Báo cáo: Tích hợp Clerk Authentication trong NestJS

1. Tổng quan

Clerk là một dịch vụ xác thực và quản lý người dùng hiện đại, cung cấp các tính năng như đăng nhập, đăng ký, quản lý phiên và bảo mật. Trong dự án này, Clerk được tích hợp vào cả phía frontend (React) và backend (NestJS).

2. Cấu hình Backend (NestJS)

2.1. Cài đặt Dependencies

Đầu tiên, cần cài đặt các package cần thiết:

```
npm install @clerk/backend @nestjs/passport passport passport-custom
```

2.2. Cấu hình Environment Variables

Tạo file `.env` trong thư mục gốc của dự án và thêm các biến môi trường sau:

```
CLERK_PUBLISHABLE_KEY=your_publishable_key
CLERK_SECRET_KEY=your_secret_key
```

2.3. Tích hợp Clerk vào NestJS

2.3.1. Clerk Client Provider

File: `apps/api/src/providers/clerk-client.provider.ts`

```
import { createClerkClient } from '@clerk/backend';
import { ConfigService } from '@nestjs/config';

export const ClerkClientProvider = {
  provide: 'ClerkClient',
  useFactory: (configService: ConfigService) => {
    return createClerkClient({
      publishableKey: configService.get('CLERK_PUBLISHABLE_KEY'),
      secretKey: configService.get('CLERK_SECRET_KEY'),
    });
  },
  inject: [ConfigService],
};
```

Provider này tạo một instance của Clerk client để sử dụng trong toàn bộ ứng dụng.

2.3.2. Clerk Strategy

File: `apps/api/src/auth/clerk.strategy.ts`

```
import { User, verifyToken } from '@clerk/backend';
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy } from 'passport-custom';
import { UsersService } from 'src/users/users.service';
import { Request } from 'express';

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

  async validate(req: Request): Promise<User> {
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
      throw new UnauthorizedException('No token provided');
    }

    try {
      const tokenPayload = await verifyToken(token, {
        secretKey: this.configService.get('CLERK_SECRET_KEY'),
      });

      const user = await this.usersService.getUser(tokenPayload.sub);
      return user;
    } catch (error) {
      console.error(error);
      throw new UnauthorizedException('Invalid token');
    }
  }
}
```

Strategy này xử lý việc xác thực token từ Clerk và trả về thông tin người dùng.

2.3.3. Clerk Auth Guard

File: `apps/api/src/auth/clerk-auth.guard.ts`

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

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

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

    if (isPublic) {
      return true;
    }

    return super.canActivate(context);
  }
}
```

Guard này bảo vệ các route trong ứng dụng, chỉ cho phép truy cập khi có token hợp lệ.

2.4. Cấu hình Module

2.4.1. Auth Module

File: `apps/api/src/auth/auth.module.ts`

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

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

2.4.2. App Module

File: `apps/api/src/app/app.module.ts`

```
import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { ClerkClientProvider } from '../providers/clerk-client.provider';
import { UsersModule } from '../users/users.module';
import { AuthModule } from '../auth/auth.module';
import { ClerkAuthGuard } from '../auth/clerk-auth.guard';
import { APP_GUARD } from '@nestjs/core';
import { AppController } from './app.controller';

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    UsersModule,
    AuthModule,
  ],
  providers: [
    ClerkClientProvider,
    {
      provide: APP_GUARD,
      useClass: ClerkAuthGuard,
    },
  ],
  controllers: [AppController],
})
export class AppModule {}
```

3. Cấu hình Frontend (React)

3.1. Cài đặt Dependencies

```
npm install @clerk/clerk-react
```

3.2. Cấu hình Environment Variables

Tạo file `.env` trong thư mục frontend:

```
VITE_CLERK_PUBLISHABLE_KEY=your_publishable_key
```

3.3. Tích hợp Clerk Provider

File: `apps/web/src/main.tsx`

```tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.tsx';
import './index.css';
import { ClerkProvider } from '@clerk/clerk-react';

const PUBLISHABLE_KEY = import.meta.env.VITE_CLERK_PUBLISHABLE_KEY;

if (!PUBLISHABLE_KEY) {
  throw new Error('Missing Publishable Key');
}

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <ClerkProvider publishableKey={PUBLISHABLE_KEY} afterSignOutUrl='/'>
      <App />
    </ClerkProvider>
  </React.StrictMode>
);
```

3.4. Sử dụng Clerk trong Components

File: `apps/web/src/App.tsx`

```
import {
  SignedIn,
  SignedOut,
  SignInButton,
  SignUpButton,
  useAuth,
  UserButton,
} from '@clerk/clerk-react';
import { useState } from 'react';
import axios from 'axios';

const axiosInstance = axios.create({
  baseURL: 'http://localhost:4000',
});

export default function App() {
  const { getToken } = useAuth();
  const [loading, setLoading] = useState(false);

  const handleExternalApi = async () => {
    setLoading(true);
    const token = await getToken();
    try {
      const response = await axiosInstance.get('/auth/me', {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });
      console.log(response);
    } catch (error) {
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className='w-screen h-screen flex flex-col items-center justify-center'>
      <SignedOut>
```

```
        <div className='flex flex-col gap-4'>
          <SignInButton />
          <SignUpButton />
        </div>
      </SignedOut>
      <SignedIn>
        <div className='flex flex-col gap-4 items-center justify-center'>
          <button onClick={handleExternalApi}>
            {loading ? 'Loading...' : 'Ping external API'}
          </button>
          <UserButton />
        </div>
      </SignedIn>
    </div>
  );
}
```

4. Cách sử dụng

4.1. Bảo vệ Routes

Để bảo vệ một route trong NestJS, sử dụng `@UseGuards(ClerkAuthGuard)`:

```
@Controller('protected')
@UseGuards(ClerkAuthGuard)
export class ProtectedController {
  @Get()
  getProtectedData() {
    return { message: 'This is protected data' };
  }
}
```

4.2. Đánh dấu Route là Public

Để đánh dấu một route là public (không cần xác thực), sử dụng decorator `@Public()`:

```
@Controller('public')
export class PublicController {
  @Public()
  @Get()
  getPublicData() {
    return { message: 'This is public data' };
  }
}
```

4.3. Truy cập Thông tin Người dùng

Trong controller, bạn có thể truy cập thông tin người dùng thông qua `@Request()` decorator:

```
@Controller('user')
@UseGuards(ClerkAuthGuard)
export class UserController {
  @Get('profile')
  getProfile(@Request() req) {
    return req.user;
  }
}
```

5. Best Practices

1. **Bảo mật Environment Variables**: Luôn sử dụng biến môi trường cho các khóa API của Clerk.
2. **Xử lý Lỗi**: Luôn xử lý các trường hợp lỗi khi xác thực token.
3. **Middleware**: Sử dụng middleware để xử lý các request trước khi chúng đến controller.
4. **Type Safety**: Sử dụng TypeScript để đảm bảo type safety cho các object liên quan đến người dùng.
5. **Error Handling**: Implement proper error handling cho cả frontend và backend.

6. Kết luận

Việc tích hợp Clerk vào dự án NestJS cung cấp một giải pháp xác thực mạnh mẽ và dễ sử dụng. Với cấu trúc được trình bày ở trên, bạn có thể dễ dàng:

- Xác thực người dùng
- Bảo vệ các route
- Quản lý phiên đăng nhập
- Tích hợp với frontend React

Báo cáo này cung cấp một hướng dẫn toàn diện về cách tích hợp và sử dụng Clerk trong dự án NestJS của bạn.