



Correção do Problema de Autenticação 401

Data: 08/10/2025

Branch: `fix/auth-401-public-decorator`

PR: <https://github.com/nerdrico2025/deliverei-v1/pull/4>



Problema Identificado

Os endpoints de autenticação estavam retornando **401 Unauthorized**:

- `POST /api/auth/login`
- `POST /api/auth/signup`
- `POST /api/auth/refresh`

Enquanto os endpoints públicos (`/api/public/*`) funcionavam perfeitamente.



Análise da Causa Raiz

Arquitetura do Sistema

1. **Guard Global JWT** (`main.ts`):

```
// Guard aplicado globalmente em TODAS as rotas
const reflector = app.get(Reflector);
app.useGlobalGuards(new JwtAuthGuard(reflector));
```

1. **JwtAuthGuard** (`guards/jwt-auth.guard.ts`):

```

@Injectables()
export class JwtAuthGuard extends AuthGuard('jwt') {
  constructor(private reflector: Reflector) {
    super();
  }

  canActivate(context: ExecutionContext) {
    // Verifica se a rota tem decorator @Public()
    const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    if (isPublic) {
      return true; // Permite acesso sem token
    }

    return super.canActivate(context); // Requer token JWT válido
  }
}

```

1. AuthController ANTES da correção (❌ SEM @Public()):

```

@Controller('auth')
export class AuthController {
  @Post('login') // ❌ Sem @Public() - Guard bloqueia!
  async login(@Body() loginDto: LoginDto) {
    return this.authService.login(loginDto);
  }

  @Post('signup') // ❌ Sem @Public() - Guard bloqueia!
  async signup(@Body() signupDto: SignupDto) {
    return this.authService.signup(signupDto);
  }
}

```

O Ciclo Impossível

```

Cliente → POST /api/auth/login (sem token)
↓
JwtAuthGuard verifica: "Tem @Public()?"
↓
NÃO → Verifica token JWT
↓
Token não existe ou inválido
↓
❌ 401 Unauthorized
↓
Cliente nunca consegue obter o token inicial!

```

✓ Solução Implementada

Código Corrigido

```
import { Controller, Post, Body, HttpStatusCode, HttpStatus } from '@nestjs/common';
import { AuthService } from '../auth.service';
import { LoginDto, SignupDto, RefreshTokenDto } from '../dto';
import { Public } from '../../decorators/public.decorator'; // ✓ Importado

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Public() // ✓ Permite acesso sem token
  @Post('login')
  @HttpStatusCode(HttpStatus.OK)
  async login(@Body() loginDto: LoginDto) {
    return this.authService.login(loginDto);
  }

  @Public() // ✓ Permite acesso sem token
  @Post('signup')
  @HttpStatusCode(HttpStatus.CREATED)
  async signup(@Body() signupDto: SignupDto) {
    return this.authService.signup(signupDto);
  }

  @Public() // ✓ Permite renovação sem autenticação
  @Post('refresh')
  @HttpStatusCode(HttpStatus.OK)
  async refresh(@Body() refreshTokenDto: RefreshTokenDto) {
    return this.authService.refreshAccessToken(refreshTokenDto.refreshToken);
  }

  @Post('logout') // ✗ SEM @Public() - Requer autenticação (correto!)
  @HttpStatusCode(HttpStatus.OK)
  async logout(@Body() refreshTokenDto: RefreshTokenDto) {
    return this.authService.logout(refreshTokenDto.refreshToken);
  }
}
```

Decisões de Design

Endpoint	@Public()	Justificativa
POST /auth/login	✓ Sim	Usuário precisa fazer login SEM ter token
POST /auth/signup	✓ Sim	Novo usuário precisa se cadastrar SEM ter token
POST /auth/refresh	✓ Sim	Renovação de token usando refreshToken (não JWT)
POST /auth/logout	✗ Não	Logout deve ser autenticado para invalidar token correto

Testes Realizados

1. Signup - Criação de Usuário

```
curl -X POST http://localhost:3000/api/auth/signup \  
-H "Content-Type: application/json" \  
-d '{  
  "email": "teste@example.com",  
  "senha": "senha123",  
  "nome": "Usuario Teste"  
}'
```

Resultado:

```
{  
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "refreshToken": "058aca36-fc31-488a-bdfb-8ccb043aff22",  
  "user": {  
    "id": "42ccbf3-0fcd-432c-b0a6-f43cd4815c11",  
    "email": "teste@example.com",  
    "nome": "Usuario Teste",  
    "role": "SUPER_ADMIN",  
    "empresaId": null,  
    "empresa": null  
  }  
}
```

✓ Status: 201 Created

2. Login - Autenticação

```
curl -X POST http://localhost:3000/api/auth/login \  
-H "Content-Type: application/json" \  
-d '{  
  "email": "teste@example.com",  
  "senha": "senha123"  
}'
```

Resultado:

```
{  
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
  "refreshToken": "59442029-8e6f-460d-81df-a2f2d96aa7a3",  
  "user": {  
    "id": "42ccbf3-0fcd-432c-b0a6-f43cd4815c11",  
    "email": "teste@example.com",  
    "nome": "Usuario Teste",  
    "role": "SUPER_ADMIN",  
    "empresaId": null,  
    "empresa": null  
  }  
}
```

✓ Status: 200 OK

3. Refresh Token - Renovação

```
curl -X POST http://localhost:3000/api/auth/refresh \  
-H "Content-Type: application/json" \  
-d '{  
  "refreshToken": "59442029-8e6f-460d-81df-a2f2d96aa7a3"  
}'
```

Resultado:

```
{  
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}
```

✓ Status: 200 OK

4. Logout - Sem Token (Deve Falhar)

```
curl -X POST http://localhost:3000/api/auth/logout \  
-H "Content-Type: application/json" \  
-d '{  
  "refreshToken": "59442029-8e6f-460d-81df-a2f2d96aa7a3"  
}'
```

Resultado:

```
{  
  "statusCode": 401,  
  "timestamp": "2025-10-08T17:19:15.406Z",  
  "path": "/api/auth/logout",  
  "error": "UnauthorizedException",  
  "message": "Unauthorized"  
}
```

✓ Status: 401 Unauthorized (comportamento correto!)

5. Logout - Com Token (Deve Funcionar)

```
curl -X POST http://localhost:3000/api/auth/logout \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \  
-d '{  
  "refreshToken": "59442029-8e6f-460d-81df-a2f2d96aa7a3"  
}'
```

Resultado:

```
{
  "message": "Logout realizado com sucesso"
}
```

✅ Status: 200 OK

**Resumo dos Resultados**

Endpoint	Antes	Depois	Status
POST /auth/signup	❌ 401	✅ 201 + JWT	Corrigido
POST /auth/login	❌ 401	✅ 200 + JWT	Corrigido
POST /auth/refresh	❌ 401	✅ 200 + JWT	Corrigido
POST /auth/logout (sem token)	❌ 401	✅ 401	Correto
POST /auth/logout (com token)	❌ 401	✅ 200	Corrigido

**Impacto da Correção****Benefícios**

1. **Autenticação Funcional:** Fluxo completo de login/signup restaurado
2. **Segurança Mantida:** Logout ainda requer autenticação
3. **Compatibilidade:** Não afeta multi-tenancy ou outros módulos
4. **Padrão Correto:** Uso adequado do decorator `@Public()`

**Segurança**

- ✅ Endpoints públicos claramente marcados
- ✅ Guard JWT continua protegendo rotas privadas
- ✅ Logout requer token válido (previne invalidação maliciosa)
- ✅ Validação de dados mantida (DTOs com class-validator)

**Performance**

- ✅ Sem overhead adicional
- ✅ Guard verifica metadata antes de validar JWT
- ✅ Rotas públicas não processam validação JWT desnecessária

Deploy

Comandos Executados

```
# 1. Criar branch
git checkout -b fix/auth-401-public-decorator


# 2. Aplicar correção
# (Edição do auth.controller.ts)

# 3. Commit
git add backend/src/modules/auth/auth.controller.ts
git commit -m "fix: adicionar decorator @Public() nos endpoints de autenticação"

# 4. Push
git push origin fix/auth-401-public-decorator

# 5. Criar PR
# PR #4 criado via GitHub API
```

Pull Request

- **URL:** <https://github.com/nerdrico2025/delivre-v1/pull/4>
- **Status:** Open
- **Título:**  Fix: Corrigir erro 401 nos endpoints de autenticação
- **Pronto para merge:** ☒ Sim

Lições Aprendidas

1. Guards Globais Requerem Atenção

Quando um guard é aplicado globalmente, **TODAS** as rotas são afetadas. É crucial:

- Documentar claramente quais rotas são públicas
- Usar decorators para marcar exceções
- Testar fluxos de autenticação desde o início

2. Decorators são Metadata

O decorator `@Public()` não “desliga” o guard, ele apenas:

```
SetMetadata(IS_PUBLIC_KEY, true)
```

O guard então **lê essa metadata** e decide se valida ou não o JWT.

3. Ordem de Execução

```
Request → Middleware → Guards → Interceptors → Controller → Service
```

O `TenantMiddleware` executa **antes** do `JwtAuthGuard`, então:

- Multi-tenancy funciona independente de autenticação
- Endpoints públicos ainda podem ser tenant-specific

Referências

- **Código Fonte:** `/backend/src/modules/auth/auth.controller.ts`
 - **Guard JWT:** `/backend/src/guards/jwt-auth.guard.ts`
 - **Decorator Public:** `/backend/src/decorators/public.decorator.ts`
 - **Main Bootstrap:** `/backend/src/main.ts`
 - **PR GitHub:** <https://github.com/nerdrico2025/deliverei-v1/pull/4>
-

Checklist de Validação

- [x] Problema identificado e documentado
 - [x] Causa raiz analisada
 - [x] Solução implementada
 - [x] Testes realizados (signup, login, refresh, logout)
 - [x] Código commitado
 - [x] PR criado
 - [x] Documentação atualizada
 - [x] Segurança validada
 - [x] Compatibilidade verificada
-

Status Final:  **PROBLEMA RESOLVIDO**

A autenticação está funcionando corretamente. O PR está pronto para review e merge.