Báo Cáo Code Review: Module Auth và Clerk

Table of Contents

1. 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. 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. 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. 2.1.1. `auth.module.ts`

```
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 '../Infracstructre/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. 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 t Clerk vào local database
   * Đc gi 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 đc qun 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 t local database
   * Dùng cho các endpoints cn thông tin user t DB local
   */
  async getUserProfile(userId: string) {
    return this.usersService.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. 2.1.3. `auth.controller.ts`

```
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 '../Infracstructre/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. 2.1.4. `roles.decorator.ts`

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

// Export constant đ tránh magic strings và đm bo tính nht 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. 2.1.5. roles.guard.ts

```
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';

// Đ□nh nghĩa m□t ki□u cho payload c□a ng□□i dùng t□ Clerk đ□ tăng tính an toàn v□ ki□u
interface ClerkUserPayload {
  publicMetadata?: {
    role?: UserRole; // H□ tr□ vai trò đ□n l□ nh□ hi□n t□i
    roles?: UserRole[]; // H□ tr□ m□ng các vai trò cho t□□ng lai
  };
  // Thêm các thu□c tính khác c□a user n□u c□n
  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 {
    // S□ d□ng getAllAndOverride đ□ l□y các vai trò t□ c□ handler và class
    const requiredRoles = this.reflector.getAllAndOverride<UserRole[]>(ROLES_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    // N□u không có vai trò nào đ□□c yêu c□u, áp d□ng nguyên t□c fail-safe
    // Guard này ch□ nên đ□□c kích ho□t trên các endpoint CÓ decorator @Roles.
    // Các endpoint công khai nên đ□□c x□ lý b□i m□t @Public decorator và m□t AuthGuard
toàn c□c.
    if (!requiredRoles || requiredRoles.length === 0) {
      this.logger.warn('RolesGuard đ□□c áp d□ng cho endpoint không có @Roles decorator.
T□ ch□i truy c□p theo nguyên t□c 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. Ki□m tra phòng v□: Đ□m b□o `user` t□n t□i
    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 xu□t vai trò c□a ng□□i dùng m□t cách an toàn
    const userRoles = this.extractUserRoles(user);

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

    // 4. Th□c hi□n so kh□p vai trò
    const hasPermission = this.matchRoles(requiredRoles, userRoles);

    if (!hasPermission) {
      this.logger.warn(`User ${user.id || 'unknown'} v□i roles [${userRoles.join(',
')}] không có quy□n truy c□p endpoint yêu c□u roles [${requiredRoles.join(', ')}].`);
      throw new ForbiddenException('You do not have the required permissions to access
this resource.');
    }

    this.logger.debug(`User ${user.id || 'unknown'} đ□□c phép truy c□p v□i roles
[${userRoles.join(', ')}].`);
    return true;
  }

  /**
   * Trích xu□t danh sách vai trò c□a ng□□i dùng t□ Clerk payload
   * H□ tr□ c□ đ□nh d□ng cũ (role đ□n l□) và đ□nh d□ng m□i (roles array)
   * @param user Clerk user payload
   * @returns M□ng các vai trò c□a ng□□i dùng
   */
  private extractUserRoles(user: ClerkUserPayload): UserRole[] {
    if (!user.publicMetadata) {
      return [];
    }

    // □u tiên s□ d□ng roles array n□u có (cho t□□ng lai)
    if (user.publicMetadata.roles && Array.isArray(user.publicMetadata.roles)) {
      return user.publicMetadata.roles;
    }

    // Fallback sang role đ□n l□ (hi□n t□i)
```

```
    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 thừa kế 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. 2.2. Module `Clerk` (`src/modules/Infracstructre/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. 2.2.1. `clerk.module.ts`

```
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. 2.2.2. clerk.controller.ts

```
import {
  Controller,
  Get,
  Delete,
  Post,
  Param,
  UseGuards,
  Request,
  HttpCode,
  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. 2.2.3. clerk.session.service.ts

```
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. 2.2.4. `clerk-auth.guard.ts`

```
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. 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.

3.1. 3.1. Kiểm thử Đơn vị (Unit Tests)

- `test/unit/modules/auth/guards/roles.guard.spec.ts`:
- Kiểm tra chi tiết logic của `RolesGuard`, bao gồm việc trích xuất vai trò từ các định dạng khác nhau của `publicMetadata` (vai trò đơn lẻ và mảng vai trò), xử lý các trường hợp `user` hoặc `publicMetadata` bị thiếu, và đảm bảo nguyên tắc "fail-safe" được thực thi khi không có vai trò nào được yêu cầu.
- Xác nhận rằng các ngoại lệ `ForbiddenException` và `InternalServerErrorException` được ném ra đúng cách với thông báo rõ ràng.
- `test/unit/modules/Infracstructre/clerk/clerk.controller.spec.ts`:
- Kiểm tra các endpoint của `ClerkController`, đặc biệt là các endpoint admin.
- Xác nhận rằng các endpoint admin được bảo vệ bởi `RolesGuard` và `ClerkAuthGuard` một cách chính xác.
- Đảm bảo rằng `AdminGuard` và các decorator liên quan đã được loại bỏ hoàn toàn khỏi controller.
- `test/unit/modules/Infracstructre/clerk/clerk.module.spec.ts`:
- Xác minh cấu hình của `ClerkModule`, đảm bảo nó cung cấp đúng các service và guard (`ClerkSessionService`, `ClerkAuthGuard`).
- Quan trọng hơn, nó kiểm tra rằng `AdminGuard` không còn được export hoặc cung cấp bởi module, củng cố sự tách biệt trách nhiệm giữa xác thực và phân quyền.

3.2. 3.2. Kiểm thử Tích hợp (Integration Tests)

- `test/integration/clerk-admin-endpoints.integration.spec.ts`:
- Kiểm thử luồng xác thực và ủy quyền đầy đủ cho các endpoint admin của Clerk.
- Mô phỏng các yêu cầu HTTP và xác nhận rằng các yêu cầu từ người dùng không được xác thực hoặc không có quyền admin bị từ chối một cách chính xác (HTTP 401/403).
- Kiểm tra thứ tự thực thi của các guard: `ClerkAuthGuard` luôn chạy trước `RolesGuard`.
- Xác nhận rằng các service method của `ClerkSessionService` được gọi đúng cách khi truy cập được cấp phép.

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

- `test/e2e/clerk-admin-e2e.spec.ts`:
- Mô phỏng các kịch bản người dùng thực tế, bao gồm các luồng quản lý phiên admin (xem, thu hồi phiên cụ thể, thu hồi tất cả các phiên).
- Xác nhận rằng người dùng thông thường không thể truy cập các chức năng admin, trong khi người dùng admin có thể thực hiện các tác vụ của họ.
- Bao gồm các kịch bản xử lý lỗi (ví dụ: dịch vụ Clerk không khả dụng, lỗi mạng) và kiểm tra hiệu suất dưới tải đồng thời.
- Xác minh rằng việc loại bỏ `AdminGuard` và thay thế bằng `RolesGuard` mới hoạt động liền mạch

trong môi trường end-to-end.

4. 4. Thách thức và Giải pháp

4.1. 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. 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. 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. 5. Cải tiến và Tối ưu hóa

5.1. 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. 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. 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. 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. 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. 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 lỗ hổng 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. 7.1. Khuyến nghị Tiếp theo

- **Sửa lỗi chính tả**: Đổi tên thư mục `src/modules/Infracstructre` thành `src/modules/Infrastructure`.
- **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.