NestJS Workshop

01 Project Setup

```
$ git checkout 01_cleanup
```

- Install the latest NodeJS https://nodejs.org/en/
- Install nest-cli by running

```
$ npm i -g @nestjs/cli
```

- This tuturial uses `node 12.4.0` and `nest 6.5.0`
- Navigate to a folder and run

```
$ nest new task-manager
```

- CD into `task-manager` directory and run: `nest info`

```
$ cd task-manager
$ nest info
```

- Run: `npm start` and point your browser to `http://localhost:3000/`
- Terminate the process and open the folder in your preferred editor
- VSCode users: Install the `REST Client` extension by `Huachao Mao`
- Explore and explain the initial project generated code
- Lunch the project in watch mode

```
$ npm run start:dev
```

- Delete everything except: `main.ts`, `app.module.ts` and `app.controller.ts`
- Refactor `app.controller.ts` to return a string

app.controller.ts

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

@Controller()
export class AppController {
  @Get()
  getHello(): string {
    return "Hello NestJS";
  }
}
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';

@Module({
  controllers: [AppController],
})
export class AppModule {}
```

02 First feature module - Tasks

```
$ git checkout 02_tasks_module
```

- Explain the concept of a feature module
- Generate the tasks module and controller (optional: no spec flag)

```
$ nest g module tasks
$ nest g controller tasks --no-spec
```

- Show that the tasks module was imported into tha app module
- Show that the controller was declared in tasks module
- Explain the NestJS router
- Implement a ping method to test the new route:

tasks.controller.ts

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

@Controller('tasks')
export class TasksController {

    @Get()
    ping(): string {
        return "Tasks controller alive!"
    }
}
```

- Create a directory named: `api`
- Create a file named: `tasks.http`
- Implement a test for a `GET` on the `tasks` route:

api/tasks.http

```
GET http://localhost:3000/tasks
Accept: application/json
```

- Run the test and make sure you get the string response

03 Tasks Controller and Entities

```
$ git checkout 03_tasks_controller
```

- Create a class that represent a `Task`
- You can use the cli to generate a class

task.entity.ts

```
export class Task {
    id: number;
    title: string;
    description: string;
    created_at: string;
    creator_id: number;
    assigned_to: number;
}
```

- Specify the required api on `tasks.http`

api/tasks.http

```
# Get all tasks
GET http://localhost:3000/tasks
Accept: application/json

###

# Get a single task by id
GET http://localhost:3000/tasks/1
Accept: application/json

###
```

```
# Create a new task
POST http://localhost:3000/tasks
Content-Type: application/json

{}

###

# Update a task by id

PUT http://localhost:3000/tasks/1
Content-Type: application/json

{}

###

# Delete a task by id
DELETE http://localhost:3000/tasks/1

###
```

- Implement the controller with stubs to satisfy the spec
- Run the spec during implementation to make sure it works (split screen)

tasks.controller.ts

```
@Controller('tasks')
export class TasksController {

    @Get()
    allTasks(): Task[] {
        return [];
    }

    @Get(':id')
    getTaskById(@Param('id') id: number): Task {
        return <Task>{id};
    }

    @Post()
    createNewTask(@Body() newTask: Partial<Task>): any{
        return "new task created successful";
    }

    @Put(':id')
    updateTaskById(@Param('id') id: number): any {
        return `task # ${id} updated`;
    }
    @Delete(':id')
    deleteTaskById(@Param('id') id: number): any {
        return `task # ${id} deleted`;
    }
}
```

04 Wiring a database and use an ORM

```
$ git checkout 04_database_with_orm
```

- Explain the concept of an ORM
- Why TypeORM fits well
- Nest can be used with database wiring solutions
- install required dependencies (for local development)

```
# Postgres db must be installed locally
$ npm i @nestjs/typeorm typeorm pg
```

- Import and configure type orm with an `ormconfig.json` file

```
{
  "type": "postgres",
  "host": "localhost",
  "port": 5432,
  "username": "nirkaufman",
  "password": "",
  "database": "tasks_db",
  "entities": ["src/**/*.entity{.ts,.js}"],
  "synchronize": true
}
```

- Decorate the Task class to the database:

task.entity.ts

```
@Entity('tasks')
export class Task {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({length: 128})
    title: string;
    @Column({type: 'text'})
    description: string;

    @Column()
    created_at: string;

    @Column()
    creator_id: number;

    @Column({nullable: true})
    assigned_to: number;
}
```

- Import the typeORM module to the app module

app.module.ts

```
@Module({
  controllers: [AppController],
  imports: [TasksModule, TypeOrmModule.forRoot()],
})
export class AppModule {}
```

- Stop the current npm process and use `npm start` instead. otherwise, the auto loading of entity files won't work

- Import the typeORM module to the `tasks` feature module

tasks.module.ts

```
@Module({
  imports: [TypeOrmModule.forFeature([Task])],
  controllers: [TasksController]
})
export class TasksModule {}
```

05 Implementing the tasks service

```
$ git checkout 05_data_access_layer
```

- Explain the role of the data access layer service
- Explain the repository pattern
- Create a `TaskService` using the `cli`

```
$ nest g service tasks --no-spec
```

- Explore the `Repository` class and the available methods
- Implement the `TaskService`:

tasks.service.ts

```
@Injectable()
export class TasksService {

    constructor(
        @InjectRepository(Task)
        private taskRepository: Repository<Task>
    ) {}

    getAllTasks(): Promise<Task[]> {
       return this.taskRepository.find({})
    }

    getTaskById(id: number): Promise<Task> {
        return this.taskRepository.findOne(id);
    }

    createTask(newTask: Partial<Task>): Promise<Task> {
        const task = this.taskRepository.create(newTask);
        return this.taskRepository.save(task);
    }

    updateTaskById(id: number, updatedTask:Partial<Task>):
Promise<UpdateResult> {
        return this.taskRepository.update(id, updatedTask);
    }

    deleteTaskById(id: number): Promise<DeleteResult> {
        return this.taskRepository.delete(id);
    }
}
```

- Refactor the tasks controller to use this service

tasks.controller.ts

```
@Controller('tasks')
export class TasksController {
    constructor(private tasksService: TasksService) {}

    @Get()
    allTasks(): Promise<Task[]> {
        return this.tasksService.getAllTasks();
    }

    @Get(':id')
    getTaskById(@Param('id') id: number): Promise<Task> {
        return this.tasksService.getTaskById(id);
    }

    @Post()
    createNewTask(@Body() newTask: Partial<Task>): Promise<Task> {
        return this.tasksService.createTask(newTask);
    }

    @Put(':id')
    updateTaskById(@Param('id') id: number, @Body() updatedTask:
Partial<Task>): Promise<UpdateResult>{
        return this.tasksService.updateTaskById(id, updatedTask);
    }

    @Delete(':id')
    deleteTaskById(@Param('id') id: number): any {
        return this.tasksService.deleteTaskById(id);
    }
}
```

- Test the API using the `tasks.http` file

```
# Get all tasks
GET http://localhost:3000/tasks
Accept: application/json

###

# Get a single task by id
GET http://localhost:3000/tasks/7
Accept: application/json

###

# Create a new task
POST http://localhost:3000/tasks
Content-Type: application/json

{
  "title": "new task",
  "description": "new task description",
  "created_at": "12-1-2019",
  "creator_id": "1"
}

###
```

```
# Update a task by id

PUT http://localhost:3000/tasks/7
Content-Type: application/json

{
"title": "new title for 7"
}

###

# Delete a task by id
DELETE http://localhost:3000/tasks/7

###
```

05 Authorization guard

```
$ git checkout 06_authorization_guard
```

- Generate an Auth guard using nest cli (skip tests)

```
$ nest g guard auth --no-spec
```

- Clean th provider so it will return a simple boolean

auth.guard.ts

```
@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean  {
    return false;
  }
}
```

- Use this guard to protect the tasks controller

tasks.controller.ts

```
@Controller('tasks')
@UseGuards(AuthGuard)
export class TasksController {}
```

- test the route to get a `403 Forbidden` error
- Move the guard to protect just the `createPost` method and test it

tasks.controller.ts

```
@Post()
@UseGuards(AuthGuard)
createNewTask(@Body() newTask: Partial<Task>): Promise<Task> {}
```

- show the concept of global guard

main.ts

```
app.useGlobalGuards(new AuthGuard());
```

- Explain the `ExecutionContext` data

auth.guard.ts

```
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean  {
    console.log('handler', context.getHandler());
    console.log('request', context.switchToHttp().getRequest());
    return false;
  }
}
```

- Move the guard back to `tasks.controller.ts`:

tasks.controller.ts

```
@Post()
@UseGuards(AuthGuard)
createNewTask(@Body() newTask: Partial<Task>): Promise<Task> {}
```

- Refactor the `AuthGuard` to validate token:

auth.guard.ts

```
@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean  {
    const token = context.switchToHttp().getRequest().headers.token;
    return token === '123456';
    }
}
```

- Test it

api/tasks.http

```
# Create a new task
POST http://localhost:3000/tasks
Content-Type: application/json
token: 1234567
```

06 Exception filters

```
$ git checkout 07_exception_filters
```

- Generate a filter using the cli

```
$ nest g filter auth --no-spec
```

- Implement custom exception filter for Forbidden actions

auth.filter.ts

```
@Catch(ForbiddenException)
export class AuthFilter implements ExceptionFilter {
    catch(exception: ForbiddenException, host: ArgumentsHost) {
        const ctx = host.switchToHttp();
        const response = ctx.getResponse();
        const request = ctx.getRequest();
        const status = exception.getStatus();

        response
            .status(status)
            .json({
                statusCode: status,
                timestamp: new Date().toISOString(),
                path: request.url,
                message: 'Protected method'
            })
    }
}
```

- Use this filter in tasks controller for the create method

tasks.controller.ts

```
@Post()
@UseGuards(AuthGuard)
@UseFilters(AuthFilter)
createNewTask(@Body() newTask: Partial<Task>): Promise<Task> {
    return this.tasksService.createTask(newTask);
}
```

- Use this filter in tasks controller scope **tasks.controller.ts**

```
@Controller('tasks')
@UseFilters(AuthFilter)
export class TasksController {}
```

- Can be used as global filters **main.ts**

```
app.useGlobalFilters()
```

- Explore other types of exception

```
BadRequestException
UnauthorizedException
NotFoundException
ForbiddenException
NotAcceptableException
RequestTimeoutException
ConflictException
GoneException
PayloadTooLargeException
UnsupportedMediaTypeException
UnprocessableEntityException
InternalServerErrorException
NotImplementedException
BadGatewayException
ServiceUnavailableException
GatewayTimeoutException
```

07 Pipe

```
$ git checkout 08_pipes
```

- Lets create another table for users.
- Generate a feature set for users: entity, service, controller and module

```
$ nest g module users
$ nest g controller users --no-spec
$ nest g service users --no-spec
$ nest g class users/user --no-spec
```

- Rename the `User` class and map it to the database

user.entity.ts

```
@Entity('users')
export class User {

    @PrimaryGeneratedColumn()
    id: number;

    @Column()
    username: string;

    @Column()
    email: string;
}
```

- Import typeorm module to users.module and wire it up

users.module.ts

```
@Module({
  imports: [TypeOrmModule.forFeature([User])],
  controllers: [UsersController],
  providers: [UsersService]
})
export class UsersModule {}
```

- Implement the user service (similer to tasks service. can we create an interface?)

users.service.ts

```
@Injectable()
export class UsersService {

    constructor(
        @InjectRepository(User)
        private userRepository: Repository<User>
    ) {}

    getAllUsers(): Promise<User[]> {
        return this.userRepository.find({})
    }

    createUser(newUser: Partial<User>): Promise<User> {
        const user = this.userRepository.create(newUser);
        return this.userRepository.save(user);
    }
}
```

- Finish the users controller

users.controller.ts

```
@Controller('users')
export class UsersController {
    constructor(private userService: UsersService) {}

    @Get()
    allUsers(): Promise<User[]> {
        return this.userService.getAllUsers();
    }

    @Post()
    createNewUser(@Body() newUser: Partial<User>): Promise<User> {
        return this.userService.createUser(newUser);
    }
}
```

- Test the new end points with an .http file

api/users.http

```
GET http://localhost:3000/users
Accept: application/json

###

POST http://localhost:3000/users
Content-Type: application/json

{
  "username": "nirkaufman",
  "email": "nir@500tech.com"
}

###
```

- Generate your first pipe with nest cli

```
$ nest g pipe Validate
```

- Use the pipe on the CreateUser method (comment out the actual creation)

users.controller.ts

```
    @Post()
    @UsePipes(ValidatePipe)
    createNewUser(@Body() newUser: Partial<User>): any {
        console.log(newUser);
    }
```

- Explore the value and metadata arguments og the pipe

validate.pipe.ts

```
@Injectable()
export class ValidatePipe implements PipeTransform {
  transform(value: any, metadata: ArgumentMetadata) {
    console.log('value', value);
    console.log('metadata', metadata);

    return value['email'];
  }
}
```

- Explain the potential and real-world usage
- Bring the original code back to `users.controller` **users.controller.ts**

```
@Post()
@UsePipes(ValidatePipe)
createNewUser(@Body() newUser: Partial<User>): Promise<User> {
    return this.userService.createUser(newUser);
}
```

- Make the email unique on the user entity

user.entity.ts

```
@Column({unique: true})
email: string;
```

- Explain and install the `class-validator` and `class-transformer` package

```
$ npm i class-validator class-transformer
```

- Explain the concept of a DTO (data transfer object)
- Create a `CreateUserDTO` under `users`

```
$ nest g class users/CreateUserDTO --no-spec
```

- Use the `class validator` package to annotate this class

create-user-dto.ts

```
import {IsEmail, IsNotEmpty} from 'class-validator';

export class CreateUserDto {
    @IsNotEmpty()
    username: string;

    @IsEmail()
    email: string;
}
```

- Use the DTO in `users.controller` with the built-in ValidationPipe

users.controller.ts

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

@Post()
@UsePipes(ValidationPipe)
createNewUser(@Body() newUser: CreateUserDto): Promise<User> {
    return this.userService.createUser(newUser);
}
```

- Test the api with an empty username or invalid email and inspect the response

08 Interceptors

```
$ git checkout 09_interceptors
```

- Generate an interceptor

```
nest g interceptor log --no-spec
```

- Bind the interceptor as a global interceptor.
- We need to instantiate it because we are out of a module context

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalInterceptors(new LogInterceptor());
  await app.listen(3000);
}
```

- Implement simple logger

```
@Injectable()
export class LogInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any>
{
    const requestInfo = {
      time: new Date().toLocaleTimeString(),
      controller: context.getClass().name,
      method: context.getHandler().name
    };

    console.log(requestInfo);
    return next.handle();
  }
}
```

- Use the `users.http` file to dispatch requests and watch the log

09 Swagger

```
$ git checkout 10_swagger
```

-install the required dependencies

```
$ npm install --save @nestjs/swagger swagger-ui-express
```

- Setup in `main.ts`

main.ts

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalInterceptors(new LogInterceptor());

  const options = new DocumentBuilder()
      .setTitle('Tasks Example')
      .setDescription('The Task manager API description')
      .setVersion('1.0')
      .addTag('Tasks')
      .build();
  const document = SwaggerModule.createDocument(app, options);
  SwaggerModule.setup('api', app, document);

  await app.listen(3000);
}
```