Second Assignment Mongo DB:

Connect your project to mongo DB

You can spin up a docker container with mongo on it if you don’t want to download and install mongo on you machine.

docker run --name mongo -p 27017:27017 -d mongo

install dependencies

npm I mongoose @nestjs/mongoose

mongo connection

Importing the **dynamic** MongooseModule using its **.forRoot()** method

src/app.module.ts

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

import { AppController } from './app.controller';

import { AppService } from './app.service';

import { PlayersModule } from './players/players.module';

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

const MONGO\_URI = 'mongodb://localhost:27017/nest-nba-demo';

const mongooseOptions: MongooseModuleOptions = {

useNewUrlParser: true,

useFindAndModify: false,

useUnifiedTopology: true,

};

@Module({

imports: [MongooseModule.forRoot(MONGO\_URI, mongooseOptions) ,PlayersModule],

controllers: [AppController],

providers: [AppService],

})

export class AppModule {}

Mongoose schema

Change your entity to fit a mongoose schema

src/players/entities/player.entity.ts

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

import { Document } from 'mongoose';

@Schema()

export class Player extends Document {

@Prop()first\_name: string

@Prop()last\_name: string

@Prop()ppg : number

}

export const PlayerSchema = SchemaFactory.createForClass(Player);

import the dynamic MongooseModule using its forFeature method

src/players/players.module.ts

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

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

import { Player, PlayerSchema } from "./entities/player.entity";

import { PlayersController } from './players.controller';

import { PlayersService } from './players.service';

@Module({

imports: [

MongooseModule.forFeature([

{

name: Player.name,

schema: PlayerSchema,

},

]),

],

controllers: [PlayersController],

providers: [PlayersService]

})

export class PlayersModule {}

use the model API in PlayersService

pay attention, the whole service has changed and now it uses the PlayerModel in order to communicate with mongoDB. Moreover change all the ids to string since mongo id is a string

src/players/players.service.ts

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

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

import { Model, QueryOptions } from 'mongoose';

import { PaginationQueryDto } from 'src/common/dto/pagination-query.dto';

import { CreatePlayerDto } from './dto/create-player.dto';

import { UpdatePlayerDto } from './dto/update-player.dto';

import { Player } from './entities/player.entity';

@Injectable()

export class PlayersService {

constructor(@InjectModel(Player.name) private readonly playerModel: Model<Player>){}

async getAllPlayers(dto: PaginationQueryDto): Promise<Player[]>{

const { offset, limit } = dto;

return await this.playerModel.find().skip(offset).limit(limit);

}

async getPlayerById(id:string): Promise<Player> {

const player = await this.playerModel.findById(id).select('-\_id');

if(!player) {

throw new NotFoundException(`player not found id : ${id}`);

}

return player;

}

async createPlayer(createPlayerDto: CreatePlayerDto): Promise<Player>{

return await this.playerModel.create(createPlayerDto)

}

async updatePlayer(id:string, updatePlayerDto: UpdatePlayerDto): Promise<Player> {

const options: QueryOptions = {new: true, upsert: true};

const player = await this.playerModel.findByIdAndUpdate(id, updatePlayerDto, options)

.select('-\_\_v')

return player;

}

async removePlayer(id:string): Promise<Player>{

return await this.playerModel.findByIdAndRemove(id);

}

}

Update the players controller

Change all the function to be async functions and change all ids to be a string

src/players/players.controller.ts

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

import { CreatePlayerDto } from './dto/create-player.dto';

import { PlayersService } from './players.service';

import { UpdatePlayerDto } from './dto/update-player.dto';

import { Player } from './entities/player.entity';

import { IdvalidationPipe } from './pipes/idvalidation.pipe';

import { PaginationQueryDto } from 'src/common/dto/pagination-query.dto';

import { MongoIdPipe } from 'src/common/pipes/mongo-id.pipe';

@Controller('players')

export class PlayersController {

constructor(private playersService: PlayersService){}

@Get()

async getAllPlayers(@Query() dto: PaginationQueryDto): Promise<Player[]>{

return await this.playersService.getAllPlayers(dto)

}await

@Get(':id')

async getPlayerById(@Param('id', MongoIdPipe) id: string): Promise<Player>{

return await this.playersService.getPlayerById(id)

}

@Post()

async createPlayer(@Body() createPlayerDto: CreatePlayerDto): Promise<Player>{

return await this.playersService.createPlayer(createPlayerDto)

}

@Put(':id')

async updatePlayer(

@Param('id', MongoIdPipe) id: string,

@Body() updatePlayerDto: UpdatePlayerDto

){

return await this.playersService.updatePlayer(id, updatePlayerDto)

}

@Delete(':id')

async removePlayer(

@Param('id', MongoIdPipe) id: string,

){

return await this.playersService.removePlayer(id)

}

}

Pagination:

Add pagination to the project – when we will get all player we could set an offset and a limit

Nest g class common/dto/pagination-query.dto –no-spec

src/common/dto/pagination-query.dto.ts

import { IsOptional, IsPositive } from 'class-validator';

export class PaginationQueryDto {

@IsOptional()

@IsPositive()

limit: number;

@IsOptional()

@IsPositive()

offset: number;

}

Since query string parameters are always strings, we need to convert them to numbers

To enable this conversion, we need to set it in the validationPipe

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({

transform: true,

transformOptions: {

enableImplicitConversion: true,

}

}));

await app.listen(3000);

}

bootstrap();

make sure you update the getAllPlayers method in the controller

src/players/players.controller.ts

@Get()

async getAllPlayers(@Query() dto: PaginationQueryDto): Promise<Player[]>{

return await this.playersService.getAllPlayers(dto)

}

And the service

async getAllPlayers(dto: PaginationQueryDto): Promise<Player[]>{

const { offset, limit } = dto;

return await this.playerModel.find().skip(offset).limit(limit);

}

**TEST!**

Third assignment - Mongo relationships:

In order to achieve relationships between documents in mongo DB we have 2 ways. First way is to embed a sub document in the schema (Document). That means that in the document itself we will have all the data about another document. A document in a document. This approach is faster compared to the next approach; however, it has to be only one sided. If we embed both of the documents, we can create a circular reference. Second way is to reference by id only. And then when we pull the data we can “populate” this filed by calling the populate method and reference the filed name.

First create another module, controller and service

nest g mo teams –no-spec

nest g co teams –no-spec

nest g s teams –no-spec

create a team entity and add an array of players to the team.

src/teams/entities/team.entity.ts

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

import { Document, Types } from 'mongoose';

import { Player, PlayerSchema } from '../../players/entities/player.entity';

@Schema({timestamps: true})

export class Team extends Document {

@Prop() name: string;

@Prop() players\_amount: number;

//Option 1 - embed sub document

@Prop({PlayerSchema}) players:Player[];

// Option 2- reference id only

// @Prop({type: Types.ObjectId, ref: 'Player' }) players:Player[];

}

export const TeamSchema = SchemaFactory.createForClass(Team);

update the player schema

src/players/entities/player.entity.ts

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

import { Document, Types } from 'mongoose';

import { Team } from 'src/teams/entities/team.entity';

@Schema({ timestamps: true })

@Schema()

export class Player extends Document {

@Prop()first\_name: string

@Prop()last\_name: string

@Prop()ppg : number

// reference by id only

@Prop({type: Types.ObjectId, ref: Team.name}) team: Team;

}

export const PlayerSchema = SchemaFactory.createForClass(Player);

create a teams dto

src/teams/dto/create-team.dto.ts

import { IsNotEmpty, IsNumber, MinLength } from "class-validator";

export class CreateTeamDto {

@IsNotEmpty()

@MinLength(3)

readonly name: string

@IsNotEmpty()

@IsNumber()

readonly players\_amount: number

}

src/teams/dto/update-team.dto.ts

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

import { CreateTeamDto } from './create-team.dto';

export class UpdateTeamDto extends PartialType(CreateTeamDto) {}

update the players dto and add a mongo id filed in order to know to which team the player belongs

src/players/dto/create-player.dto.ts

import { IsString, IsNotEmpty, Matches, MinLength, IsNumber } from 'class-validator';

export class CreatePlayerDto{

@IsNotEmpty()

@IsString()

readonly first\_name: string

@IsNotEmpty()

@IsString()

readonly last\_name: string

@IsNotEmpty()

@IsNumber()

readonly ppg : number

@IsNotEmpty({ message: 'team is required' })

@Matches('^[0-9a-fA-F]{24}$', 'ig', {

message: 'team is not a valid id',

})

readonly team: any;

}

Create teams controller which is similar to the players controller

src/teams/teams.controller.ts

import {

Body,

Controller,

Delete,

Get,

Param,

Post,

Put,

Query,

UsePipes,

} from '@nestjs/common';

import { PaginationQueryDto } from 'src/common/dto/pagination-query.dto';

import { MongoIdPipe } from 'src/common/pipes/mongo-id.pipe';

import { CreateTeamDto } from './dto/create-team.dto';

import { UpdateTeamDto } from './dto/update-team.dto';

import { Team } from './entities/team.entity';

import { TeamsService } from './teams.service';

@Controller('teams')

export class TeamsController {

constructor(private readonly teamsService: TeamsService) {}

@Get()

async getAllteams(@Query() dto: PaginationQueryDto): Promise<Team[]> {

return await this.teamsService.getAllTeams(dto);

}

@Get(':id')

@UsePipes(MongoIdPipe)

async getteamByID(@Param('id') id: string): Promise<Team> {

return await this.teamsService.getTeamById(id);

}

@Post()

async createteam(@Body() dto: CreateTeamDto): Promise<Team> {

return await this.teamsService.createTeam(dto);

}

@Put(':id')

async updateteam(

@Param('id', MongoIdPipe) id: string,

@Body() dto: UpdateTeamDto,

): Promise<Team> {

return await this.teamsService.updateTeam(id, dto);

}

@Delete(':id')

@UsePipes(MongoIdPipe)

async removeteam(@Param('id') id: string): Promise<Team> {

return await this.teamsService.removeTeam(id);

}

}

Update the player service

src/players/players.service.ts

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

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

import { Model, QueryOptions } from 'mongoose';

import { PaginationQueryDto } from 'src/common/dto/pagination-query.dto';

import { Team } from 'src/teams/entities/team.entity';

import { CreatePlayerDto } from './dto/create-player.dto';

import { UpdatePlayerDto } from './dto/update-player.dto';

import { Player } from './entities/player.entity';

@Injectable()

export class PlayersService {

constructor(

@InjectModel(Player.name) private readonly playerModel: Model<Player>,

@InjectModel(Team.name) private readonly teamModel: Model<Team>

){}

async getAllPlayers(dto: PaginationQueryDto): Promise<Player[]>{

const { offset, limit } = dto;

return await this.playerModel.find().skip(offset).limit(limit).populate('team');

// .populate({ path: 'team', select: '-players' }); //exclude redundent data

}

async getPlayerById(id:string): Promise<Player> {

const player = await this.playerModel.findById(id).select('-\_id');

if(!player) {

throw new NotFoundException(`player not found id : ${id}`);

}

return player;

}

async createPlayer(createPlayerDto: CreatePlayerDto): Promise<Player>{

const player = await this.playerModel.create(createPlayerDto)

const team = await this.teamModel.findById(createPlayerDto.team)

team.players.push(player);

await team.save();

return player;

}

async updatePlayer(id:string, updatePlayerDto: UpdatePlayerDto): Promise<Player> {

const options: QueryOptions = {new: true, upsert: true};

const player = await this.playerModel.findByIdAndUpdate(id, updatePlayerDto, options)

.select('-\_\_v')

return player;

}

async removePlayer(id:string): Promise<Player>{

return await this.playerModel.findByIdAndRemove(id);

}

}

Create team service

src/teams/teams.service.ts

import { Player } from './../players/entities/player.entity';

import { Team } from 'src/teams/entities/team.entity';

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

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

import { Model, QueryOptions } from 'mongoose';

import { PaginationQueryDto } from 'src/common/dto/pagination-query.dto';

import { CreateTeamDto } from './dto/create-team.dto';

import { UpdateTeamDto } from './dto/update-team.dto';

@Injectable()

export class TeamsService {

constructor(

@InjectModel(Team.name) private readonly teamModel: Model<Team>,

@InjectModel(Player.name) private readonly playerModel: Model<Player>

){}

async getAllTeams(dto: PaginationQueryDto): Promise<Team[]>{

const { offset, limit } = dto;

return await this.teamModel.find().skip(offset).limit(limit)

}

async getTeamById(id:string): Promise<Team> {

const team = await this.teamModel.findById(id).select('-\_id');

if(!team) {

throw new NotFoundException(`team not found id : ${id}`);

}

return team;

}

async createTeam(createTeamDto: CreateTeamDto): Promise<Team>{

return await this.teamModel.create(createTeamDto)

}

async updateTeam(id:string, updateTeamDto: UpdateTeamDto): Promise<Team> {

const options: QueryOptions = {new: true, upsert: true};

const team = await this.teamModel.findByIdAndUpdate(id, updateTeamDto, options)

.select('-\_\_v')

return team;

}

async removeTeam(id:string): Promise<Team>{

const team = await this.teamModel.findByIdAndRemove(id);

if(!team){

throw new NotFoundException(`team not found ${id}`);

}

team.players.forEach(async (player) => {

await this.playerModel.findByIdAndRemove(player.id);

})

return team;

}

}

Pay attention that in the remove we need to remove all players from the team as well.

Update modules

Finally, since we're using both Mongoose models in both services, we need to make sure we're including both Mongoose models in both our own modules.

src/players/players.module.ts

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

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

import { Player, PlayerSchema } from "./entities/player.entity";

import { PlayersController } from './players.controller';

import { PlayersService } from './players.service';

import { Team, TeamSchema } from 'src/teams/entities/team.entity';

@Module({

imports: [

MongooseModule.forFeature([

{

name: Player.name,

schema: PlayerSchema,

},

{

name: Team.name,

schema: TeamSchema

},

]),

],

controllers: [PlayersController],

providers: [PlayersService]

})

export class PlayersModule {}

src/teams/teams.module.ts

import { Player } from './../players/entities/player.entity';

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

import { TeamsController } from './teams.controller';

import { TeamsService } from './teams.service';

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

import { PlayerSchema } from '../players/entities/player.entity';

import { Team, TeamSchema } from './entities/team.entity';

@Module({

imports:[

MongooseModule.forFeature([

{

name: Player.name,

schema: PlayerSchema

},

{

name: Team.name,

schema: TeamSchema

},

])

],

controllers: [TeamsController],

providers: [TeamsService]

})

export class TeamsModule {}

**Test!**