**Day 1 Recap:**

1. **Fundamentals of Nest**
   1. **What is Nest JS**
   2. **Why Nest**
   3. **What are the advantages and disadvantages of Nest JS**
2. **Nest JS Core Components/ Nest JS Architecture**
   1. **Modules -> Module based architectures/ Components**
      1. **nest generate module modulename**
         1. **it’s a classs, defined with the @Module decorder**
            1. **to inform about the class**

**imports:[]**

**modules:[]**

**providers:[]**

* 1. **Controllers**
     1. **Route handler to handle all incoming request and handle respone**
     2. **It also a class, defined with @controller decorder**
        1. **Contains a methods , each methods has to decorde with http methods**
           1. **@GET, @PUT**
     3. **If we receive the request, request may contains data also**
        1. **We can receive data in methods by using some decorder**
           1. **@Param(“id”) -> data via the url parameters**
           2. **@Query() -> data cia url queries**
           3. **@Body() -> body paramerters**
     4. **Create controller**
        1. **Nest g controller controllername**
  2. **Providers (services)**
     1. **Providers are like a services**
     2. **Define some business logic to interact with data storage / or some extra functions**
     3. **Are also class, defined with @Injectable()**
     4. **Create a services**
        1. **Nest g service servicename**

**Create a Nest Project:**

* **Install the nest CLI**
  + **Npm I -g @nestJS/cli**
* **Create a nest project**
  + **Nest new projectname**
* **Create a module, controller, service**
* **Create all compontes in a single command**
  + **Nest g resource resourcename**
    - **Create controller, module, services, dto**

**CRUD Operation :**

* **Based on List**
  + **Store in list**
  + **Reterive from list**
  + **Update in list**
  + **Delete in list**
* **Model are created**
  + **Defined class / interface**
  + **Defined dto**
* **CRUD Based on JSON Files**
  + **Store in list**
  + **Reterive from list**
  + **Update in list**
  + **Delete in list**
* **Model with Id**
  + **Generate unique values based on UUID / Guid**
    - **Npm I uuid (npm package)**
      * **Uuid()**
* **Interact with JSOn Files**
  + **Fetch methods**
    - **Fetch(url, {**

**Method, post**

**})**

* + **Axios (npm package)**
    - **Axios.get ,post, put**

**Data Validation:**

* **Validate your incoming data, from url receiving some data, validate**
  + **Used some pipes**
    - **Validate data, and transform**
      * **Int**
      * **Float**
      * **Bool**
      * **Uuid**
  + **Some package (class-validator, class-transformer)**
    - **In my model, dto**
* **Cread Operation Based on MongoDB**

**How to Authenticat a application , validation creadition**

**How to autherozation , validation access**

**Web API, how to apply this**

* **Route authentication**

**API Endpoints, who can access endpoint**

**Guard our Route**

**AUthGuard**

**Once login is sucees, the controller , services will return token**

**Basically token will contains encrupted form of user information(email, user id, name, role) those encryption infor will keep enable , disabled some amount**

**Authorize ur endpoint ,**

**JWT -> JSON Web Token**

**JWT Structure:**

* **Header -> type, hasing, algorithm**
* **Payload -> infor about user (claims)**
* **Signature -> encoded header , encode paya, signed**

**Some Packages:**

1. **Passport**
2. **Passport-jwt**
3. **@nestjs/jwt**
4. **@nestjs/passport**
5. **@types/passport-jwt**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# How to Work With Database:

* In Nest Js , how to connect database
  + Typeorm -> @nestjs/typeorm
    - We can use SQL datbaee/ nosql datbases
    - Repositiry pattern
      * **Way to interact with database**
  + Mangoose ->@nestjs/mongoose
    - Service directly
  + Prisma -> @nestjs/primsa

SQL VS NOSQL

SQL-> structure query lanfuay

NOSQL -> Non structure query language

* Predefined methods can use for all operations
* Course.Insert()
* Coust.Insermany()
* Course.Find()
* Update()
* Delete()

SQL -> Tables -> row / coloums

* Many tables, reterive data,complex query

Nosql db uses the document Model, we can collect all releational data’s in a document

JSOn:

Course:

Id, name , ……., particpnats:[]

JSON Format

Mongo DB:

Mongoose:

<https://mongoosejs.com/docs/queries.html>

# Demo : Working With Mongoose DB – Authentication and Authorization using JWT Token

1. Crete a nest project named as cms-mongoodb
   1. Nest new cms-mongoodb
2. Step 2:
   1. Install the mongo depencies
      1. Npm I @nestjs/mongoose
      2. Npm I mongoose
      3. Npm I @nestjs/config
      4. For unique intstall the uuid
         1. Npm install uuid
3. Create a .env file in tha application
   1. Add the db connection URI as follows
      1. DB\_URI=mongodb://localhost:27017/Course-Management-System
4. Add the env configuration inside the app module imports
   * 1. ConfigModule.forRoot({
     2. envFilePath: '.env',
     3. isGlobal: true
     4. })
5. Add the database configuration insuide the app mosule
   1. MongooseModule.forRoot(process.env.DB\_URI),
6. Create the Modules required for our application
   1. nest g module course
   2. nest g controller course --no-spec
   3. nest g service course --no-spec
7. Create a schema for Course Module
   1. Add a schema folder
      1. Add a course.schema.ts file

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

import { v4 as uuidv4 } from 'uuid';

// define enum for course Stayus

export enum CourseStatus {

  STARTED,

  IN\_PROGRESS,

  COMPLETED,

}

// creata a class for Course

@Schema({

  timestamps: true,

  collection: 'courses',

})

export class Course {

  @Prop({

    type: String,

    unique: true,

    default: function genUUID() {

      return uuidv4();

    },

  })

  id: string;

  @Prop({

    type: String,

    unique: true,

  })

  name: string;

  @Prop({

    type: String,

    required: true,

  })

  description: string;

  @Prop({

    type: Number,

  })

  price: number;

  @Prop({

    type: Number,

    required: true,

  })

  duration: number;

  @Prop()

  status: CourseStatus;

}

// create a  schema based on class

export const CourseSchema= SchemaFactory.createForClass(Course);

1. Add a schema in course Module,

imports: [

    MongooseModule.forFeature([{ name: 'Course', schema: CourseSchema }]),

  ],

1. Add the service for interacting with database

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

import mongoose from 'mongoose';

import { Course, CourseStatus } from './schema/course.schema';

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

// services are used to interact with database

@Injectable()

export class CourseService {

  // inject the db Model

  constructor(

    @InjectModel(Course.name)

    private courseModel: mongoose.Model<Course>,

  ) {}

  //  add a new course

  async addCourse(newcourse: Course): Promise<Course> {

    const res = await this.courseModel.create(newcourse);

    return res;

  }

  // get all courses

  async getAllCourses(): Promise<Course[]> {

    const res = await this.courseModel.find();

    return res;

  }

  // get a course by id

  async getCourseById(id: string): Promise<Course> {

    const res = await this.courseModel.findOne({

        id: id

    });

    return res;

  }

  // get all courses by price

  async getCourseByPrice(sprice: number): Promise<Course[]> {

    const res = await this.courseModel.find(

        {

            price:sprice

        }

    )

    return res;

  }

  // update a course status

  // delete a course

  async deleteCourse(id: string): Promise<Course> {

    const res = await this.courseModel.findByIdAndDelete(id);

    return res;

  }

}

1. Create a dto for Model

export class CreateCourseDto{

    name: string;

    description: string;

    price: number;

    duration: number

}

1. Validate the dto, by using validator and transform data from one type to another use class transformer
   1. @nestjs/class-validator@0.13.1
   2. npm i @nestjs/class-transformer
2. Create CourseDTO with Validation

import {

  IsNumber,

  IsString,

  MAX,

  Max,

  MaxLength,

  Min,

  MinLength,

} from '@nestjs/class-validator';

export class CreateCourseDto {

  @IsString()

  @MaxLength(10)

  @MinLength(3)

  name: string;

  @IsString()

  description: string;

  @IsNumber()

  price: number;

  @IsNumber()

  duration: number;

}

1. Create the update Course status DTO

import { IsEnum } from "@nestjs/class-validator";

import { CourseStatus } from "../schema/course.schema";

export class UpdateCourseStatusDTO

{

    @IsEnum(CourseStatus)

    status: CourseStatus;

}

1. Create a controller for Course

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

import { CourseService } from './course.service';

import { Course, CourseStatus } from './schema/course.schema';

import { CreateCourseDto } from './dto/create-course';

import { v4 as uuidv4 } from 'uuid';

import { UpdateCourseStatusDTO } from './dto/update-course-status';

@Controller('course')

export class CourseController {

    // add a service a dpencey

    constructor(private courseService: CourseService) {}

    // create a controller to add a new Course

    @Post()

    async addCourse(@Body() newcourse: CreateCourseDto): Promise<Course> {

        const course:Course={

            id:uuidv4(),

            name: newcourse.name,

            description: newcourse.description,

            price: newcourse.price,

            duration: newcourse.duration,

            status: CourseStatus.STARTED

        }

        const res = await this.courseService.addCourse(course);

        return res;

    }

    // Get ALl Cources

    @Get()

    async getAllCourses(): Promise<Course[]> {

        const res = await this.courseService.getAllCourses();

        return res;

    }

    //  Get the Course by Id

    @Get(':id')

    async getCourseById(@Param("id") id: string): Promise<Course> {

        const res = await this.courseService.getCourseById(id);

        return res;

    }

    // Delete the Course by Id

    @Delete(':id')

    async deleteCourse(@Param("id") id: string): Promise<string> {

        const res = await this.courseService.deleteCourse(id);

        return res;

    }

    @Patch(':id')

    async updateCourseStatus(@Param("id") id: string,@Body("status") updateStaus: UpdateCourseStatusDTO): Promise<string> {

        const status=updateStaus.status;

        const res = await this.courseService.updateCourseStatus(id, status);

        return res;

    }

}

1. Test the Course Controlleer in Postman or thender Client

Working with Auth Controller and JWT Token

1. Create a Auth Module, Controller, service, schme
   1. Create a User Schema inside the Auth Folder , Create a schema folder and define the user Schema
2. //  create a user Model
3. import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
4. export const enum UserRole {
5. ADMIN = 'ADMIN',
6. USER = 'USER',
7. MANAGER = 'MANAGER',
8. }
9. @Schema(
10. {
11. timestamps: true
12. }
13. )
14. export class User {
15. @Prop(
16. {
17. type: String,
18. unique: true
19. }
20. )
21. id: string;
23. @Prop(
24. {
25. type: String
26. }
27. )
28. name: string;
29. @Prop(
30. {
31. type: String,
32. unique: true
33. }
34. )
35. email: string;
36. @Prop(
37. {
38. type: String,
39. }
40. )
41. password: string;
42. @Prop()
43. role: UserRole;
44. }
45. export const UserSchema = SchemaFactory.createForClass(User);
    1. Include the User schema inside auth modules,
46. MongooseModule.forFeature([{name: 'User', schema: UserSchema}])
    1. Add the Jwt, passpoet package to generate token Packages For Application
    2. **Npm I Passport**
    3. **Npm I Passport-jwt**
    4. **Npm I @nestjs/jwt**
    5. **Npm I @nestjs/passport**
    6. **Npm I @types/passport-jwt**
47. Add the passport Module in Auth Module
    1. PassportModule.register({defaultStrategy: 'jwt'}),
    2. JwtModule.register({
    3. secret: 'course#$123$ecret',
    4. signOptions: {
    5. expiresIn: 3600
    6. }
    7. }),
    8. Create the Auth Service For Register the new User and Login the User