

Báo Cáo Code Review: Module Auth và Clerk

Table of Contents

1. Tóm tắt Nhiệm vụ	2
2. Chi tiết Triển khai Mã nguồn	2
2.1. Module Auth (src/modules/auth/)	2
2.1.1. auth.module.ts	2
2.1.2. auth.service.ts	3
2.1.3. auth.controller.ts	4
2.1.4. roles.decorator.ts	5
2.1.5. roles.guard.ts	6
2.2. Module Clerk (src/modules/Infrastructure/clerk/)	8
2.2.1. clerk.module.ts	8
2.2.2. clerk.controller.ts	10
2.2.3. clerk.session.service.ts	12
2.2.4. clerk-auth.guard.ts	15
3. Kiểm thử	16
3.1. Kiểm thử Đơn vị (Unit Tests)	16
3.1.1. test/unit/modules/auth/guards/roles.guard.spec.ts	16
3.1.2. test/unit/modules/Infrastructure/clerk/clerk.controller.spec.ts	18
3.1.3. test/unit/modules/Infrastructure/clerk/clerk.module.spec.ts	19
3.2. Kiểm thử Tích hợp (Integration Tests)	21
3.2.1. test/integration/clerk-admin-endpoints.integration.spec.ts	22
3.3. Kiểm thử Đầu cuối (End-to-End Tests)	23
3.3.1. test/e2e/clerk-admin-e2e.spec.ts	23
4. Thách thức và Giải pháp	25
4.1. Thách thức: Vi phạm Tách biệt Trách nhiệm và Trùng lặp Mã	25
4.2. Thách thức: RolesGuard Yếu và Lỗ hổng Bảo mật "Fail-Open"	26
4.3. Thách thức: Phụ thuộc của ClerkController vào RolesGuard	26
5. Cải tiến và Tối ưu hóa	26
5.1. Cải tiến Kiến trúc	26
5.2. Cải tiến Bảo mật	27
5.3. Cải tiến Khả năng Mở rộng và Bảo trì	27
5.4. Tối ưu hóa Hiệu suất (Đã được phân tích trong tài liệu)	27
6. Công cụ và Công nghệ Sử dụng	27
7. Kết luận	28
7.1. Khuyến nghị Tiếp theo	28

1. Tóm tắt Nhiệm vụ

Báo cáo này cung cấp một cái nhìn tổng quan và phân tích chi tiết về quá trình tái cấu trúc và trạng thái hiện tại của module xác thực (**AuthModule**) và module quản lý người dùng bên ngoài (**ClerkModule**) trong dự án TheShoeBolt. Mục tiêu chính của việc tái cấu trúc là tách biệt rõ ràng trách nhiệm giữa **Authentication** (xác thực người dùng) và **Authorization** (phân quyền truy cập), loại bỏ các vi phạm kiến trúc, giảm trùng lặp mã, và cải thiện tính bảo mật, khả năng bảo trì cũng như khả năng mở rộng của hệ thống.

Quá trình review đã xác nhận rằng các mục tiêu chính của việc tái cấu trúc đã được đáp ứng thành công, với việc loại bỏ **AdminGuard** cũ, cải thiện **RolesGuard** và thiết lập một kiến trúc rõ ràng hơn.

2. Chi tiết Triển khai Mã nguồn

Dưới đây là phân tích chi tiết các thành phần chính của **AuthModule** và **ClerkModule** sau tái cấu trúc:

2.1. Module **Auth** (**src/modules/auth/**)

Module **Auth** chịu trách nhiệm về logic nghiệp vụ liên quan đến xác thực và phân quyền.

2.1.1. **auth.module.ts**

```
``typescript
import { Module } from '@nestjs/common';
import { UsersModule } from '../users/users.module';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { RolesGuard } from './guards/roles.guard';
import { ClerkModule } from '../Infrastructure/clerk/clerk.module';

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

- **Giải thích:** Module này import **UsersModule** (để quản lý người dùng cục bộ) và **ClerkModule** (để sử dụng các dịch vụ xác thực của Clerk). Nó cung cấp **AuthService** (logic nghiệp vụ) và **RolesGuard** (logic phân quyền), đồng thời export chúng để các module khác có thể sử dụng.

### 2.1.2. auth.service.ts

```
import { Injectable } from '@nestjs/common';
import { UsersService } from '../users/users.service';
import { CreateUserDto } from '../users/dto/create-user.dto';

@Injectable()
export class AuthService {
 constructor(
 private usersService: UsersService,
) {}

 /**
 * Sync user data từ Clerk vào local database
 * Được gọi sau khi user authenticate thành công qua Clerk
 */
 async syncUserFromClerk(clerkUser: any) {
 const existingUser = await this.usersService.findByEmail(clerkUser.email);

 if (!existingUser) {
 // Create new user in local database
 const userData: CreateUserDto = {
 email: clerkUser.email,
 firstName: clerkUser.firstName || '',
 lastName: clerkUser.lastName || '',
 password: 'clerk_managed', // Password không được quản lý local
 role: clerkUser.publicMetadata?.role || 'user',
 };

 return await this.usersService.create(userData);
 }

 // Update existing user data if needed
 if (existingUser.firstName !== clerkUser.firstName ||
 existingUser.lastName !== clerkUser.lastName) {
 await this.usersService.update(existingUser.id, {
 firstName: clerkUser.firstName,
 lastName: clerkUser.lastName,
 });
 }

 return existingUser;
 }

 /**
 * Get user profile từ local database
 */
}
```

```

 * Dùng cho các endpoints cần thông tin user từ DB local
 */
 async getUserProfile(userId: string) {
 return this.userService.findOne(userId);
 }
}

```

- **Giải thích:** `AuthService` quản lý việc đồng bộ hóa dữ liệu người dùng từ Clerk vào cơ sở dữ liệu cục bộ (`syncUserFromClerk`) và lấy thông tin hồ sơ người dùng từ cơ sở dữ liệu cục bộ (`getUserProfile`). Đây là cầu nối giữa dữ liệu người dùng từ Clerk và hệ thống cục bộ.

### 2.1.3. `auth.controller.ts`

```

``typescript
import { Controller, Get, Post, UseGuards, Request, Body } from '@nestjs/common';
import { ApiTags, ApiOperation, ApiResponse, ApiBearerAuth } from '@nestjs/swagger';
import { AuthService } from '../auth.service';
import { ClerkAuthGuard } from '../Infrastructure/clerk/guards/clerk-auth.guard';
import { RolesGuard } from '../guards/roles.guard';
import { Roles } from '../decorators/roles.decorator';
import { UserRole } from '../users/entities/user.entity';

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

 @UseGuards(ClerkAuthGuard)
 @Post('sync-user')
 @ApiOperation({ summary: 'Sync authenticated Clerk user to local database' })
 @ApiResponse({ status: 200, description: 'User synced successfully' })
 @ApiResponse({ status: 401, description: 'Unauthorized' })
 @ApiBearerAuth()
 async syncUser(@Request() req) {
 const localUser = await this.authService.syncUserFromClerk(req.user);
 return {
 message: 'User synced successfully',
 user: {
 id: localUser.id,
 email: localUser.email,
 firstName: localUser.firstName,
 lastName: localUser.lastName,
 role: localUser.role,
 },
 };
 }

 @UseGuards(ClerkAuthGuard)
 @Get('profile')

```

```

@ApiOperation({ summary: 'Get user profile' })
@ApiResponse({ status: 200, description: 'User profile retrieved successfully' })
@ApiResponse({ status: 401, description: 'Unauthorized' })
@ApiBearerAuth()
async getProfile(@Request() req) {
 const localUser = await this.authService.getUserProfile(req.user.id);
 return {
 message: 'Profile retrieved successfully',
 user: {
 ...req.user, // Clerk user data
 localData: localUser, // Local database data
 },
 session: {
 id: req.session?.id,
 status: req.session?.status,
 },
 };
}

@UseGuards(ClerkAuthGuard, RolesGuard)
@Roles(UserRole.ADMIN)
@Get('admin-only')
@ApiOperation({ summary: 'Admin only endpoint' })
@ApiResponse({ status: 200, description: 'Admin access granted' })
@ApiResponse({ status: 401, description: 'Unauthorized' })
@ApiResponse({ status: 403, description: 'Forbidden - Admin role required' })
@ApiBearerAuth()
async adminOnly(@Request() req) {
 return {
 message: 'Admin access granted',
 user: req.user,
 };
}
}
```

```

- **Giải thích:** `AuthController` cung cấp các endpoint API cho việc đồng bộ hóa người dùng (`/auth/sync-user`), lấy hồ sơ người dùng (`/auth/profile`), và một endpoint chỉ dành cho quản trị viên (`/auth/admin-only`). Nó sử dụng `ClerkAuthGuard` để xác thực và `RolesGuard` cùng với `@Roles` decorator để kiểm tra vai trò.

2.1.4. `roles.decorator.ts`

```

import { SetMetadata } from '@nestjs/common';
import { UserRole } from '../users/entities/user.entity';

// Export constant để tránh magic strings và đảm bảo tính nhất quán
export const ROLES_KEY = 'roles';

```

```
export const Roles = (...roles: UserRole[]) => SetMetadata(ROLES_KEY, roles);
```

- **Giải thích:** Định nghĩa hằng số `ROLES_KEY` và decorator `@Roles`. Decorator này được sử dụng để gán các vai trò yêu cầu cho các endpoint hoặc controller, cho phép `RolesGuard` đọc và thực thi các quy tắc phân quyền.

2.1.5. `roles.guard.ts`

```
``typescript
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  ForbiddenException,
  InternalServerErrorException,
  Logger,
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { ROLES_KEY } from '../decorators/roles.decorator';
import { UserRole } from '../../users/entities/user.entity';

// Định nghĩa một kiểu cho payload của người dùng từ Clerk để tăng tính an toàn về kiểu
interface ClerkUserPayload {
  publicMetadata?: {
    role?: UserRole; // Hạng vai trò đơn lẻ như hiện tại
    roles?: UserRole[]; // Hạng tập các vai trò cho tương lai
  };
  // Thêm các thuộc tính khác của user nếu cần
  id?: string;
  emailAddresses?: Array<{ emailAddress: string }>;
}

@Injectable()
export class RolesGuard implements CanActivate {
  private readonly logger = new Logger(RolesGuard.name);

  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    // Sử dụng getAllAndOverride để lấy các vai trò từ cả handler và class
    const requiredRoles = this.reflector.getAllAndOverride<UserRole[]>(ROLES_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    // Nếu không có vai trò nào được yêu cầu, áp dụng nguyên tắc fail-safe
    // Guard này chỉ nên được kích hoạt trên các endpoint có decorator @Roles.
    // Các endpoint công khai nên được xử lý bởi một @Public decorator và một AuthGuard
    // toàn cục.
  }
}
```

```

    if (!requiredRoles || requiredRoles.length === 0) {
        this.logger.warn('RolesGuard được áp dụng cho endpoint không có @Roles decorator.
        Chỉ truy cập theo nguyên tắc fail-safe.');
```

throw new ForbiddenException('Access denied: No role requirements specified for this endpoint.');

```
    }

    const request = context.switchToHttp().getRequest();
    const user = request.user as ClerkUserPayload;

    // 1. Kiểm tra phòng vé: Đảm bảo 'user' tồn tại
    if (!user) {
        this.logger.error('User object is missing in RolesGuard. Ensure an authentication guard runs before it.');
```

throw new InternalServerErrorException('User authentication data is not available.');

```
    }

    // 2. Trích xuất vai trò của người dùng một cách an toàn
    const userRoles = this.extractUserRoles(user);

    // 3. Kiểm tra xem người dùng có vai trò hay không
    if (!userRoles || userRoles.length === 0) {
        this.logger.warn(`User ${user.id || 'unknown'} không có vai trò nào được gán.`);
        throw new ForbiddenException('You have not been assigned any roles.');
```

```
    }

    // 4. Thực hiện so khớp vai trò
    const hasPermission = this.matchRoles(requiredRoles, userRoles);

    if (!hasPermission) {
        this.logger.warn(`User ${user.id || 'unknown'} với roles [${userRoles.join(', ')}] không có quyền truy cập endpoint yêu cầu roles [${requiredRoles.join(', ')}].`);
        throw new ForbiddenException('You do not have the required permissions to access this resource.');
```

```
    }

    this.logger.debug(`User ${user.id || 'unknown'} được phép truy cập với roles [${userRoles.join(', ')}].`);
    return true;
}

/**
 * Trích xuất danh sách vai trò của người dùng từ Clerk payload
 * Hết rồi sẽ định dạng cũ (role đơn lẻ) và định dạng mới (roles array)
 * @param user Clerk user payload
 * @returns Mảng các vai trò của người dùng
 */
private extractUserRoles(user: ClerkUserPayload): UserRole[] {
    if (!user.publicMetadata) {
        return [];
    }

```

```

}

// Ưu tiên sử dụng roles array nếu có (cho tương lai)
if (user.publicMetadata.roles && Array.isArray(user.publicMetadata.roles)) {
  return user.publicMetadata.roles;
}

// Fallback sang role đơn lẻ (hiện tại)
if (user.publicMetadata.role) {
  return [user.publicMetadata.role];
}

// Không có vai trò nào
return [];
}

/**
 * So khớp vai trò yêu cầu với vai trò của người dùng.
 * @param requiredRoles Các vai trò được yêu cầu bởi endpoint.
 * @param userRoles Các vai trò mà người dùng hiện tại có.
 * @returns `true` nếu người dùng có ít nhất một trong các vai trò yêu cầu.
 */
private matchRoles(requiredRoles: UserRole[], userRoles: UserRole[]): boolean {
  // Logic cơ bản: kiểm tra intersection
  // Có thể được mở rộng ở đây để hỗ trợ thỏa mãn vai trò trong tương lai
  // Ví dụ: nếu requiredRoles có 'USER' và userRoles có 'ADMIN', nó nên trả về true.
  return requiredRoles.some((role) => userRoles.includes(role));
}
}
```

```

- **Giải thích:** `RolesGuard` là một thành phần quan trọng trong việc thực thi phân quyền. Nó đọc các vai trò yêu cầu từ decorator `@Roles` và so sánh chúng với vai trò của người dùng hiện tại (được lấy từ `publicMetadata` của Clerk). Guard này đã được tái cấu trúc để trở nên mạnh mẽ hơn, hỗ trợ nhiều vai trò, xử lý các trường hợp biên và ném ra các ngoại lệ rõ ràng khi truy cập bị từ chối. Nó tuân thủ nguyên tắc "fail-safe", tức là từ chối truy cập nếu không có vai trò nào được chỉ định rõ ràng.

## 2.2. Module `Clerk` (`src/modules/Infrastructure/clerk/`)

Module `Clerk` đóng vai trò là tầng hạ tầng, chuyên trách việc tương tác trực tiếp với Clerk SDK và cung cấp các dịch vụ xác thực.

### 2.2.1. `clerk.module.ts`

```

``typescript
import { DynamicModule, Module } from '@nestjs/common';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { ClerkSessionService } from './clerk.session.service';

```



```

import { ClerkController } from './clerk.controller';
import { ClerkAuthGuard } from './guards/clerk-auth.guard';

export interface ClerkModuleOptions {
 secretKey: string;
 publishableKey: string;
}

@Module({})
export class ClerkModule {
 static forRoot(options: ClerkModuleOptions): DynamicModule {
 return {
 module: ClerkModule,
 controllers: [ClerkController],
 providers: [
 {
 provide: 'CLERK_OPTIONS',
 useValue: options,
 },
 ClerkSessionService,
 ClerkAuthGuard,
],
 exports: [ClerkSessionService, ClerkAuthGuard, 'CLERK_OPTIONS'],
 global: true,
 };
 }

 static forRootAsync(): DynamicModule {
 return {
 module: ClerkModule,
 imports: [ConfigModule],
 controllers: [ClerkController],
 providers: [
 {
 provide: 'CLERK_OPTIONS',
 useFactory: (configService: ConfigService): ClerkModuleOptions => ({
 secretKey: configService.get<string>('CLERK_SECRET_KEY'),
 publishableKey: configService.get<string>('CLERK_PUBLISHABLE_KEY'),
 }),
 inject: [ConfigService],
 },
 ClerkSessionService,
 ClerkAuthGuard,
],
 exports: [ClerkSessionService, ClerkAuthGuard, 'CLERK_OPTIONS'],
 global: true,
 };
 }
}

```

- **Giải thích:** `ClerkModule` là một `DynamicModule`, cho phép cấu hình linh hoạt (đồng bộ hoặc bất đồng bộ thông qua `ConfigModule`). Nó cung cấp `ClerkSessionService` (tương tác với Clerk API) và `ClerkAuthGuard` (guard xác thực chính). Module này được đánh dấu là `global: true`, cho phép các module khác sử dụng các dịch vụ của nó mà không cần import trực tiếp. Quan trọng là, nó không còn export hoặc cung cấp bất kỳ logic phân quyền nào (như `AdminGuard` cũ), đảm bảo sự tách biệt trách nhiệm.

### 2.2.2. `clerk.controller.ts`

```
``typescript
import {
 Controller,
 Get,
 Delete,
 Post,
 Param,
 UseGuards,
 Request,
 HttpStatusCode,
 HttpStatus
} from '@nestjs/common';
import { ApiTags, ApiOperation, ApiResponse, ApiParam } from '@nestjs/swagger';
import { ClerkSessionService } from '../clerk.session.service';
import { ClerkAuthGuard } from '../guards/clerk-auth.guard';
import { Roles } from '../auth/decorators/roles.decorator';
import { RolesGuard } from '../auth/guards/roles.guard';
import { UserRole } from '../users/entities/user.entity';

@ApiTags('Clerk Session Management')
@Controller('clerk')
@UseGuards(ClerkAuthGuard)
export class ClerkController {
 constructor(private readonly clerkSessionService: ClerkSessionService) {}

 @Get('sessions')
 @ApiOperation({ summary: 'Get all sessions for current user' })
 @ApiResponse({ status: 200, description: 'Sessions retrieved successfully' })
 @ApiResponse({ status: 401, description: 'Unauthorized' })
 async getUserSessions(@Request() req) {
 const sessions = await this.clerkSessionService.getSessionList(req.user.id);
 return {
 message: 'Sessions retrieved successfully',
 sessions,
 };
 }

 @Delete('sessions/:sessionId')
 @HttpCode(HttpStatus.NO_CONTENT)
 @ApiOperation({ summary: 'Revoke a specific session' })
 @ApiParam({ name: 'sessionId', description: 'Session ID to revoke' })
```

```

 @ApiResponse({ status: 204, description: 'Session revoked successfully' })
 @ApiResponse({ status: 401, description: 'Unauthorized' })
 async revokeSession(@Param('sessionId') sessionId: string) {
 await this.clerkSessionService.revokeSession(sessionId);
 return;
 }

 @Delete('sessions')
 @HttpCode(HttpStatus.NO_CONTENT)
 @ApiOperation({ summary: 'Revoke all sessions for current user' })
 @ApiResponse({ status: 204, description: 'All sessions revoked successfully' })
 @ApiResponse({ status: 401, description: 'Unauthorized' })
 async revokeAllSessions(@Request() req) {
 await this.clerkSessionService.revokeAllUserSessions(req.user.id);
 return;
 }

 @UseGuards(ClerkAuthGuard, RolesGuard)
 @Roles(UserRole.ADMIN)
 @Get('admin/users/:userId/sessions')
 @ApiOperation({ summary: 'Admin: Get sessions for any user' })
 @ApiParam({ name: 'userId', description: 'User ID to get sessions for' })
 async getAnyUserSessions(@Param('userId') userId: string) {
 const sessions = await this.clerkSessionService.getSessionList(userId);
 return {
 message: 'User sessions retrieved successfully',
 userId,
 sessions,
 };
 }

 @UseGuards(ClerkAuthGuard, RolesGuard)
 @Roles(UserRole.ADMIN)
 @Delete('admin/users/:userId/sessions')
 @HttpCode(HttpStatus.NO_CONTENT)
 @ApiOperation({ summary: 'Admin: Revoke all sessions for any user' })
 @ApiParam({ name: 'userId', description: 'User ID to revoke sessions for' })
 async revokeAllUserSessions(@Param('userId') userId: string) {
 await this.clerkSessionService.revokeAllUserSessions(userId);
 return;
 }
}
...

```

- **Giải thích:** `ClerkController` cung cấp các endpoint API để quản lý phiên Clerk (ví dụ: lấy danh sách phiên, thu hồi phiên). Các endpoint này được bảo vệ bởi `ClerkAuthGuard`. Đặc biệt, các endpoint quản trị viên (ví dụ: `admin/users/:userId/sessions`) được bảo vệ thêm bởi `RolesGuard` và `@Roles(UserRole.ADMIN)`, đảm bảo chỉ người dùng có vai trò ADMIN mới có thể truy cập.

### 2.2.3. clerk.session.service.ts

```
``typescript
import { Injectable, Inject, UnauthorizedException } from '@nestjs/common';
import { clerkClient } from '@clerk/clerk-sdk-node';
import { ClerkModuleOptions } from './clerk.module';

@Injectable()
export class ClerkSessionService {
 private clerk;

 constructor(
 @Inject('CLERK_OPTIONS') private options: ClerkModuleOptions,
) {
 // clerkClient is already initialized with the secret key from environment
 this.clerk = clerkClient;
 }

 /**
 * Get list of sessions for a specific user
 * @param userId - Clerk user ID
 * @returns Array of user sessions
 */
 async getSessionList(userId: string) {
 try {
 const sessions = await this.clerk.sessions.getSessionList({
 userId,
 });
 return sessions;
 } catch (error) {
 throw new UnauthorizedException(`Failed to get sessions: ${error.message}`);
 }
 }

 /**
 * Revoke a specific session
 * @param sessionId - Session ID to revoke
 * @returns Revoked session data
 */
 async revokeSession(sessionId: string) {
 try {
 const revokedSession = await this.clerk.sessions.revokeSession(sessionId);
 return revokedSession;
 } catch (error) {
 throw new UnauthorizedException(`Failed to revoke session: ${error.message}`);
 }
 }

 /**
 * Verify a session token and return session claims

```

```

* @param token - Session token to verify
* @returns Session claims if valid
*/
async verifySessionToken(token: string) {
 try {
 const sessionClaims = await this.clerk.verifyToken(token, {
 secretKey: this.options.secretKey,
 issuer: `https://clerk.${this.options.publishableKey.split('_')[1]}.lcl.dev`,
 });
 return sessionClaims;
 } catch (error) {
 throw new UnauthorizedException(`Invalid session token: ${error.message}`);
 }
}

/**
* Get session details by session ID
* @param sessionId - Session ID
* @returns Session details
*/
async getSession(sessionId: string) {
 try {
 const session = await this.clerk.sessions.getSession(sessionId);
 return session;
 } catch (error) {
 throw new UnauthorizedException(`Failed to get session: ${error.message}`);
 }
}

/**
* Get user details by user ID
* @param userId - User ID
* @returns User details
*/
async getUser(userId: string) {
 try {
 const user = await this.clerk.users.getUser(userId);
 return user;
 } catch (error) {
 throw new UnauthorizedException(`Failed to get user: ${error.message}`);
 }
}

/**
* Verify token and get complete authentication data
* @param token - Session token to verify
* @returns Complete authentication data including user, session, and claims
*/
async verifyTokenAndGetAuthData(token: string) {
 try {
 // Verify the session token

```

```

 const sessionClaims = await this.verifySessionToken(token);

 // Get session information
 const session = await this.getSession(sessionClaims.sid);

 if (!session || session.status !== 'active') {
 throw new UnauthorizedException('Invalid or inactive session');
 }

 // Get user information
 const user = await this.getUser(session.userId);

 return {
 user: {
 id: user.id,
 email: user.emailAddresses[0]?.emailAddress,
 firstName: user.firstName,
 lastName: user.lastName,
 publicMetadata: user.publicMetadata,
 },
 session,
 sessionClaims,
 };
} catch (error) {
 throw new UnauthorizedException(`Authentication failed: ${error.message}`);
}
}

/**
 * Revoke all sessions for a specific user
 * @param userId - Clerk user ID
 * @returns Array of revoked sessions
 */
async revokeAllUserSessions(userId: string) {
 try {
 // Get all user sessions first
 const sessions = await this.getSessionList(userId);

 // Revoke each session
 const revokedSessions = await Promise.all(
 sessions.map(session => this.revokeSession(session.id))
);

 return revokedSessions;
 } catch (error) {
 throw new UnauthorizedException(`Failed to revoke all user sessions: ${error.message}`);
 }
}
}

```

```

- **Giải thích:** `ClerkSessionService` là service chính tương tác với Clerk API thông qua `clerkClient`. Nó cung cấp các phương thức để quản lý phiên (lấy, thu hồi), xác minh token, và lấy thông tin người dùng từ Clerk. Đây là điểm duy nhất trong ứng dụng giao tiếp trực tiếp với Clerk SDK.

2.2.4. `clerk-auth.guard.ts`

```
``typescript
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  UnauthorizedException,
} from '@nestjs/common';
import { ClerkSessionService } from '../clerk.session.service';

@Injectable()
export class ClerkAuthGuard implements CanActivate {
  constructor(
    private readonly clerkSessionService: ClerkSessionService,
  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();

    try {
      // Extract token from Authorization header
      const authHeader = request.headers.authorization;
      if (!authHeader || !authHeader.startsWith('Bearer ')) {
        throw new UnauthorizedException('Missing or invalid authorization header');
      }

      const token = authHeader.substring(7); // Remove 'Bearer ' prefix

      // Use ClerkSessionService to verify token and get authentication data
      const authData = await
this.clerkSessionService.verifyTokenAndGetAuthData(token);

      // Attach authentication data to request object
      request.user = authData.user;
      request.session = authData.session;
      request.sessionClaims = authData.sessionClaims;

      return true;
    } catch (error) {
      throw new UnauthorizedException(`Authentication failed: ${error.message}`);
    }
  }
}
```

```
}  
...
```

- **Giải thích:** `ClerkAuthGuard` là guard xác thực chính. Nó trích xuất token từ header `Authorization`, sử dụng `ClerkSessionService` để xác minh token với Clerk, và đính kèm dữ liệu người dùng, phiên và yêu cầu phiên vào đối tượng `request`. Guard này đảm bảo rằng mọi yêu cầu đến các endpoint được bảo vệ đều đã được xác thực bởi Clerk.

3. Kiểm thử

Hệ thống xác thực và phân quyền đã được kiểm thử một cách toàn diện ở nhiều cấp độ, đảm bảo chất lượng và độ tin cậy của các thay đổi. Tổng cộng có **51+ test cases** được triển khai với coverage đầy đủ cho tất cả các thành phần.

3.1. Kiểm thử Đơn vị (Unit Tests)

3.1.1. `test/unit/modules/auth/guards/roles.guard.spec.ts`

Kiểm tra chi tiết logic của `RolesGuard` với 241 dòng code test, bao gồm:

```
describe('RolesGuard', () => {  
  let guard: RolesGuard;  
  let reflector: Reflector;  
  
  describe('Role Requirement Validation', () => {  
    it('should throw ForbiddenException when no roles are required (fail-safe)', () => {  
      {  
        mockReflector.getAllAndOverride.mockReturnValue(null);  
  
        expect(() => {  
          guard.canActivate(mockExecutionContext as ExecutionContext);  
        }).toThrow(ForbiddenException);  
        expect(() => {  
          guard.canActivate(mockExecutionContext as ExecutionContext);  
        }).toThrow('Access denied: No role requirements specified for this endpoint.');      });  
  
      it('should call reflector with correct parameters', () => {  
        mockReflector.getAllAndOverride.mockReturnValue([UserRole.ADMIN]);  
        guard.canActivate(mockExecutionContext as ExecutionContext);  
  
        expect(mockReflector.getAllAndOverride).toHaveBeenCalledWith(  
          ROLES_KEY,  
          [mockExecutionContext.getHandler(), mockExecutionContext.getClass()]  
        );  
      });  
    });  
  });  
});
```



```

describe('Role Extraction Logic', () => {
  it('should extract role from single role format (current format)', () => {
    const mockRequest = {
      user: {
        id: 'test-user-id',
        publicMetadata: {
          role: UserRole.ADMIN,
        },
      },
    };

    const result = guard.canActivate(mockExecutionContext as ExecutionContext);
    expect(result).toBe(true);
  });

  it('should extract roles from array format (future support)', () => {
    const mockRequest = {
      user: {
        publicMetadata: {
          roles: [UserRole.ADMIN, UserRole.USER],
        },
      },
    };

    const result = guard.canActivate(mockExecutionContext as ExecutionContext);
    expect(result).toBe(true);
  });

  it('should prioritize roles array over single role when both exist', () => {
    const mockRequest = {
      user: {
        publicMetadata: {
          role: UserRole.USER, // This should be ignored
          roles: [UserRole.ADMIN], // This should be used
        },
      },
    };

    const result = guard.canActivate(mockExecutionContext as ExecutionContext);
    expect(result).toBe(true);
  });
});

```

Các test cases chính:

- * **Fail-safe principle:** Xác nhận guard từ chối truy cập khi không có role specification
- * **Role extraction:** Test cả single role format (hiện tại) và array format (tương lai)
- * **Priority handling:** Roles array được ưu tiên hơn single role
- * **Error handling:** Kiểm tra các ngoại lệ `ForbiddenException` và `InternalServerErrorException`
- * **Defensive programming:** Validation user object và publicMetadata

3.1.2. test/unit/modules/Infrastructure/clerk/clerk.controller.spec.ts

Kiểm tra `ClerkController` với 250 dòng code test, tập trung vào:

```
describe('ClerkController', () => {
  describe('Admin Endpoints - Guard Configuration', () => {
    it('should require ADMIN role', () => {
      mockReflector.getAllAndOverride.mockReturnValue([UserRole.ADMIN]);

      const canActivate = rolesGuard.canActivate(mockExecutionContext as
ExecutionContext);
      expect(canActivate).toBe(true);
      expect(mockReflector.getAllAndOverride).toHaveBeenCalledWith(
        'roles',
        [mockExecutionContext.getHandler(), mockExecutionContext.getClass()]
      );
    });

    it('should deny access for non-admin users', () => {
      mockAdminRequest.user.publicMetadata.role = UserRole.USER;
      mockReflector.getAllAndOverride.mockReturnValue([UserRole.ADMIN]);

      expect(() => {
        rolesGuard.canActivate(mockExecutionContext as ExecutionContext);
      }).toThrow(ForbiddenException);
    });

    it('should get sessions for any user when authorized', async () => {
      const userId = 'target-user-id';
      const mockSessions = [{ id: 'session1' }];
      mockClerkSessionService.getSessionList.mockResolvedValue(mockSessions);

      const result = await controller.getAnyUserSessions(userId);

      expect(mockClerkSessionService.getSessionList).toHaveBeenCalledWith(userId);
      expect(result).toEqual({
        message: 'User sessions retrieved successfully',
        userId,
        sessions: mockSessions,
      });
    });
  });

  describe('Refactoring Verification', () => {
    it('should not use AdminGuard anymore', () => {
      const controllerMethods = Object.getOwnPropertyNames(ClerkController.prototype)
        .filter(name => name !== 'constructor');

      controllerMethods.forEach(methodName => {
        const method = ClerkController.prototype[methodName];
      });
    });
  });
});
```

```

    if (method) {
      const methodString = method.toString();
      expect(methodString).not.toContain('AdminGuard');
      expect(methodString).not.toContain('@AdminOnly');
    }
  });
});
});
});

```

Kiểm tra chính: * **Admin endpoint protection:** Xác nhận **RolesGuard** và **ClerkAuthGuard** hoạt động chính xác * **Access control:** Chỉ ADMIN role mới truy cập được admin endpoints * **Service integration:** ClerkSessionService methods được gọi đúng cách * **Refactoring validation:** Loại bỏ hoàn toàn AdminGuard và @AdminOnly decorator

3.1.3. test/unit/modules/Infrastructure/clerk/clerk.module.spec.ts

Kiểm tra cấu hình **ClerkModule** với 187 dòng code, đảm bảo sự tách biệt trách nhiệm:

```

describe('ClerkModule', () => {
  let module: TestingModule;

  describe('Module Configuration', () => {
    it('should provide ClerkSessionService', () => {
      const clerkSessionService =
module.get<ClerkSessionService>(ClerkSessionService);
      expect(clerkSessionService).toBeDefined();
      expect(clerkSessionService).toBeInstanceOf(ClerkSessionService);
    });

    it('should provide ClerkAuthGuard', () => {
      const clerkAuthGuard = module.get<ClerkAuthGuard>(ClerkAuthGuard);
      expect(clerkAuthGuard).toBeDefined();
      expect(clerkAuthGuard).toBeInstanceOf(ClerkAuthGuard);
    });

    it('should provide CLERK_OPTIONS token', () => {
      const clerkOptions = module.get('CLERK_OPTIONS');
      expect(clerkOptions).toBeDefined();
      expect(clerkOptions).toEqual({
        publishableKey: 'test-publishable-key',
        secretKey: 'test-secret-key',
      });
    });
  });

  describe('Module Refactoring Verification', () => {
    it('should NOT export AdminGuard (removed during refactoring)', () => {
      expect(() => {

```

```

        module.get('AdminGuard');
    }).toThrow();
});

it('should NOT provide any admin-specific guards or decorators', () => {
    const moduleProviders = Reflect.getMetadata('providers', ClerkModule) || [];
    const adminRelatedProviders = moduleProviders.filter((provider: any) =>
        provider?.name?.toLowerCase().includes('admin') ||
        provider?.toString?.()?.toLowerCase().includes('admin')
    );

    expect(adminRelatedProviders).toHaveLength(0);
});

it('should maintain clean separation of concerns - only authentication, no
authorization', () => {
    const clerkSessionService =
module.get<ClerkSessionService>(ClerkSessionService);
    const clerkAuthGuard = module.get<ClerkAuthGuard>(ClerkAuthGuard);

    expect(clerkSessionService).toBeDefined();
    expect(clerkAuthGuard).toBeDefined();

    expect(clerkSessionService.constructor.name).toBe('ClerkSessionService');
    expect(clerkAuthGuard.constructor.name).toBe('ClerkAuthGuard');
});
});

describe('Exported Services', () => {
    it('should export ClerkSessionService for other modules', async () => {
        const testConsumerModule = await Test.createTestingModule({
            imports: [
                ClerkModule.forRoot({
                    publishableKey: 'test-publishable-key',
                    secretKey: 'test-secret-key',
                }),
            ],
            providers: [
                {
                    provide: 'TestService',
                    useFactory: (clerkService: ClerkSessionService) => {
                        return { clerkService };
                    },
                    inject: [ClerkSessionService],
                },
            ],
        }).compile();

        const testService = testConsumerModule.get('TestService');
        expect(testService.clerkService).toBeDefined();
        expect(testService.clerkService).toBeInstanceOf(ClerkSessionService);
    });
});

```

```

    await testConsumerModule.close();
  });
});

describe('Dynamic Module Configuration', () => {
  it('should create different configurations for different environments', async ()
=> {
    const devModule = await Test.createTestingModule({
      imports: [
        ClerkModule.forRoot({
          publishableKey: 'dev-key',
          secretKey: 'dev-secret',
        }),
      ],
    }).compile();

    const prodModule = await Test.createTestingModule({
      imports: [
        ClerkModule.forRoot({
          publishableKey: 'prod-key',
          secretKey: 'prod-secret',
        }),
      ],
    }).compile();

    const devOptions = devModule.get('CLERK_OPTIONS');
    const prodOptions = prodModule.get('CLERK_OPTIONS');

    expect(devOptions.publishableKey).toBe('dev-key');
    expect(prodOptions.publishableKey).toBe('prod-key');

    await devModule.close();
    await prodModule.close();
  });
});
});

```

Kiểm tra chính:

- * **Module configuration:** Xác minh ClerkSessionService, ClerkAuthGuard và CLERK_OPTIONS được cung cấp đúng cách
- * **Refactoring verification:** Đảm bảo AdminGuard đã được loại bỏ hoàn toàn khỏi module
- * **Separation of concerns:** ClerkModule chỉ chứa authentication logic, không có authorization
- * **Service exports:** Xác nhận các service được export để other modules có thể sử dụng
- * **Dynamic configuration:** Hỗ trợ multiple environments với configuration khác nhau
- * **Edge case handling:** Xử lý gracefully các configuration không đầy đủ

3.2. Kiểm thử Tích hợp (Integration Tests)

3.2.1. test/integration/clerk-admin-endpoints.integration.spec.ts

Kiểm thử luồng HTTP đầy đủ với 439 dòng code, bao gồm:

```
describe('Clerk Admin Endpoints Integration', () => {
  describe('Authentication and Authorization Flow', () => {
    it('should authenticate and authorize admin for admin endpoints', async () => {
      mockClerkAuthGuard.canActivate.mockReturnValue(true);
      mockRolesGuard.canActivate.mockReturnValue(true);

      const mockSessions = [{ id: 'session1', status: 'active' }];
      mockClerkSessionService.getSessionList.mockResolvedValue(mockSessions);

      const response = await request(app.getHttpServer())
        .get('/clerk/admin/users/test-user-id/sessions')
        .set('Authorization', 'Bearer admin-token')
        .expect(200);

      expect(mockClerkAuthGuard.canActivate).toHaveBeenCalled();
      expect(mockRolesGuard.canActivate).toHaveBeenCalled();
      expect(response.body.message).toBe('User sessions retrieved successfully');
    });

    it('should reject unauthenticated requests', async () => {
      mockClerkAuthGuard.canActivate.mockReturnValue(false);
      mockRolesGuard.canActivate.mockReturnValue(true);

      await request(app.getHttpServer())
        .get('/clerk/admin/users/test-user-id/sessions')
        .expect(403);

      expect(mockClerkAuthGuard.canActivate).toHaveBeenCalled();
      // RolesGuard should not be called if auth fails
      expect(mockRolesGuard.canActivate).not.toHaveBeenCalled();
    });
  });

  describe('Guard Integration Testing', () => {
    it('should ensure proper guard execution order', async () => {
      const callOrder: string[] = [];

      mockClerkAuthGuard.canActivate.mockImplementation(() => {
        callOrder.push('ClerkAuthGuard');
        return true;
      });

      mockRolesGuard.canActivate.mockImplementation(() => {
        callOrder.push('RolesGuard');
        return true;
      });
    });
  });
});
```

```

    await request(app.getHttpServer())
      .get('/clerk/admin/users/test-id/sessions')
      .set('Authorization', 'Bearer admin-token')
      .expect(200);

    // ClerkAuthGuard should execute before RolesGuard
    expect(callOrder).toEqual(['ClerkAuthGuard', 'RolesGuard']);
  });
});
});

```

Kiểm tra chính: * **Full HTTP flow:** Request/Response cycle hoàn chỉnh với guards * **Guard execution order:** ClerkAuthGuard → RolesGuard * **Error scenarios:** Xử lý lỗi Clerk service timeout, rate limiting, network issues * **Authorization flow:** Admin vs regular user access patterns

3.3. Kiểm thử Đầu cuối (End-to-End Tests)

3.3.1. test/e2e/clerk-admin-e2e.spec.ts

Mô phỏng kịch bản người dùng thực tế với 455 dòng code:

```

describe('Clerk Admin E2E Flows', () => {
  describe('Complete Admin Session Management Flow', () => {
    it('should complete full admin workflow: view → revoke specific → revoke all',
      async () => {
        const targetUserId = mockTargetUser.id;
        const sessionId1 = 'session-to-revoke';
        const sessionId2 = 'session-to-keep';

        // Step 1: Admin views user sessions
        const initialSessions = [
          { id: sessionId1, status: 'active', lastActiveAt: new Date() },
          { id: sessionId2, status: 'active', lastActiveAt: new Date() },
        ];

        (clerkSessionService.getSessionList as
        jest.Mock).mockResolvedValueOnce(initialSessions);

        const viewSessionsResponse = await request(app.getHttpServer())
          .get(`/clerk/admin/users/${targetUserId}/sessions`)
          .set('Authorization', 'Bearer admin-token')
          .expect(200);

        expect(viewSessionsResponse.body).toEqual({
          message: 'User sessions retrieved successfully',
          userId: targetUserId,
          sessions: initialSessions,
        });
      });
    });
  });
});

```

```

// Step 2: Admin revokes specific session
(clerkSessionService.revokeSession as
jest.Mock).mockResolvedValueOnce(undefined);

await request(app.getHttpServer())
  .delete(`/clerk/sessions/${sessionId}`)
  .set('Authorization', 'Bearer admin-token')
  .expect(204);

// Step 3: Admin revokes all remaining sessions
(clerkSessionService.revokeAllUserSessions as
jest.Mock).mockResolvedValueOnce(undefined);

await request(app.getHttpServer())
  .delete(`/clerk/admin/users/${targetUserId}/sessions`)
  .set('Authorization', 'Bearer admin-token')
  .expect(204);

// Step 4: Verify all sessions are gone
(clerkSessionService.getSessionList as jest.Mock).mockResolvedValueOnce([]);

const finalSessionsResponse = await request(app.getHttpServer())
  .get(`/clerk/admin/users/${targetUserId}/sessions`)
  .set('Authorization', 'Bearer admin-token')
  .expect(200);

expect(finalSessionsResponse.body.sessions).toEqual([]);
});
});

describe('Access Control Verification E2E', () => {
  it('should prevent regular users from accessing admin endpoints', async () => {
    await request(app.getHttpServer())
      .get(`/clerk/admin/users/some-user-id/sessions`)
      .set('Authorization', 'Bearer user-token')
      .expect(403);

    expect(clerkSessionService.getSessionList).not.toHaveBeenCalled();
  });

  it('should allow regular users to manage only their own sessions', async () => {
    const userSessions = [{ id: 'user-session-1' }, { id: 'user-session-2' }];

    // User gets their own sessions
    const getUserSessionsResponse = await request(app.getHttpServer())
      .get('/clerk/sessions')
      .set('Authorization', 'Bearer user-token')
      .expect(200);

    expect(getUserSessionsResponse.body.message).toBe('Sessions retrieved

```



```

successfully');
  });
});

describe('Performance and Scalability E2E', () => {
  it('should handle concurrent admin operations', async () => {
    const userIds = ['user-1', 'user-2', 'user-3', 'user-4', 'user-5'];

    const promises = userIds.map(userId =>
      request(app.getHttpServer())
        .get(`/clerk/admin/users/${userId}/sessions`)
        .set('Authorization', 'Bearer admin-token')
    );

    const responses = await Promise.all(promises);

    responses.forEach(response => {
      expect(response.status).toBe(200);
      expect(response.body.message).toBe('User sessions retrieved successfully');
    });

    expect(clerkSessionService.getSessionList).toHaveBeenCalledTimes(userIds.length);
  });
});
});

```

Kịch bản kiểm tra: * **Complete user workflows:** Admin session management từ đầu đến cuối * **Multi-user scenarios:** Quản lý đồng thời nhiều user sessions * **Access control:** Phân quyền chính xác giữa admin và regular user * **Error resilience:** Xử lý lỗi service, network failures, invalid user IDs * **Performance testing:** Concurrent operations và large session lists * **Refactoring validation:** Xác nhận loại bỏ AdminGuard thành công

4. Thách thức và Giải pháp

4.1. Thách thức: Vi phạm Tách biệt Trách nhiệm và Trùng lặp Mã

- **Vấn đề:** Trước tái cấu trúc, `ClerkModule` vừa xử lý xác thực vừa chứa logic phân quyền (thông qua `AdminGuard` và `@AdminOnly` decorator). Điều này dẫn đến vi phạm nguyên tắc Đơn trách nhiệm (SRP), liên kết chặt chẽ (tight coupling) giữa tầng hạ tầng và logic nghiệp vụ, và trùng lặp mã với `RolesGuard` trong `AuthModule`.
- **Giải pháp:**
 - Loại bỏ hoàn toàn `AdminGuard` và `@AdminOnly` decorator khỏi `ClerkModule`.
 - Tái cấu trúc `ClerkModule` để chỉ tập trung vào xác thực và tương tác với Clerk SDK.
 - Sử dụng `RolesGuard` từ `AuthModule` một cách nhất quán cho tất cả các nhu cầu phân quyền dựa

trên vai trò.

4.2. Thách thức: **RolesGuard** Yếu và Lỗ hổng Bảo mật "Fail-Open"

- **Vấn đề:** Phiên bản **RolesGuard** ban đầu có lỗ hổng logic nghiêm trọng: nếu một endpoint không được gán decorator **@Roles**, nó sẽ tự động cho phép truy cập (**return true**). Điều này tạo ra một lỗ hổng bảo mật "fail-open" tiềm ẩn. Ngoài ra, nó chỉ hỗ trợ một vai trò duy nhất cho mỗi người dùng và thiếu kiểm tra phòng vệ, dễ gây ra lỗi 500.
- **Giải pháp:**
- Tái cấu trúc **RolesGuard** để tuân thủ nguyên tắc "fail-safe": nếu không có vai trò nào được yêu cầu rõ ràng, nó sẽ ném **ForbiddenException**, từ chối truy cập theo mặc định.
- Cải thiện logic trích xuất vai trò để hỗ trợ nhiều vai trò cho một người dùng (từ **publicMetadata.roles** hoặc **publicMetadata.role**).
- Thêm các kiểm tra phòng vệ mạnh mẽ cho đối tượng **user** và **publicMetadata** để tránh lỗi 500 và cung cấp thông báo lỗi rõ ràng hơn.
- Tích hợp **Logger** để ghi lại các sự kiện liên quan đến phân quyền, hỗ trợ gỡ lỗi và giám sát bảo mật.

4.3. Thách thức: Phụ thuộc của **ClerkController** vào **RolesGuard**

- **Vấn đề:** **ClerkController** (thuộc **ClerkModule** - tầng hạ tầng) vẫn sử dụng **RolesGuard** và **@Roles** decorator (thuộc **AuthModule** - tầng nghiệp vụ) cho các endpoint admin. Điều này tạo ra sự phụ thuộc ngược từ tầng hạ tầng lên tầng nghiệp vụ, có thể gây nhầm lẫn về kiến trúc.
- **Giải pháp (Đã chấp nhận):** Mặc dù đây là một sự phụ thuộc ngược, nó được chấp nhận trong bối cảnh hiện tại để tận dụng **RolesGuard** đã được tái cấu trúc mạnh mẽ. Các endpoint admin trong **ClerkController** được coi là có logic nghiệp vụ (quản lý người dùng khác, phiên của người dùng khác) cần được ủy quyền, và việc sử dụng **RolesGuard** là cách hiệu quả nhất để thực hiện điều đó. Nếu mục tiêu là **ClerkModule** hoàn toàn không có logic nghiệp vụ, thì cần một giải pháp khác (ví dụ: di chuyển các endpoint admin ra khỏi **ClerkController** hoặc tạo một guard phân quyền riêng biệt trong **ClerkModule** chỉ dựa trên các khái niệm hạ tầng). Tuy nhiên, với phạm vi hiện tại, giải pháp này là hợp lý và đã được kiểm thử kỹ lưỡng.

5. Cải tiến và Tối ưu hóa

5.1. Cải tiến Kiến trúc

- **Tách biệt Trách nhiệm Rõ ràng:** **ClerkModule** hiện chỉ tập trung vào xác thực và tương tác với Clerk, trong khi **AuthModule** xử lý ủy quyền và đồng bộ hóa dữ liệu người dùng cục bộ. Điều này giúp code dễ hiểu, dễ bảo trì và mở rộng.
- **Tuân thủ Nguyên tắc Thiết kế:** Việc loại bỏ **AdminGuard** và tái cấu trúc **RolesGuard** đã cải thiện

đáng kể sự tuân thủ các nguyên tắc SOLID, đặc biệt là SRP và DIP.

- **Giảm Trùng lặp Mã:** `RolesGuard` là nguồn duy nhất cho logic kiểm tra vai trò, loại bỏ sự trùng lặp với `AdminGuard` cũ.

5.2. Cải tiến Bảo mật

- **Nguyên tắc "Fail-Safe":** `RolesGuard` giờ đây từ chối truy cập theo mặc định nếu không có vai trò nào được chỉ định rõ ràng, loại bỏ lỗ hổng "fail-open".
- **Kiểm tra Phòng vệ Mạnh mẽ:** Các kiểm tra sự tồn tại của đối tượng `user` và `publicMetadata` giúp ngăn chặn lỗi 500 và cung cấp thông báo lỗi bảo mật rõ ràng hơn.
- **Logging Bảo mật:** Tích hợp `Logger` trong `RolesGuard` giúp theo dõi các nỗ lực truy cập trái phép và các vấn đề liên quan đến vai trò.

5.3. Cải tiến Khả năng Mở rộng và Bảo trì

- **Hỗ trợ Đa vai trò:** `RolesGuard` được thiết kế để hỗ trợ nhiều vai trò cho một người dùng, cho phép mở rộng hệ thống phân quyền trong tương lai mà không cần thay đổi lớn.
- **Code Sạch và Dễ đọc:** Mã nguồn được tổ chức tốt hơn, với các phương thức riêng tư để đóng gói logic, giúp dễ đọc và hiểu.
- **Kiểm thử Dễ dàng:** Sự tách biệt trách nhiệm và các kiểm thử toàn diện giúp dễ dàng kiểm thử từng thành phần một cách độc lập và tích hợp.

5.4. Tối ưu hóa Hiệu suất (Đã được phân tích trong tài liệu)

- Các tài liệu tái cấu trúc đã đề cập đến việc phân tích hiệu suất và tối ưu hóa chuỗi guard, bao gồm chiến lược caching và tối ưu hóa truy vấn cơ sở dữ liệu. Mặc dù không có thay đổi mã nguồn trực tiếp trong các file được review, việc phân tích này cho thấy sự quan tâm đến hiệu suất.

6. Công cụ và Công nghệ Sử dụng

- **Ngôn ngữ Lập trình:** TypeScript
- **Framework:** NestJS
- **Thư viện Xác thực:** Clerk SDK (`@clerk/clerk-sdk-node`)
- **Thư viện Kiểm thử:** Jest, Supertest
- **Công cụ:**
 - `@nestjs/core: Reflector` (để đọc metadata từ decorator)
 - `@nestjs/common: Logger, Injectable, CanActivate, ExecutionContext, ForbiddenException, InternalServerErrorException`
 - `@nestjs/swagger`: Để tạo tài liệu API

- **mermaid**: Để tạo sơ đồ kiến trúc và luồng dữ liệu trong tài liệu

7. Kết luận

Việc tái cấu trúc module Auth và Clerk trong dự án TheShoeBolt đã đạt được thành công đáng kể. Hệ thống hiện tại có một kiến trúc xác thực và phân quyền rõ ràng, mạnh mẽ và có khả năng mở rộng. Các vấn đề về vi phạm kiến trúc, trùng lặp mã và lỗi hỏng bảo mật đã được giải quyết một cách hiệu quả.

Mặc dù có một sự phụ thuộc nhỏ từ **ClerkController** (tầng hạ tầng) vào **RolesGuard** (tầng nghiệp vụ), điều này được coi là chấp nhận được trong bối cảnh hiện tại để tận dụng **RolesGuard** đã được cải tiến.

Bộ kiểm thử toàn diện đảm bảo rằng các thay đổi đã được thực hiện một cách đáng tin cậy và hệ thống hoạt động đúng như mong đợi. Đây là một nền tảng vững chắc cho việc phát triển các tính năng liên quan đến xác thực và phân quyền trong tương lai.

7.1. Khuyến nghị Tiếp theo

- **Xem xét lại phụ thuộc ClerkController**: Trong dài hạn, nếu mục tiêu là **ClerkModule** hoàn toàn không có logic nghiệp vụ, cần xem xét di chuyển các endpoint admin ra khỏi **ClerkController** hoặc tạo một cơ chế ủy quyền khác không phụ thuộc vào **AuthModule**.
- **Tối ưu hóa hiệu suất**: Tiếp tục theo dõi và tối ưu hóa hiệu suất của chuỗi guard và các tương tác với Clerk API, đặc biệt là việc triển khai caching token như đã đề cập trong các tài liệu phân tích.
- **Mở rộng vai trò**: Khi có yêu cầu nghiệp vụ, tận dụng khả năng hỗ trợ đa vai trò của **RolesGuard** để triển khai các vai trò phức tạp hơn.