**NestJs Authentification and authorization**

1. **Introduction**

Authentication and authorization are crucial aspects of building secure applications. In NestJS, authentication refers to verifying the identity of users, while authorization involves granting or denying access to resources or functionalities based on the authenticated user's permissions.

Here's an overview of authentication and authorization in NestJS:

**Authentication:**

**Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement**

1. **Authentication Strategies:**
   * NestJS supports various authentication strategies, such as JWT (JSON Web Tokens), OAuth, Passport, and more.
   * These strategies can be implemented using middleware, guards, or interceptors to authenticate incoming requests.
2. **Passport Module:**
   * NestJS integrates well with the popular **passport** module for implementing authentication strategies.
   * Passport strategies can be used to authenticate requests against various providers like JWT, OAuth, local username/password, etc.
3. **AuthGuard:**
   * NestJS provides the **AuthGuard** class that can be used to protect routes based on authentication status.
   * It can be extended to implement custom authentication logic.

### Authorization:

1. **Role-Based Access Control (RBAC):**
   * Authorization in NestJS often involves role-based access control, where users are assigned roles with specific permissions.
   * Guards and interceptors can be used to implement RBAC by checking a user's role or permissions before granting access to resources.
2. **Authorization Guards:**
   * NestJS provides **Guards** to implement authorization logic. These guards can be used to check permissions or roles before allowing access to routes or resources.
   * Custom guards can be created to implement specific authorization requirements.
3. **Use of Decorators:**
   * Decorators like **@Roles()** or **@Permissions()** can be created to define roles or permissions on specific endpoints or controllers.
4. This example uses a JWT authentication strategy with a guard (**JwtAuthGuard**) to protect the **profile** endpoint. The **JwtStrategy** class handles JWT validation, and the **JwtAuthGuard** ensures that only authenticated users can access the **profile** route.
5. Overall, NestJS provides a robust framework for implementing authentication and authorization using a variety of strategies, guards, interceptors, and middleware, allowing developers to build secure applications with ease.
6. This example uses a JWT authentication strategy with a guard (**JwtAuthGuard**) to protect the **profile** endpoint. The **JwtStrategy** class handles JWT validation, and the **JwtAuthGuard** ensures that only authenticated users can access the **profile** route.
7. Overall, NestJS provides a robust framework for implementing authentication and authorization using a variety of strategies, guards, interceptors, and middleware, allowing developers to build secure applications with ease.

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

1. **Creating a users resource**

**nest g resource users**

**Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement**

**Une image contenant texte, Police, logiciel, Logiciel multimédia

Description générée automatiquement**

1. **Hashing password**

**Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement**

#### Hashing[#](https://docs.nestjs.com/security/encryption-and-hashing#hashing)

For hashing, we recommend using either the **[bcrypt](https://www.npmjs.com/package/bcrypt" \t "_blank)** or [**argon2**](https://www.npmjs.com/package/argon2) packages. Nest itself does not provide any additional wrappers on top of these modules to avoid introducing unnecessary abstractions (making the learning curve short).

As an example, let's use bcrypt to hash a random password.

First install required packages:

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

1. **Implement Sign-in and Sign-up routes.**

**Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**

**Authentication.service.ts**

import {  
 ConflictException,  
 Injectable,  
 UnauthorizedException,  
} from '@nestjs/common';  
import { InjectRepository } from '@nestjs/typeorm';  
import { User } from '../../users/entities/user.entity';  
import { Repository } from 'typeorm';  
import { HashingService } from '../hashing/hashing.service';  
import { SignUpDto } from './dto/sign-up.dto/sign-up.dto';  
import { SignInDto } from './dto/sign-in.dto/sign-in.dto';  
  
@Injectable()  
export class AuthentificationService {  
 constructor(  
 @InjectRepository(User) private readonly usersRepository: Repository<User>,  
 private readonly hashingService: HashingService,  
 ) {}  
  
 async signUp(signUpDto: SignUpDto) {  
 try {  
 const user = new User();  
 user.email = signUpDto.email;  
 user.password = await this.hashingService.hash(signUpDto.password);  
  
 await this.usersRepository.save(user);  
 } catch (err) {  
 const pgUniqueViolationErrorCode = '23505';  
 if (err.code === pgUniqueViolationErrorCode) {  
 throw new ConflictException();  
 }  
 throw err;  
 }  
 }  
  
 async signIn(signInDto: SignInDto) {  
 const user = await this.usersRepository.findOneBy({  
 email: signInDto.email,  
 });  
 if (!user) {  
 throw new UnauthorizedException('User does not exists');  
 }  
 const isEqual = await this.hashingService.compare(  
 signInDto.password,  
 user.password,  
 );  
 if (!isEqual) {  
 throw new UnauthorizedException('Password does not match');  
 }  
  
 // *TODO :We'll fill this gap in the next lesson* }  
}

**Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**

**Une image contenant texte, Logiciel multimédia, logiciel, Police

Description générée automatiquement**

**Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement**

**Une image contenant texte, Logiciel multimédia, logiciel, Logiciel de graphisme

Description générée automatiquement**