [AccountsService],

})

export class AccountsModule {}

**4) Service (CRUD; create respects type automatically)**

// src/accounts/accounts.service.ts

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

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

import { Model, FilterQuery } from 'mongoose';

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

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

import { UpdateAccountDto } from './dtos/update-account.dto';

import { AccountResponseDto } from './dtos/account-response.dto';

@Injectable()

export class AccountsService {

constructor(

@InjectModel(Account.name) private readonly accountModel: Model<AccountDocument>,

) {}

private toDto(doc: AccountDocument): AccountResponseDto {

const anyDoc = doc.toObject({ virtuals: false }) as any;

return {

id: doc.\_id.toString(),

type: anyDoc.type,

accountNo: anyDoc.accountNo,

balance: anyDoc.balance,

holderName: anyDoc.holderName,

interestRate: anyDoc.interestRate,

overdraftLimit: anyDoc.overdraftLimit,

createdAt: (anyDoc.createdAt ?? doc.createdAt)?.toISOString(),

updatedAt: (anyDoc.updatedAt ?? doc.updatedAt)?.toISOString(),

};

}

async create(dto: CreateAccountDto): Promise<AccountResponseDto> {

// If discriminators are registered, passing 'type' + fields is enough.

const created = await this.accountModel.create(dto as any);

return this.toDto(created);

}

async findAll(type?: AccountType): Promise<AccountResponseDto[]> {

const filter: FilterQuery<Account> = {};

if (type) filter.type = type;

const docs = await this.accountModel.find(filter).lean(false).exec();

return docs.map((d) => this.toDto(d));

}

async findOne(id: string): Promise<AccountResponseDto> {

const doc = await this.accountModel.findById(id).exec();

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

return this.toDto(doc);

}

async update(id: string, dto: UpdateAccountDto): Promise<AccountResponseDto> {

const doc = await this.accountModel.findByIdAndUpdate(id, dto, { new: true }).exec();

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

return this.toDto(doc);

}

async remove(id: string): Promise<void> {

const res = await this.accountModel.findByIdAndDelete(id).exec();

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

}

}

**5) Controller (single endpoint handles both types)**

// src/accounts/accounts.controller.ts

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

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

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

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

import { UpdateAccountDto } from './dtos/update-account.dto';

import { AccountResponseDto } from './dtos/account-response.dto';

import { AccountType } from './schemas/account.schema';

@ApiTags('accounts')

@Controller('accounts')

export class AccountsController {

constructor(private readonly service: AccountsService) {}

@Post()

@ApiOperation({ summary: 'Create an account (savings/current) via discriminator' })

@ApiResponse({ status: 201, type: AccountResponseDto })

create(@Body() dto: CreateAccountDto): Promise<AccountResponseDto> {

return this.service.create(dto);

}

@Get()

@ApiOperation({ summary: 'List accounts, optionally filter by type' })

@ApiQuery({ name: 'type', enum: AccountType, required: false })

@ApiResponse({ status: 200, type: [AccountResponseDto] })

findAll(@Query('type') type?: AccountType): Promise<AccountResponseDto[]> {

return this.service.findAll(type);

}

@Get(':id')

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

@ApiResponse({ status: 200, type: AccountResponseDto })

findOne(@Param('id') id: string): Promise<AccountResponseDto> {

return this.service.findOne(id);

}

@Patch(':id')

@ApiOperation({ summary: 'Update an account by id' })

@ApiResponse({ status: 200, type: AccountResponseDto })

update(@Param('id') id: string, @Body() dto: UpdateAccountDto): Promise<AccountResponseDto> {

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

}

@Delete(':id')

@ApiOperation({ summary: 'Delete an account by id' })

@ApiResponse({ status: 204 })

async remove(@Param('id') id: string): Promise<void> {

await this.service.remove(id);

}

}

**6) App wiring**

In your root module:

// app.module.ts (snippet)

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

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

@Module({

imports: [

MongooseModule.forRoot('mongodb://localhost:27017/bankdb'), // adjust URI

AccountsModule,

],

})

export class AppModule {}

**7) Example requests**

**Create a Savings account**

POST /accounts

{

"type": "savings",

"accountNo": 1234567890,

"balance": 2500,

"holderName": "Alice",

"interestRate": 4.25

}

**Create a Current account**

POST /accounts

{

"type": "current",

"accountNo": 9876543210,

"balance": 10000,

"holderName": "Bob",

"overdraftLimit": 50000

}

**List only savings**

GET /accounts?type=savings