

Báo Cáo Phân Tích Cài Đặt Clerk Authentication trong NestJS

Table of Contents

1. 1. Giới Thiệu	1
1.1. 1.1. Mục Đích Phân Tích	1
1.2. 1.2. Phạm Vi Phân Tích	2
1.3. 1.3. Phương Pháp Phân Tích	2
2. 2. Hiện Trạng Implementation	2
2.1. 2.1. Cấu Trúc Thư Mục	2
2.2. 2.2. AuthModule - Phân Tích Chi Tiết (src/modules/auth/auth.module.ts)	3
2.3. 2.3. ClerkStrategy - Phân Tích Chi Tiết (Không tồn tại)	4
2.4. 2.4. ClerkAuthGuard - Phân Tích Chi Tiết (src/modules/auth/guards/clerk-auth.guard.ts)	5
2.5. 2.5. ClerkModule - Phân Tích Chi Tiết (src/modules/clerk/clerk.module.ts)	6
2.6. 2.6. AppModule - Phân Tích Chi Tiết (src/app.module.ts)	7
3. 3. Tổng Hợp Các Vấn Đề và Đề Xuất	8
4. 4. Kế Hoạch Hành Động Đề Xuất	9
5. 5. Kết Luận	10

Tóm Tắt Điều Hành

Báo cáo này trình bày kết quả phân tích toàn diện việc tích hợp Clerk Authentication trong dự án NestJS TheShoe. Qua việc đối chiếu implementation hiện tại với tài liệu tham khảo [doc/pdf/bao_cao_chi_tiet_clerk.pdf](#) và best practices, chúng tôi đã xác định được 10 điểm cần cải thiện quan trọng để tối ưu hóa kiến trúc xác thực, tăng cường bảo mật và cải thiện khả năng bảo trì hệ thống.

1. 1. Giới Thiệu

1.1. 1.1. Mục Đích Phân Tích

- Đánh giá mức độ tuân thủ của implementation hiện tại với best practices của Clerk và NestJS
- Xác định các điểm yếu trong kiến trúc xác thực hiện tại

- Đề xuất roadmap cụ thể để nâng cấp hệ thống xác thực
- Đảm bảo tính bảo mật và khả năng mở rộng của ứng dụng

1.2. 1.2. Phạm Vi Phân Tích

Thành phần	Mô tả
Module Auth	<code>src/modules/auth/</code> - Module xác thực chính
Module Clerk	<code>src/modules/clerk/</code> - Module tích hợp Clerk
AppModule	<code>src/app.module.ts</code> - Cấu hình toàn cục
Tài liệu tham khảo	<code>doc/pdf/bao_cao_chi_tiet_clerk.pdf</code>

1.3. 1.3. Phương Pháp Phân Tích

1. **Code Review:** Rà soát từng tệp trong module auth và clerk
2. **Architectural Analysis:** So sánh với pattern chuẩn của NestJS Passport
3. **Best Practices Check:** Đối chiếu với tài liệu Clerk chính thức
4. **Security Assessment:** Đánh giá các vấn đề bảo mật tiềm ẩn

2. 2. Hiện Trạng Implementation

2.1. 2.1. Cấu Trúc Thư Mục

<code>src/modules/auth/</code>	
<code> — auth.controller.ts</code>	□ Có sẵn
<code> — auth.module.ts</code>	□□ Cần cập nhật
<code> — auth.service.ts</code>	□□ Vai trò cần xem xét
<code> — decorators/</code>	
<code> — roles.decorator.ts</code>	□ Có sẵn
<code> — guards/</code>	
<code> — clerk-auth.guard.ts</code>	□ Cần refactor
<code> — roles.guard.ts</code>	□ Có sẵn
<code> — strategies/</code>	
<code> — (trống)</code>	□ Thiếu ClerkStrategy
 <code>src/modules/clerk/</code>	
<code> — clerk.controller.ts</code>	□ Có sẵn
<code> — clerk.module.ts</code>	□□ Cần bổ sung Provider cho ClerkClient
<code> — clerk.session.service.ts</code>	□ Có sẵn

2.2. 2.2. AuthModule - Phân Tích Chi Tiết (src/modules/auth/auth.module.ts)

Code Hiện Tại

```
import { Module } from '@nestjs/common';
import { UsersModule } from '../users/users.module';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';

@Module({
  imports: [
    UsersModule,
  ],
  controllers: [AuthController],
  providers: [AuthService],
  exports: [AuthService],
})
export class AuthModule {}
```

Đề Xuất Theo Tài Liệu Tham Khảo (mục 2.4.1)

```
import { Module } from '@nestjs/common';
import { AuthController } from './auth.controller';
import { ClerkStrategy } from './strategies/clerk.strategy'; ①
import { PassportModule } from '@nestjs/passport';           ②
import { UsersModule } from '../users/users.module';
import { ConfigModule } from '@nestjs/config';               ③

@Module({
  controllers: [AuthController],
  imports: [
    PassportModule,           ②
    UsersModule,
    ConfigModule               ③
  ],
  providers: [ClerkStrategy], ①
  exports: [PassportModule],  ②
})
export class AuthModule {}
```

- ① Thiếu `ClerkStrategy` trong `providers`.
- ② Thiếu `PassportModule` trong `imports` và `exports`.
- ③ Thiếu `ConfigModule` trong `imports`.

2.3. 2.3. ClerkStrategy - Phân Tích Chi Tiết (Không tồn tại)

Hiện tại, tệp `src/modules/auth/strategies/clerk.strategy.ts` không tồn tại. Đây là một thiếu sót lớn.

Đề Xuất Theo Tài Liệu Tham Khảo (mục 2.3.2)

```
// src/modules/auth/strategies/clerk.strategy.ts
import { User, verifyToken } from '@clerk/backend'; // Hòc sđ dng ClerkClient tđ
provider
import { Injectable, UnauthorizedException, Inject } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy } from 'passport-custom';
import { UsersService } from '../users/users.service';
import { Request } from 'express';
// Giđ sđ có ClerkClient đđc provide:
// import { ClerkClient } from '@clerk/backend';

@Injectable()
export class ClerkStrategy extends PassportStrategy(Strategy, 'clerk') {
  constructor(
    private readonly usersService: UsersService,
    private readonly configService: ConfigService,
    // @Inject('ClerkClient') private readonly clerkClient: ClerkClient, ❶
  ) {
    super();
  }

  async validate(req: Request): Promise<any> {
    const token = req.headers.authorization?.split(' ')[1];
    if (!token) {
      throw new UnauthorizedException('No token provided');
    }
    try {
      // Nên sđ dng ClerkClient đđc inject nđu có
      const tokenPayload = await verifyToken(token, { ❷
        secretKey: this.configService.get('CLERK_SECRET_KEY'),
        // issuer: Cđn cđu hình linh hođt hđn
      });
      // Logic tìm hòc tđo user tđ DB
      const user = await this.usersService.findOrCreateFromClerk(tokenPayload); // Giđ
đđnh phđng thđc này
      return user;
    } catch (error) {
      console.error('Clerk Strategy Error:', error);
      throw new UnauthorizedException('Invalid token or authentication failed');
    }
  }
}
```

```
}
```

- ① Nên inject `ClerkClient` nếu được cung cấp bởi một provider.
- ② Logic xác thực token nên nằm ở đây.

2.4.2.4. ClerkAuthGuard - Phân Tích Chi Tiết (`src/modules/auth/guards/clerk-auth.guard.ts`)

Code Hiện Tại

```
// ... imports ...
import { clerkClient } from '@clerk/clerk-sdk-node'; ①
// ...
@Injectable()
export class ClerkAuthGuard implements CanActivate { ②
  constructor(
    @Inject('CLERK_OPTIONS') private options: ClerkModuleOptions,
  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    // ... logic trích xuất token ...
    const sessionToken = await clerkClient.verifyToken(token, { ①
      secretKey: this.options.secretKey,
      issuer: `https://clerk.${this.options.publishableKey.split('_')[1]}.lcl.dev`, ③
    });
    // ... logic lấy session, user ...
    return true;
  }
}
```

Đề Xuất Theo Tài Liệu Tham Khảo (mục 2.3.3)

```
import { ExecutionContext, Injectable } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { AuthGuard } from '@nestjs/passport';
import { IS_PUBLIC_KEY } from '../decorators/public.decorator'; ④

@Injectable()
export class ClerkAuthGuard extends AuthGuard('clerk') { ②
  constructor(private reflector: Reflector) { ④
    super();
  }

  canActivate(context: ExecutionContext) {
    const isPublic = this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [ ④
      context.getHandler(),
      context.getClass(),
    ]);
  }
}
```

```

    if (isPublic) {
      return true;
    }
    return super.canActivate(context); ②
  }
}

```

- ① Guard đang tự gọi `clerkClient` từ SDK. Logic này nên thuộc về `ClerkStrategy`.
- ② Guard nên kế thừa `AuthGuard('clerk')` và gọi `super.canActivate(context)`.
- ③ Hardcoded `issuer` URL. Cần cấu hình linh hoạt hơn.
- ④ Thiếu xử lý cho route `@Public()` sử dụng `Reflector` và `IS_PUBLIC_KEY`. Cần tạo decorator `@Public()`.

2.5.2.5. ClerkModule - Phân Tích Chi Tiết (`src/modules/clerk/clerk.module.ts`)

Code Hiện Tại

```

// ...
@Module({})
export class ClerkModule {
  static forRootAsync(): DynamicModule {
    return {
      // ...
      providers: [
        {
          provide: 'CLERK_OPTIONS', // Cung cấp CLERK_OPTIONS (keys)
          useFactory: (configService: ConfigService): ClerkModuleOptions => ({
            secretKey: configService.get<string>('CLERK_SECRET_KEY'),
            publishableKey: configService.get<string>('CLERK_PUBLISHABLE_KEY'),
          }),
          inject: [ConfigService],
        },
        ClerkSessionService,
      ],
      exports: [ClerkSessionService, 'CLERK_OPTIONS'],
      global: true,
    };
  }
}

```

Đề Xuất Theo Tài Liệu Tham Khảo (mục 2.3.1 - `ClerkClientProvider`)

```

// src/modules/clerk/clerk-client.provider.ts (hoàc trong clerk.module.ts)
import { createClerkClient, ClerkClient } from '@clerk/backend';
import { ConfigService } from '@nestjs/config';

export const ClerkClientProvider = {
  provide: 'ClerkClient', // Hoà trộn InjectionToken/Class

```

```

useFactory: (configService: ConfigService): ClerkClient => {
  return createClerkClient({
    secretKey: configService.get('CLERK_SECRET_KEY'),
    // publishableKey: có thể không cần cho backend client
  });
},
inject: [ConfigService],
};

// Trong clerk.module.ts
// @Module({
//   providers: [ClerkClientProvider, ClerkSessionService, /* ... CLERK_OPTIONS
//   provider ... */],
//   exports: [ClerkClientProvider, ClerkSessionService, /* ... CLERK_OPTIONS provider
//   ... */],
// })

```

Hiện tại **ClerkModule** cung cấp **CLERK_OPTIONS** (chứa keys) nhưng không trực tiếp cung cấp một instance của **ClerkClient**. **ClerkAuthGuard** đang import **clerkClient** trực tiếp từ SDK. **Khuyến nghị:** Tạo một provider cho **ClerkClient** (ví dụ: **ClerkClientProvider**) như trong tài liệu tham khảo. Provider này nên được đặt trong **ClerkModule** và được export để **ClerkStrategy** có thể inject.

2.6. 2.6. AppModule - Phân Tích Chi Tiết (src/app.module.ts)

Code Hiện Tại

```

// ...
@Module({
  imports: [
    ConfigModule.forRoot({ isGlobal: true, /* ... */ }), // ConfigModule đúng
    AuthModule,
    ClerkModule.forRootAsync(), // ClerkModule được import
    // ...
  ],
  providers: [AppService], ①
})
export class AppModule {}

```

Đề Xuất Theo Tài Liệu Tham Khảo (mục 2.4.2)

```

import { APP_GUARD } from '@nestjs/core';
import { ClerkAuthGuard } from '../modules/auth/guards/clerk-auth.guard';
// ...
@Module({
  // ...
  providers: [
    AppService,

```

```
// ClerkClientProvider, // <2> Nếu không đăt trong ClerkModule
{
  provide: APP_GUARD,
  useClass: ClerkAuthGuard, // <1> Đăng ký ClerkAuthGuard làm global guard
},
],
})
export class AppModule {}
```

- ① Thiếu đăng ký `ClerkAuthGuard` làm Global Guard sử dụng `APP_GUARD`.
- ② Thiếu `ClerkClientProvider` nếu nó không được cung cấp và export bởi `ClerkModule`.

3. 3. Tổng Hợp Các Vấn Đề và Đề Xuất

Dựa trên phân tích, các điểm chính cần cải thiện bao gồm:

1. AuthModule:

- **Vấn đề:** Thiếu `PassportModule`, `ConfigModule`, và `ClerkStrategy` trong `providers`.
- **Đề xuất:** Cập nhật `AuthModule` để import các module cần thiết và cung cấp `ClerkStrategy`.

2. ClerkStrategy:

- **Vấn đề:** Hoàn toàn không tồn tại.
- **Đề xuất:** Tạo tệp `src/modules/auth/strategies/clerk.strategy.ts`. Implement `ClerkStrategy` kế thừa `PassportStrategy`, inject `UsersService`, `ConfigService`, và `ClerkClient` (từ provider). Logic `validate` để xác thực token Clerk.

3. ClerkAuthGuard:

- **Vấn đề:** Tự thực hiện logic xác thực, không kế thừa `AuthGuard('clerk')`, thiếu xử lý route `@Public()`, sử dụng `clerkClient` trực tiếp từ SDK, hardcoded `issuer` URL.
- **Đề xuất:** Refactor `ClerkAuthGuard` để kế thừa `AuthGuard('clerk')`, inject `Reflector`, implement logic cho `@Public()`. Logic xác thực token sẽ do `ClerkStrategy` đảm nhiệm. Cấu hình `issuer` linh hoạt.

4. Cung cấp ClerkClient:

- **Vấn đề:** `ClerkAuthGuard` (và `ClerkStrategy` tương lai) không nên import `clerkClient` trực tiếp từ SDK.
- **Đề xuất:** Tạo `ClerkClientProvider` trong `ClerkModule` để khởi tạo và cung cấp instance của `ClerkClient`. `ClerkStrategy` sẽ inject client này.

5. AppModule:

- **Vấn đề:** Không đăng ký `ClerkAuthGuard` làm Global Guard.
- **Đề xuất:** Đăng ký `ClerkAuthGuard` (sau khi đã refactor) làm global guard sử dụng `APP_GUARD`.

6. Biến Môi Trường:

- **Hiện trạng:** `ConfigModule` đã được cấu hình trong `AppModule` và `ClerkModule` sử dụng `ConfigService` để lấy keys.

- **Đề xuất:** Đảm bảo `CLERK_PUBLISHABLE_KEY` và `CLERK_SECRET_KEY` được định nghĩa đúng trong file `.env`.

4. 4. Kế Hoạch Hành Động Đề Xuất

1. Tạo Decorator `@Public()`:

- Tạo tệp `src/modules/auth/decorators/public.decorator.ts`.
- Định nghĩa `IS_PUBLIC_KEY` và decorator `@Public()`.

2. Nâng Cấp `ClerkModule`:

- Thêm `ClerkClientProvider` vào `providers` và `exports` của `ClerkModule` (`[src/modules/clerk/clerk.module.ts](src/modules/clerk/clerk.module.ts:1)`). Provider này sẽ tạo và trả về instance của `ClerkClient`.

3. Tạo `ClerkStrategy`:

- Tạo tệp `src/modules/auth/strategies/clerk.strategy.ts` (`[src/modules/auth/strategies/clerk.strategy.ts](src/modules/auth/strategies/clerk.strategy.ts:1)`).
- Implement `ClerkStrategy` kế thừa `PassportStrategy(Strategy, 'clerk')`.
- Inject `UsersService`, `ConfigService`, và `ClerkClient` (từ `ClerkModule`).
- Implement phương thức `validate` để xác thực token và trả về thông tin người dùng.

4. Refactor `AuthModule`:

- Cập nhật `[src/modules/auth/auth.module.ts](src/modules/auth/auth.module.ts:1)`.
- Import `PassportModule` và `ConfigModule`.
- Thêm `ClerkStrategy` vào `providers`.
- Export `PassportModule`.

5. Refactor `ClerkAuthGuard`:

- Cập nhật `[src/modules/auth/guards/clerk-auth.guard.ts](src/modules/auth/guards/clerk-auth.guard.ts:1)`.
- Kế thừa từ `AuthGuard('clerk')`.
- Inject `Reflector`.
- Implement phương thức `canActivate` để xử lý decorator `@Public()` và gọi `super.canActivate(context)`.

6. Cập Nhật `AppModule`:

- Cập nhật `[src/app.module.ts](src/app.module.ts:1)`.
- Đăng ký `ClerkAuthGuard` (đã refactor) làm global guard sử dụng `APP_GUARD`.

7. Kiểm Tra và Thử Nghiệm:

- Đảm bảo các biến môi trường Clerk được thiết lập đúng.
- Kiểm tra kỹ lưỡng các API endpoint (cả public và protected) để đảm bảo xác thực hoạt động

chính xác.

- Kiểm tra việc truy cập thông tin người dùng trong controller.

5. 5. Kết Luận

Việc triển khai các đề xuất trên sẽ giúp hệ thống xác thực của dự án TheShoe tuân thủ tốt hơn các chuẩn mực của NestJS và Clerk, đồng thời cải thiện tính bảo mật, khả năng bảo trì và mở rộng. Kiến trúc mới sẽ rõ ràng hơn, với sự phân tách trách nhiệm tốt hơn giữa Guard và Strategy.