NestJS User Module Documentation

Moien Uddin

December 11, 2025

Contents

1 Project Overview

This project implements a **User Module** in NestJS. Features include:

- User Registration
- User Sign-in (with password hashing)
- User Update
- Password hashing using bcrypt
- DTO validation using class-validator

The database used is PostgreSQL, connected via TypeORM.

2 Database Configuration

Listing 1: Database Connection (app.module.ts)

```
TypeOrmModule.forRoot({
  type: 'postgres',
  host: 'localhost',
  port: 5432,
  username: 'postgres',
  password: 'your_password',
  database: 'your_db',
  entities: [User],
  synchronize: true,
}),
```

Explanation:

- `entities`: connects our `User` entity to the database.
- `synchronize`: automatically creates tables (disable in production).

3 User Entity

Listing 2: User Entity (user.entity.ts)

```
@Entity('users')
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @Column({ unique: true })
  username: string;

  @Column()
  password: string;

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

  @Column({ default: true })
  is_active: boolean;

  @Column()
  first_name: string;

  @Column()
  last_name: string;

  @Column({ type: 'date', nullable: true })
  dob: Date;

  @Column({ nullable: true })
  gender: string;

  @Column({ default: 'user' })
  type: string;

  @CreateDateColumn()
  created_at: Date;

  @UpdateDateColumn()
  updated_at: Date;

  @Column({ nullable: true })
  created_by: string;

  @Column({ nullable: true })
  updated_by: string;
}
```

Explanation:

- `@Entity('users')`: maps the class to a database table named `users`.
- UUID is used for `id` to ensure global uniqueness.
- `is_active` and `type` have default values.

4 DTOs (Data Transfer Objects)

4.1 RegisterDto

Listing 3: RegisterDto (dto/register.dto.ts)

```
export class RegisterDto {
  @IsString()
  username: string;

  @IsString()
  password: string;

  @IsEmail()
  email: string;

  @IsString()
  first_name: string;

  @IsString()
  last_name: string;

  @IsOptional()
  @IsDateString()
  dob?: string;

  @IsOptional()
  @IsString()
  gender?: string;
}
```

Explanation: Uses class-validator decorators to enforce input validation.

4.2 SignInDto

Listing 4: SignInDto (dto/signin.dto.ts)

```
export class SignInDto {
  @IsString()
  username: string;

  @IsString()
  password: string;
}
```

5 User Service

5.1 Sign-in Function

Listing 5: Sign-in Function (user.service.ts)

```
async signIn(data: SignInDto) {
    const user = await this.userRepo.findOne({
      where: { username: data.username },
    });

    if (!user) throw new NotFoundException('User not found');

    const match = await bcrypt.compare(data.password, user.
       password);

    if (!match) throw new UnauthorizedException('Invalid password
       ');

    // Remove password before returning
    const { password, ...result } = user;
    return result;
}
```

Explanation:

- `findOne`: fetches user by username.
- `bcrypt.compare`: compares plaintext password with hashed password.
- `Destructuring`: `const { password, ...result } = user;` removes password from returned object.
- Throws proper exceptions if user not found or password is invalid.

6 Controller Endpoints

- **POST /user/register** : Register a new user
- **POST /user/signin** : Login and receive user object (later JWT)
- **PUT /user/update/:id** : Update user details

7 Testing Instructions

1. Run the application: `npm run start:dev`
2. Use Postman or Insomnia.
3. Register a user via `POST /user/register` with JSON body.
4. Sign-in via `POST /user/signin`.
5. Check that the response does not return the password.

8 Security Considerations

- Passwords are hashed using `bcrypt` before saving to the database.
- Passwords are removed from API responses.
- Input validation prevents malicious data injection.

9 Future Improvements

- Add JWT authentication for secure access.
- Add role-based access control (admin/user).
- Implement email verification and password reset.
- Add Swagger for API documentation.