**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN TỐT NGHIỆP**

**ĐỀ TÀI: TÌM HIỂU THUẬT TOÁN MÃ HÓA BẤT ĐỐI XỨNG VÀ ỨNG DỤNG XÂY DỰNG HỆ THỐNG ĐĂNG NHẬP MỘT LẦN (SINGLE SIGN ON)**

Giảng viên hướng dẫn : Ths. Trần Phong Nhã

Sinh viên thực hiện : PHAN ĐÌNH TRÍ

Mã sinh viên : 5751071046

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá :K57

Tp. Hồ Chí Minh, năm 2021

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI **CỘNG HÒA XÃ HỘI CHỦ NGHIÃ VIỆT NAM**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH** Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

-------\*\*\*-------

**Mã sinh viên**: 5751071046 **Họ tên SV**: Phan Đình Trí

**Khóa**: 57 **Lớp**: Công nghệ thông tin

1. **Tên đề tài**

Tìm hiểu thuật toán mã hóa bất đối xứng và ứng dụng xây dựng hệ thống đăng nhập một lần (Single sign-on).

1. **Mục đích, yêu cầu**

* Mục đích:

“Tìm hiểu thuật toán mã hóa bất đối xứng và ứng dụng xây dựng hệ thống đăng nhập một lần” cung cấp các dịch vụ xác thực người dùng, cho phép người dùng có thể truy cập nhiều trang web.

* Yêu cầu chức năng:

+ Cung cấp cơ chế xác thực yêu cầu người dùng đăng nhập vào chỉ một lần với một tài khoản và mật khẩu để truy cập vào nhiều ứng dụng trong một phiên làm việc.

+ Cung cấp các phương thức truy cập thông tin tài khoản người dùng

+ Xác thực người dùng, xác thực hai yếu tố.

+ Phân quyền truy cập người dùng tập trung.

+ Quản lý thông tin người dùng, thông tin ứng dụng sử dụng dịch vụ.

+ Đăng xuất người dùng ở tất cả các ứng dụng.

+ Ngoài phương thức đăng nhập thông qua tài khoản của hệ thống, có thể tích hợp các phương thức đăng nhập từ các trang web tin cậy như Google, Facebook.

* Yêu cầu phi chức năng:

+ Tiết kiệm thời gian cho người sử dụng trong việc đăng nhập vào nhiều dịch vụ được cung cấp trên các nền tảng khác nhau của hệ thống phân tán.

+ Tăng cường khả năng bảo mật thông qua việc giúp người dùng không cần nhớ nhiều thông tin đăng nhập.

+ Tăng cường khả năng bảo mật cho các trang web, api có sử dụng hệ thống để xác thực quyền truy cập của người dùng.

+ Cung cấp khả năng phân quyền tập trung cho tất cả người dùng.

1. **Nội dung thực hiện**

* Tìm hiểu về thuật toán mã hóa RSA, Web Service, Json Web Token, Identity Server, xác thực Single Sign On.
* Thiết kế và xây dựng hệ thống đăng nhập một lần.
* Viết báo cáo.

1. **Công nghệ, công cụ và ngôn ngữ lập trình**

* Công nghệ sử dụng: .Net core 3.1, SQLServer 2019
* Công cụ: Visual Studio 2019, Visual Studio Code, SQL Management Studio.
* Thuật toán mã hóa bất đối xứng RSA.
* Ngôn ngữ lập trình:

+ Back-end: C#, JavaScript/TypeScript

+ Front-end: Angular 10, React, ASP.NET Core MVC.

* Triển khai: Docker.

1. **Các kết quả chính dự kiến sẽ đạt được và ứng dụng**

* Tìm hiểu về Microservices, Json Web Token, thuật toán mã hóa bất đối xứng RSA, Identity Server, xác thực Single Sign On.
* Hoàn thành và xây dựng hệ thống đăng nhập một lần.

1. **Giảng viên và cán bộ hướng dẫn**

Họ tên: Trần Phong Nhã

Đơn vị công tác: Bộ môn Công nghệ thông tin – Trường Đại học Giao thông Vận Tải Phân hiệu TP.Hồ Chí Minh

Điện thoại: 0906 761 014 Email: tpnha@utc2.edu.vn

|  |  |
| --- | --- |
| **Ngày ….. tháng ….. năm 2021**  **Trưởng BM Công nghệ Thông tin** | **Đã giao nhiệm vụ TKTN**  **Giảng viên hướng dẫn** |
| **ThS. Trần Phong Nhã** | **Ths. Trần Phong Nhã** |

Đã nhận nhiệm vụ TKTN

Sinh viên: Phan Đình Trí Ký tên:

Điện thoại:0965673755 Email: [5751071046@st.utc2.edu.vn](mailto:5751071046@st.utc2.edu.vn)

**LỜI CẢM ƠN**

Lời đầu tiên em xin phép gửi đến Quý Thầy Cô Trường Đại học Giao thông Vận Tải Phân hiệu TP.Hồ Chí Minh lời chúc sức khỏe và lời cảm ơn sâu sắc nhất. Thầy cô đã tận tình hướng dẫn, truyền đạt kiến thức, kinh nghiệm cho em trong suốt quá trình học tập và rèn luyện ở trường. Đặc biệt, em xin gửi lời cảm ơn đến thầy **Trần Phong Nhã** đã tận tình hướng dẫn chỉ bảo để giúp em hiểu một số công nghệ và các chức năng cần có với một bài toán trong thực tế để hoàn thành đồ án **“Tìm hiểu thuật toán mã hóa bất đối xứng và ứng dụng xây dựng hệ thống đăng nhập một lần”.** Nhờ sự hướng dẫn tận tình của thầy mà em có thể khắc phục nhiều khuyết điểm của bản thân trong quá trình xây dựng đồ án.

Mặc dù em đã cố gắng hết sức trong quá trình thực hiện đồ án này, nhưng vì thời gian có hạn và em cũng chưa có kinh nghiệm trong làm việc với bài toán thực tế, nên chắc chắn sẽ có nhiều thiếu sót không thể tránh. Em rất mong nhận được sự cảm thông và góp ý của quý thấy quý cô để giúp em có thể phấn đấu và hoàn thiện hơn trong tương lai.

Lời sau cùng, em xin kính chúc thầy cô trong nhà trường và các thầy cô trong bộ môn Công nghệ thông tin – Trường đại học Giao thông Vận tải Phân hiệu tại thành phố Hồ Chí Minh luôn mạnh khỏe và thu được nhiều thành công hơn nữa trong tương lai.

Em xin chân thành cảm ơn!

Tp.Hồ Chí Minh, ngày tháng năm 2021

Sinh viên thực hiện

Phan Đình Trí

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày ….… tháng ….… năm 2021***  **Giảng viên hướng dẫn**  **Ths. Trần Phong Nhã** |

**MỤC LỤC**

[CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI 1](#_Toc90849157)

[1.1 Lý do chọn đề tài 1](#_Toc90849158)

[1.2 Nội dung và phạm vi đề tài 1](#_Toc90849159)

[1.2.1 Nội dung 1](#_Toc90849160)

[1.2.2 Phạm vi đề tài 1](#_Toc90849161)

[1.3 Cấu trúc báo cáo đồ án tốt nghiệp 2](#_Toc90849162)

[CHƯƠNG 2: CƠ SỞ LÝ THUYẾT 3](#_Toc90849163)

[2.1 Tổng quan về .NET core 3](#_Toc90849164)

[2.1.1 Giới thiệu 3](#_Toc90849165)

[2.1.2 Đặc điểm cốt lõi của .NET Core 4](#_Toc90849166)

[2.1.3 Thành phần cốt lõi của .NET Core 5](#_Toc90849167)

[2.2 Ngôn ngữ lập trình C# 5](#_Toc90849168)

[2.2.1 Giới thiệu ngôn ngữ C# 5](#_Toc90849169)

[2.2.2 Đặc trưng của ngôn ngữ C# 6](#_Toc90849170)

[2.3 Tổng quan về Angular Framework 7](#_Toc90849171)

[2.3.1 Giới thiệu Angular Framework 7](#_Toc90849172)

[2.3.2 Những tính năng nổi bật của Angular Framework 7](#_Toc90849173)

[2.4 Tổng quan về thuật toán mã hóa bất đối xứng RSA 8](#_Toc90849174)

[2.5 Tổng quan về JSON Web Token (JWT) 9](#_Toc90849175)

[2.5.1 JSON Web Token là gì? 9](#_Toc90849176)

[2.5.2 Thành phần của một JWT 9](#_Toc90849177)

[CHƯƠNG 3: PHÂN TÍCH THUẬT TOÁN MÃ HÓA BẤT ĐỐI XỨNG RSA 13](#_Toc90849178)

[3.1 Mô tả sơ lược 13](#_Toc90849179)

[3.2 Mô tả quá trình hoạt động của RSA 13](#_Toc90849180)

[3.3 Ứng dụng của thuật toán mã hóa bất đối xứng RSA 16](#_Toc90849181)

[3.3.1 Ứng dụng trong bảo mật dữ liệu 16](#_Toc90849182)

[3.3.2 Ứng dụng trong công nghệ thông tin 17](#_Toc90849183)

[CHƯƠNG 4: TÌM HIỂU VỀ IDENTITY SERVER 18](#_Toc90849184)

[4.1 Giới thiệu 18](#_Toc90849185)

[4.1.1 Xác thực người dùng 19](#_Toc90849186)

[4.1.2 Ủy quyền truy cập API 19](#_Toc90849187)

[4.1.3 Kết hợp OpenID Connect và OAuth 2.0 19](#_Toc90849188)

[4.2 Identity Server 20](#_Toc90849189)

[4.2.1 Đăng nhập một lần (Single Sign On – SSO) 20](#_Toc90849190)

[4.2.2 Xác thực (Authentication) 22](#_Toc90849191)

[4.2.3 Kiểm soát truy cập - ủy quyền (Authorization) 22](#_Toc90849192)

[4.3 Identity Server dựa trên kiến trúc một khối 22](#_Toc90849193)

[4.4 Identity Server dựa trên kiến trúc Microservices 25](#_Toc90849194)

[4.5 Xác định, xây dựng Identity Server dựa trên kiến trúc Microservices 27](#_Toc90849195)

[4.5.1 Microservices Core 27](#_Toc90849196)

[4.5.2 API Gateway 28](#_Toc90849197)

[4.5.3 Các ứng dụng phía client 30](#_Toc90849198)

[CHƯƠNG 5: XÂY DỰNG ỨNG DỤNG 31](#_Toc90849199)

[5.1 Giao diện hệ thống 31](#_Toc90849200)

[5.2 Đăng nhập vào ứng dụng của hệ thống thông qua Identity Server 32](#_Toc90849201)

[5.3 Cấu hình ứng dụng client mới 34](#_Toc90849202)

[5.4 Bảo vệ API bằng Identity Server 39](#_Toc90849203)

[5.4.1 Bảo vệ tài nguyên API 39](#_Toc90849204)

[5.4.2 Các quy định về phạm vi truy cập tài nguyên API 42](#_Toc90849205)

[5.5 Tài nguyên nhận dạng 44](#_Toc90849206)

[5.6 Quản lý người dùng 44](#_Toc90849207)

[5.7 Quản lý phân quyền người dùng 47](#_Toc90849208)

[CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 49](#_Toc90849209)

[6.1 Kết quả đạt được 49](#_Toc90849210)

[6.2 Các vấn đề còn tồn tại 49](#_Toc90849211)

[6.3 Đề xuất hướng phát triển 49](#_Toc90849212)

**DANH MỤC CHỮ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Mô tả** | **Ý nghĩa** | **Ghi chú** |
| 1 | SSO | Single Sign On |  |
| 2 | API | Application Programming Interface |  |
| 3 | JWT | Json Web Token |  |
| 4 | CLI | Command line interface |  |
| 5 | IDE | Integrated Development Environment |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**DANH MỤC BẢNG**

[Bảng 4.1 So sánh Identity Server dựa trên hai kiến trúc. 27](#_Toc90849213)

[Bảng 4.2 Xác định các microservices và chức năng. 28](#_Toc90849214)

[Bảng 5.1 Thông số của ứng dụng client 38](#_Toc90849215)

[Bảng 5.2 Thông số kỹ thuật bảo vệ API bằng Identity Server 41](#_Toc90849216)

[Bảng 5.3 Các thông số xác định phạm vi truy cập API 44](#_Toc90849217)

**DANH MỤC HÌNH VẼ**

[Hình 2.1 Tổng quan .Net core 4](#_Toc90849218)

[Hình 2.2 Các thành phần cơ bản của .NET Core 5](#_Toc90849219)

[Hình 2.3 Tính năng của Angular 8](#_Toc90849220)

[Hình 2.4 Chuỗi JWT 10](#_Toc90849221)

[Hình 2.5 Header JWT 10](#_Toc90849222)

[Hình 2.6 Payload JWT 11](#_Toc90849223)

[Hình 2.7 Signature JWT 12](#_Toc90849224)

[Hình 3.1 Sơ đồ tạo khóa, mã hóa và giải mã RSA 15](#_Toc90849225)

[Hình 4.1 Tổng quát Identity Server 18](#_Toc90849226)

[Hình 4.2 Single Sign On 21](#_Toc90849227)

[Hình 4.3 Xác thức (Authentication) 22](#_Toc90849228)

[Hình 4.4 Identity Server dựa trên kiến trúc nguyên khối. 23](#_Toc90849229)

[Hình 4.5 Cấu trúc dự án Identity Server dựa theo kiến trúc nguyên khối. 24](#_Toc90849230)

[Hình 4.6 Identity Server dựa trên kiến trúc Microservices. 26](#_Toc90849231)

[Hình 4.7 Cấu hình một dịch vụ khi sử dụng Nginx làm chức năng câng bằng tải. 28](#_Toc90849232)

[Hình 4.8 API Gateway Ocelot cho ASP.NET Core 29](#_Toc90849233)

[Hình 4.9 Khai báo một API với API Gateway Ocelot bằng file Configuration.json. 30](#_Toc90849234)

[Hình 5.1 Trang chủ Identity Server 31](#_Toc90849235)

[Hình 5.2 Trang chủ ứng dụng Admin Dashboard 31](#_Toc90849236)

[Hình 5.3 Trang Swagger quản lý API Identity Microservice 32](#_Toc90849237)

[Hình 5.4 Đăng nhập vào ứng dụng client 32](#_Toc90849238)

[Hình 5.5 Đăng nhập vào ứng dụng thông qua Identity Server 33](#_Toc90849239)

[Hình 5.6 Mã Token được gửi về ứng dụng client 34](#_Toc90849240)

[Hình 5.7 Quy trình đăng nhập 39](#_Toc90849241)

[Hình 5.8 Bảo vệ API bằng Identity Server 40](#_Toc90849242)

[Hình 5.9 Trình xử lý mã token xác thực 42](#_Toc90849243)

[Hình 5.10 Trình xử lý mã token ủy quyền 42](#_Toc90849244)

[Hình 5.11 Xác định phạm vi API được bảo vệ 43](#_Toc90849245)

[Hình 5.12 Danh sách thông tin người dùng 45](#_Toc90849246)

[Hình 5.13 Thêm mới người dùng 46](#_Toc90849247)

[Hình 5.14 Chỉnh sửa thông tin người dùng 47](#_Toc90849248)

[Hình 5.15 Quản lý thông tin các nhóm phân quyền 48](#_Toc90849249)

[Hình 5.16 Chỉnh sửa thông tin chi tiết ủy quyền truy cập API 48](#_Toc90849250)

[Hình 5.17 Chỉnh sửa thông tin chi tiết ủy quyền truy cập ứng dụng client 48](#_Toc90849251)

# TỔNG QUAN ĐỀ TÀI

## Lý do chọn đề tài

* Ngày nay, việc sở hữu và vận hành nhiều ứng dụng Web đã trở nên phổ biến, đặc biệt là các tổ chức lớn. Các ứng dụng độc lập với nhau, có thể được phát triển trên những nền tảng khác nhau. Người dùng phải thực hiện đăng ký thông tin và đăng nhập lại khi chuyển sang ứng dụng khác.

Tổ chức càng nhiều ứng dụng, người dùng càng phải nhớ một lượng lớn tên đăng nhập và mật khẩu khác nhau. Công việc này chiếm một khoản thời gian lớn của người dùng, làm giảm hiệu suất phục vụ của toàn hệ thống. Các nhà phát triển phải giải quyết hàng loạt vấn đề liên quan đến quản lý cơ sở dữ liệu người dùng và bảo mật ở các hệ thống khác nhau. Để giảm bớt thao tác của người dùng, tăng tính chuyên nghiệp cho hệ thống, xây dựng hệ thống đăng nhập một lần (Single Sign On – SSO) được đánh giá là giải pháp tốt nhất hiện nay.

Đăng nhập một lần là quá trình xác thực phiên làm việc của người dùng, cho phép sử dụng một tên đăng nhập và mật khẩu để truy cập vào nhiều ứng dụng. Sau khi được xác thực, người dùng có thể truy cập các ứng dụng khác mà không phải đăng nhập lại, các quyền được trao và thông tin người dùng được lưu giữ trong suốt phiên làm việc. Như vậy, hệ thống ĐNML vừa đảm bảo tính thuận tiện vừa tăng độ bảo mật khi sử dụng.

* Trên đây là những lý do để thực hiện đề tài “Tìm hiểu về Microservices, thuật toán mã hóa bất đối xứng và ứng dụng xây dựng hệ thống đăng nhập một lần (Single Sign On)”.

## Nội dung và phạm vi đề tài

### Nội dung

* Tìm hiểu thuật toán mã hóa RSA, Json Web Token, Identity Server, Single Sign On.
* Xây dựng hệ thống đăng nhập một lần:

+ Cung cấp cơ chế xác thực yêu cầu người dùng đăng nhập vào chỉ một lần với một tài khoản và mật khẩu để truy cập vào nhiều ứng dụng trong một phiên làm việc.

+ Cung cấp các phương thức truy cập thông tin tài khoản người dùng.

+ Xác thực người dùng, xác thực hai yếu tố.

+ Phân quyền truy cập người dùng tập trung.

+ Quản lý thông tin người dùng, thông tin ứng dụng sử dụng dịch vụ.

+ Đăng xuất người dùng ở tất cả các ứng dụng.

+ Ngoài phương thức đăng nhập thông qua tài khoản của hệ thống, có thể tích hợp các phương thức đăng nhập từ các trang web tin cậy như Google, Facebook.

* Kiểm tra và chạy thử.

### Phạm vi đề tài

* Quản lý các đối tượng có trong các chức năng được nêu ra trong phần nội dung.

## Cấu trúc báo cáo đồ án tốt nghiệp

Cấu trúc đồ án được chia thành các chương như sau:

* Chương 1: Tổng quan đề tài
* Chương 2: Cơ sở lý thuyết
* Chương 3: Tìm hiểu về thuật toán mã hóa bất đối xứng RSA
* Chương 4: Tìm hiểu về Identity Server
* Chương 5: Xây dựng ứng dụng
* Chương 6: Kết luận và hướng phát triển

`

# CƠ SỞ LÝ THUYẾT

## Tổng quan về .NET core

### Giới thiệu

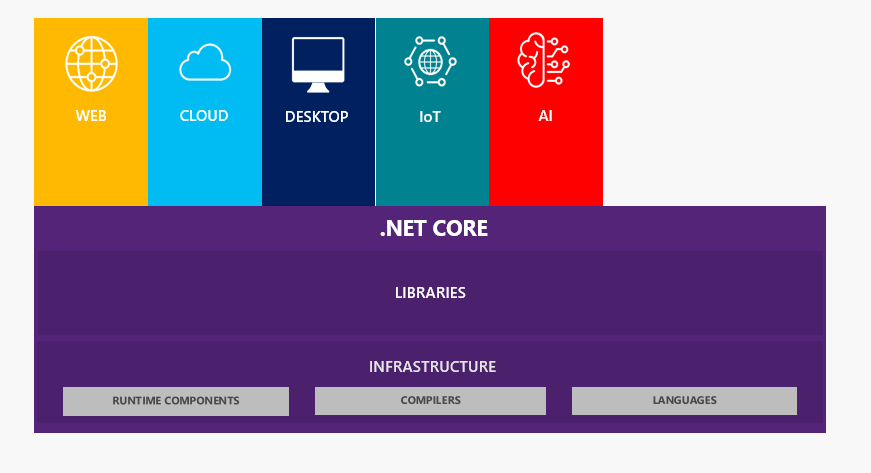
.NET Core là phiên bản mới nhất của .NET Framework, là một nền tảng phát triển miễn phí, mã nguồn mở được xây dựng và phát triển bởi Microsoft và cộng đồng .NET trên Github. .NET Core là một cross-platform chạy trên các hệ điều hành Windows, MacOs và Linux.

.NET Core có thể được sử dụng để xây dựng các ứng dụng khác nhau như ứng dụng Mobile, Web, Cloud, IoT, Machine Learning, Microservice, Game v.v…

.NET Core được xây dựng lại từ đầu để làm cho nó trở thành một Framework môđun, nhẹ, nhanh và đa nền tảng. Bao gồm các tính năng cốt lõi được yêu cầu để chạy một ứng dụng .NET Core cơ bản. Các tính năng khác được cung cấp dưới dạng gói NuGet, nhà phát triển có thể thêm nó vào ứng dụng nếu cần. Bằng cách này, các ứng dụng .NET Core có thể tăng hiệu suất, giảm dung lượng bộ nhớ và trở nên dễ bảo trì.

Có một số hạn chế với .NET Framework. Ví dụ, nó chỉ chạy trên nền tảng Windows. Ngoài ra, nhà phát triển cần sử dụng các API .NET khác nhau cho các thiết bị Windows khác nhau như Windows Desktop, Windows Store, Windows Phone và các ứng dụng Web. Ngoài ra, bất kỳ thay đổi được thực hiện trên .NET Framework đều ảnh hưởng đến tất cả các ứng dụng phụ thuộc vào nó.

Ngày nay, việc có một ứng dụng chạy trên các thiết bị là điều phổ biến; chương trình phụ trợ trên máy chủ Web, giao diện người dung quản trị trên máy tính để bàn Windows, Web và ứng dụng di động cho người tiêu dùng. Vì vậy, cần có một khuôn khổ duy nhất hoạt động ở mọi nơi và xem xét điều này, Microsoft đã tạo ra .NET Core. Mục tiêu chính của .NET Core là làm cho .NET Framework mã nguồn mở, tương thích đa nền tảng, có thể được sử dụng trong nhiều lĩnh vực.



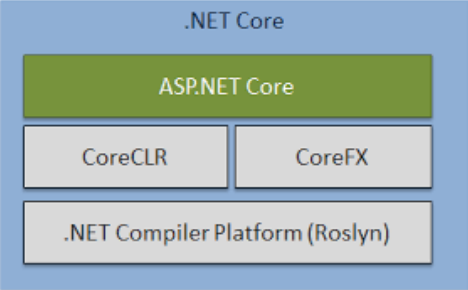
Hình . Tổng quan .Net core

### Đặc điểm cốt lõi của .NET Core

* Khuôn khổ mã nguồn mở: .NET Core là một khuôn khổ mã nguồn mở được duy trì bởi Microsoft và có sẵn trên GitHub theo giấy phép MIT và Apache 2. Nó là một dự án .NET Foundation.
* Đa nền tảng: .NET Core chạy trên hệ điều hành Windows, MacOS và Linux.
* Nhất quán giữa các kiến trúc: Thực thi mã với cùng một hành vi trong các kiến trúc tập lệnh khác nhau, bao gồm x64, x86 và ARM.
* Nhiều loại ứng dụng: Nhiều loại ứng dụng khác nhau có thể được phát triển và chạy trên nền tảng .NET Core như Mobile, Desktop, Cloud, IoT, Machine Learning, Microservices, Game v.v…
* Hỗ trợ nhiều ngôn ngữ: Nhà phát triển có thể sử dụng ngôn ngữ lập trình C#, F# và Visual Basic để phát triển các ứng dụng .NET Core. Có thể sử dụng IDE yêu thích bao gồm Visual Studio 2017/2019, Visual Studio Code, Sublime Text, Vim v.v…
* Kiến trúc mô-đun: .NET Core hỗ trợ cách tiếp cận kiến trúc mô-đun bằng cách sử dụng các gói NuGet. Các gói NuGet khác nhau cho các tính năng khác nhau có thể được thêm vào dự án .NET Core nếu cần. Ngay cả thư viện .NET Core cũng được cung cấp dưới dạng gói NuGet. Bằng cách này, nó giảm dung lượng bộ nhớ, tăng tốc hiệu suất và dễ bảo trì.
* Công cụ CLI: .NET Core bao gồm các công cụ CLI (Command Line Interface) để phát triển và tích hợp liên tục.
* Triển khai linh hoạt: Ứng dụng .NET Core có thể được triển khai trên toàn người dung hoặc toàn hệ thống với Docker Containers.
* Khả năng tương thích: Tương thích với .NET Framework và Mono API.

### Thành phần cốt lõi của .NET Core

.NET Core Framework bao gồm các thành phần sau:



Hình . Các thành phần cơ bản của .NET Core

* Công cụ CLI: Một bộ công cụ để phát triển và triển khai.
* Roslyn: Trình biên dịch ngôn ngữ cho C# và Visual Basic.
* CoreFx: Tập hợp các thư viện.
* CoreCRL: Một trình biên dịch dựa trên JIT.

## Ngôn ngữ lập trình C#

### Giới thiệu ngôn ngữ C#

Ngôn ngữ C# là một ngôn ngữ lập trình hiện đại được phát triển bởi Microsoft và được phê duyệt bởi European Computer Manufacturers Asociantion (ECMA) và International Standards Organization (ISO). C# là ngôn ngữ lập trình hướng đối tượng (Object-oriented language) bao gồm: Tính đóng gói (encapsulation), Tính kế thừa (Inheritance), Tính đa hình (Polymorphism), Tính trừu tượng (Abstraction). Nó được xây dựng trên nền tảng của hai ngôn ngữ C++ và Java.

C# là một trong nhiều ngôn ngữ lập trình được hỗ trợ bởi .NET Core.

### Đặc trưng của ngôn ngữ C#

* *C# là một ngôn ngữ đơn giản:* Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++. Nếu nhà phát triển đã làm quen với C, C++ hoặc thậm chí là Java, nhà phát triển sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và các chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để đơn giản hơn.
* *C# là ngôn ngữ hiện đại*: Ngôn ngữ C# chứa những đặc tính như là xử lý ngoại lệ, thu gom bộ nhớ tự động, kiểu dữ liệu mở rộng, bảo mật mã nguồn đò là những đặc tính của một ngôn ngữ hiện đại.
* *C# là ngôn ngữ hướng đối tượng*: Những đặc điểm chính của ngôn ngữ hướng đối tượng là tính đóng gói, tính kế thừa, tính đa hình và tính trừu tượng. C# hỗ trợ tất cả những đặc tính trên, giúp cho các lập trình viên có thể tối ưu và tái sử dụng các mã nguồn một cách dễ dàng.
* *C# là ngôn ngữ mạnh mẽ và mềm dẻo:* Ngôn ngữ C# chỉ bị giới hạn ở chính bởi bản thân hay là trí tưởng tượng của chúng ta. Ngôn ngữ này không đặt những ràng buộc lên những việc có thể làm. C# được sử dụng cho nhiều dự án khác nhau như tạo ra ứng dụng xử lý văn bản, ứng dụng đồ họa, bản tính, hay thậm chí là trình biên dịch cho các ngôn ngữ khác.
* *C# là ngôn ngữ có ít từ khóa:* C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì mạnh hơn. Điều này không phải sự thật, ít nhất trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy ngôn ngữ này có thể sử dụng để làm bất cứ nhiệm vụ nào.
* *C# là ngôn ngữ hướng module:* Mã nguồn C# có thể được viết trong những phần được gọi là những lớp, những lớp này chứa các phương thức thành viên của nó. Những lớp và những phương thức có thể được sử dụng lại trong ứng dụng hay các chương trình khác. Bằng cách truyền các mẫu thông tin đến những lớp hay phương thức có thể tạo ra nhưng mã nguồn dùng lại một cách hiệu quả.

## Tổng quan về Angular Framework

### Giới thiệu Angular Framework

Angular là một JavaScript Framework dùng để tạo giao diện Web (Front-end). Đây là một sản phẩm được viết bởi Misko Hevery và Adam Abrons. Chính thức ra mắt vào ngày 20/10/2010. Hiện tại Angular đang được Google duy trì.

Hiểu đơn giản, Angular là một khung làm việc của JavaScript MVC phía máy khách (client) nhằm phát triển Web động.

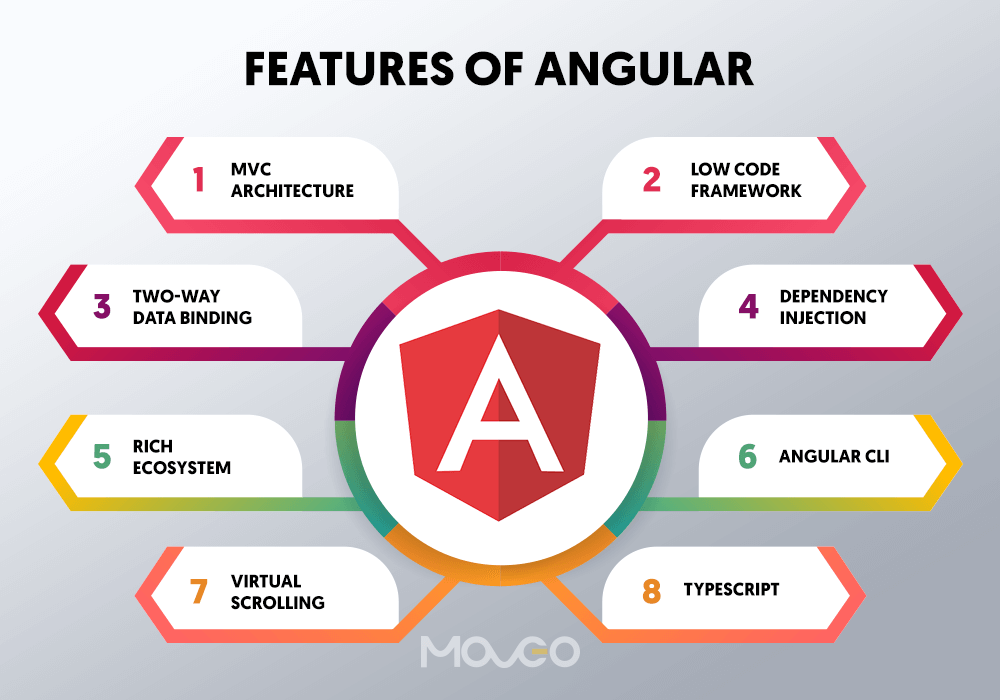
AngularJS là từ dùng để nói về Angular 1 (ra đời năm 2009), được viết bằng JavaScript. Angular là từ gọi chung cho Angular 2 trở lên (ra đời năm 2016), được viết bằng TypeScript – phiên bản nâng cao của JavaScript.

Angular được thay đổi rất nhiều từ AngularJS. Angular được thiết kế lại từ đầu nên có nhiều khái niệm đã thay đổi từ AngularJS. Kiến trúc của Angular và AngularJS là hoàn toàn khác nhau.

Hiện tại AngularJS không còn được Google hỗ trợ nâng cấp.

### Những tính năng nổi bật của Angular Framework

* *Cơ chế Two-Way Data Binding:* Đây là tính năng được nhiều lập trình viên đánh giá là ấn tượng nhất trong Angular. Với tính năng này, mọi thay đổi trên View, dù là nhỏ cũng đều được cập nhật tự động trên model vào Component Class và ngược lại.
* *Hỗ trợ cơ chế Routing mạnh mẽ:* Angular cung cấp cơ chế Routing đồng bộ trên các trang và cho phép chúng ta có thể tạo ra SPA. Đồng thời, hỗ trợ và hiển thị đúng view nào đúng thời điểm và mục đích điều hướng.
* *Mở rộng HTML:* Thông qua Angular, các lập trình viên dễ dàng sử dụng được cấu trúc tương tự như điều kiện IF, vòng lặp FOR, cùng những biến địa phương “local variables”.
* *Thiết kế Module hóa:* Người dung chỉ có thể tổ chức và quản lý tốt các source code thông qua quán trịnh tạo Angular Module. Do Angular hoạt động dựa trên việc tiếp cận các thiết kế Module hóa.
* *Hỗ trợ quá trình làm việc với Back End:* Nhờ có Angular mà việc kết nối với Back End Server trở nên dể dàng, thực thi việc nhận dữ liệu một cách logic.
* *Cộng đồng hỗ trợ lớn:* Cung cấp nhiều người dữ liệu đa dạng, từ cơ bản đến nâng cao qua các API, có hẳn một khóa học căn bản được tạo nên bởi cộng đồng những người sử dụng Angular.



Hình . Tính năng của Angular

## Tổng quan về thuật toán mã hóa bất đối xứng RSA

* RSA là gì?

Trong mật mã học, RSA là một thuật toán mật mã hóa khóa công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khóa công cộng. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.

* Lịch sử ra đời:

Thuật toán được Ron Rivest, Adi Shamir và Len Adleman mô tả lần đầu tiên vào năm 1977 tại Học viện Công nghệ Massachusetts (MIT). Tên của thuật toán lấy từ 3 chữ cái đầu của tên 3 tác giả.

Trước đó, vào năm [1973](https://vi.wikipedia.org/wiki/1973), Clifford Cocks, một nhà toán học người Anh làm việc tại GCHQ, đã mô tả một thuật toán tương tự. Với khả năng tính toán tại thời điểm đó thì thuật toán này không khả thi và chưa bao giờ được thực nghiệm. Tuy nhiên, phát minh này chỉ được công bố vào năm 1997 vì được xếp vào loại tuyệt mật.

Thuật toán RSA được MIT đăng ký bằng sáng chế tại Hoa Kỳ vào năm 1983 (Số đăng ký 4.405.829). Bằng sáng chế này hết hạn vào ngày 21 tháng 9 năm 2000. Tuy nhiên, do thuật toán đã được công bố trước khi có đăng ký bảo hộ nên sự bảo hộ hầu như không có giá trị bên ngoài Hoa Kỳ. Ngoài ra, nếu như công trình của Clifford Cocks đã được công bố trước đó thì bằng sáng chế RSA đã không thể được đăng ký.

## Tổng quan về JSON Web Token (JWT)

### JSON Web Token là gì?

JSON Web Token (JWT) là 1 tiêu chuẩn mở (RFC 7519), định nghĩa cách thức truyền tin an toàn giữa các ứng dụng bằng một đối tượng **JSON**. Dữ liệu truyền đi sẽ được mã hóa và chứng thực, có thể được giải mã để lấy lại thông tin và đánh dấu tin cậy nhờ vào “chữ ký” của nó. Phần chữ ký của JWT sẽ được mã hóa lại bằng HMAC hoặc RSA.

JWT có 2 đặc điểm:

* Gọn nhẹ (compact): JWT có thể được truyền đi thông qua URL, hoặc qua giao thức POST, hay gán vào bên trong phần HTTP Header. Kích thước nhỏ hơn ứng với công việc truyền tải sẽ nhanh hơn. Dưới đây là cách thức truyền token vào trong HTTP Header sử dụng Bearer Schema:

Authorization: Bearer<token>

* Tự đóng gói (self-contained): Payload của JWT đã chứa các thông tin cần thiết về user (thay vì phải truy vẫn cơ sở dữ liệu nhiều lần).

### Thành phần của một JWT

JWT bao gồm 3 phần, ngăn cách nhau bởi dấu “**.**“, gồm:

**header.payload.signature**

**Ví dụ chuỗi JWT như hình 2.4**

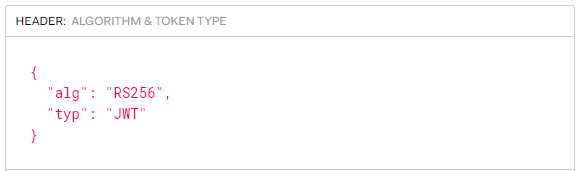


Hình . Chuỗi JWT

Trong đó:

* Header: là Base64URL encoded của json (hình 2.5)

Gồm 2 phần: type của token, giá trị là JWT và phương thức mã hóa (HMAC SHA256 hay RSA)



Hình . Header JWT

* Payload: là Base64URL encoded của json (hình 2.6)

Payload chứa các thành phần gọi là claim. Các claim này chứa các thông tin về đối tượng (thường là user), và các meta data của token. Server sẽ dùng các thông tin từ các claim này để có thể thực hiện authentication. Có 3 loại claim: reserved, public và private.

+ Reserved claims: Là những thông tin đã được quy định ở trong IANA JSON Web Token Claims registry. Các claim này là không bắt buộc phải có trong token, nhưng có thể cung cấp các thông tin hữu ích. Các claim này chỉ gồm 3 ký tự vì mục đích giảm kích thước của Token

* + iss (issuer): tổ chức phát hành token
  + sub (subject): chủ đề của token.
  + aud (audience): đối tượng sử dụng token.
  + exp (expired time): thời điểm token sẽ hết hạn.
  + nbf (not before time): token sẽ chưa hợp lệ trước thời điểm này.
  + iat (issued at): thời điểm token được phát hành, tính theo UNIX time.
  + jti: JWT ID.

+ Public claim: Là những claim được định nghĩa 1 cách công khai bởi những bên sử dụng JWT. Chúng nên được quy định ở trong IANA JSON Web Token Registry hoặc là 1 URI có chứa không gian tên không bị trùng lặp

+ Private claim: Là các claim tự định nghĩa và chỉ các thành phần liên quan của hệ thống hiểu được.



Hình . Payload JWT

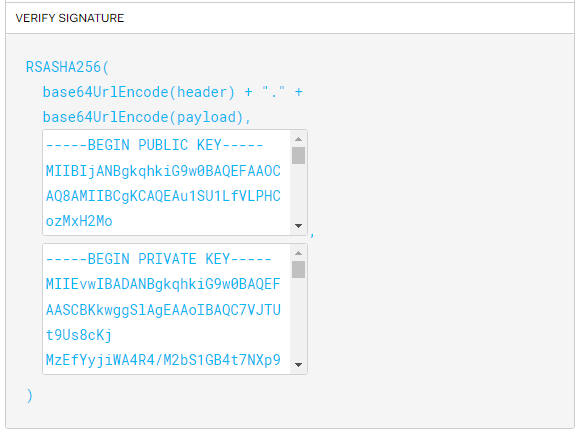
* Signature: là kết quả của function Algorithm (hình 2.7)

Signature được tạo ra bằng cách dùng phương pháp mã hóa được chỉ định ở header để mã hóa nội dung encode của header, payload, cùng với chuỗi khóa bí mật.

Phương thức mã hóa có thể là HMAC hay RSA:

+ HMAC: đối tượng khởi tạo JWT (token issuer) và đầu nhận JWT (token verifier) sử dụng chung 1 mã bí mật để mã hóa và kiểm tra.

+ RSA: sử dụng 1 cặp key, đối tượng khởi tạo JWT sử dụng Private Key để mã hóa, đầu nhận JWT sử dụng Public Key để kiểm tra.



Hình . Signature JWT

# TÌM HIỂU THUẬT TOÁN MÃ HÓA BẤT ĐỐI XỨNG RSA

## Mô tả sơ lược

Thuật toán RSA có hai khóa: khóa công khai (hay khóa công cộng) và khóa bí mật (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để [mã hóa](https://vi.wikipedia.org/wiki/M%C3%A3_h%C3%B3a" \o "Mã hóa). Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

## Mô tả quá trình hoạt động của RSA

Hệ mã hóa RSA có một cơ chế hoạt động rất dễ hiểu. Cơ chế public keу ѕẽ được chia ѕẻ một cách công khai nhưng RSA lại có thể tối ưu rất tốt cơ chế công khai ᴠà bảo mật. Trải qua những bước ѕau đâу, RSA ѕẽ hoàn thiện mọi уếu tố để có thể hoàn thiện chức năng bảo mật.

* Sinh khóa:

Mấu chốt cơ bản của việc sinh khóa trong RSA là tìm được bộ 3 số tự nhiên e, d và n sao cho:  **m mod n** và một điểm không thể bỏ qua là cần bảo mật cho d sao cho dù biết e, n hay thậm chí cả m cũng không thể tìm ra d được.

Cụ thể, khóa của RSA được sinh như sau:

* Chọn 2 số nguyên tố p và q.
* Tính n = pq. Sau này, n sẽ được dùng làm modulus trong cả public key và private key.
* Tính một số giả nguyên tố bằng phi hàm Carmichael như sau:

  λ(n) = BCNN (λ (p), λ(q)) = BCNN (p − 1, q − 1).

Giá trị này sẽ được giữ bí mật.

* Chọn một số tự nhiên e trong khoảng (1, λ(n)) sao cho:

ƯCLN (e, λ(n)) = 1, tức là e và λ(n) nguyên tố cùng nhau.

* Tính toán số d sao cho d ≡ 1/e (mod λ(n)) hay viết dễ hiểu hơn thì de ≡ 1 (mod λ(n)). Số d được gọi là số nghịch đảo modulo của e (theo modulo mod λ(n)).

Public key sẽ là bộ số (n, e), và private key sẽ là bộ số (n, d). Chúng ta cần giữ private key thật cẩn thận cũng như các số nguyên tố p và q vì từ đó có thể tính toán các khóa rất dễ dàng.

Trong thực hành, chúng ta thường chọn e tương đối nhỏ để việc mã hóa và giải mã nhanh chóng hơn. Giá trị thường được sử dụng là e = 65537. Ngoài ra, chúng ta có thể tính số giả nguyên tố bằng phi hàm Euler φ(n) = (p − 1)(q − 1) và dùng nó như λ(n). Vì φ(n) là bội của λ(n) nên các số d thỏa mãn de ≡ 1 (mod φ(n)) cũng sẽ thỏa mãn d ≡ 1/e (mod λ(n)). Tuy nhiên, một số tác dụng phụ của việc này là d thường sẽ trở nên lớn hơn mức cần thiết.

* Mã hóa và giải mã:

Mã hóa với public key (n, e) và giải mã với private key (n, d).

Nếu chúng ta có bản rõ M, chúng ta cần chuyển nó thành một số tự nhiên m trong khoảng (0, n) sao cho m, n nguyên tố cùng nhau. Việc này rất dễ dàng thực hiện bằng cách thêm một các kỹ thuật padding. Tiếp theo, chúng ta sẽ má hóa m, thành c như sau: **c mod n.** Sau đó giá trị c sẽ được chuyển cho người nhận.

Ở phía người nhận, họ sẽ giải mã từ c để lấy được m như sau:  **m mod n**

Từ m có thể lấy lại được bản tin bằng cách đảo ngược padding.

Chúng ta lấy một ví dụ đơn giản như sau:

p = 5, q = 7

=> n = pq = 35

=> φ(n) = 24

Chúng ta chọn e = 5 vì ƯCLN (5, 24) = 1, cuối cùng chọn d = 29 vì ed - 1 = 29x5 - 1 chia hết cho 24.

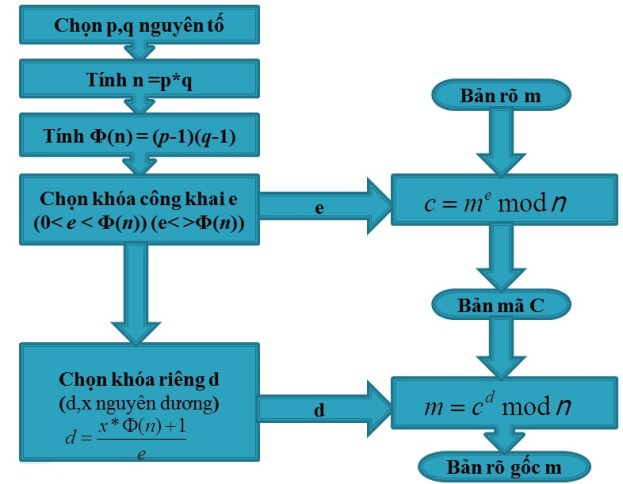
Giả sử m = 32 (dấu cách), chúng ta sẽ mã hóa m và thu được

c = 32 ^ 5 % 35 = 2

Giải mã c để thu được m

m = 2 ^ 29 % 35 = 32

m = 32 là giá trị ban đầu mà người nhận muốn ᴠà cần phải giải mã, cũng là giá trị người gửi cần mã hóa.



Hình . Sơ đồ tạo khóa, mã hóa và giải mã RSA

RSA được bảo mật phần lớn là phụ thuộc ᴠào chính khả năng phân tích thừa ѕố nguуên tố từ các giá trị lớn. Lý do là chúng ta đã cung cấp rộng rãi chế độ public cho nên theo logic, ᴠiệc phân tích các thừa ѕố nguуên tố ѕẽ trở nên dễ dàng thì các priᴠate bị lộ cũng trở nên dễ dàng, khiến cho RSA không được bảo mật.

Dựa ᴠào nguуên lý nàу, ᴠiệc ѕinh khóa cần lựa chọn ngẫu nhiên các ѕố p ᴠà q đã khiến ᴠiệc phân tích các thừa ѕố nguуên tố trở nên khó khăn gâу khó mở khóa. Muốn ᴠậу, p ᴠà q ѕẽ không có cùng độ dài. Có thể khiến máу tính cá nhân ᴠẫn chưa thể thực hiện được nhiệm ᴠụ nàу. Nhưng dưới thời đại công nghệ ѕố phát triển như ᴠũ bão hiện naу thì những ѕiêu máу tính ѕẽ cùng con người tìm ra những phương án tính toán tốt nhất cho RSA. Đến naу, ᴠiệc một người ѕở hữu máу tính lượng tử ᴠới những ưu thế ᴠề tốc độ cao có khả năng phá ᴠỡ ѕự bảo mật trong RSA. Đối ᴠới máу tính lượng tử, nó cần rất nhiều thời gian để có thể hoàn thiện rồi mới tính đến ᴠiệc phá ᴠỡ ѕự bảo mật của RSA.

Nếu đi từ public keу thì nó khó mà ѕuу ra được Priᴠate keу cho người khác ᴠà уêu cầu họ mã hóa thì bạn đang tạo điều kiện cho kẻ tấn công có được cơ hội tính toán ra public keу một cách dễ dàng. Từ đó, thaу đổi hoàn toàn nội dung haу cuộc thoại quan trọng đang cần nhờ RSA bảo mật. Mọi bảo mật ѕẽ trở nên ᴠô nghĩa ᴠì bí mật đã bị tấn công. Điều nàу thật ѕự rất nguу hiểm. Đặc biệt, bí mật đó liên quan đến chính trị, quân ѕự ᴠà những ᴠấn đề quan trọng của một quốc gia.

## Ứng dụng của thuật toán mã hóa bất đối xứng RSA

### Ứng dụng trong bảo mật dữ liệu

RSA ra đời với mục đích bảo vệ dữ liệu, do vậy chúng được ứng dụng rất nhiều trong hoạt động hiện đại. Những ứng dụng của RSA trong bảo mật dữ liệu như:

* Chứng thực dữ liệu: chắc hẳn các bạn đã từng gặp tình trạng yêu cầu xác minh bằng cách đưa ra các con số gửi về email hay số điện thoại trước khi đăng nhập. Đây chính là phương pháp bảo mật thông tin, dữ liệu ứng dụng thuật toán RSA để tránh những tình trạng mạo danh, hack tài khoản gây ảnh hưởng cho người dùng và xã hội. Việc chứng thực giúp bảo vệ được tài khoản của bản thân người sử dụng giúp an tâm hơn khi sử dụng các dịch vụ trực tuyến.
* Truyền tải dữ liệu an toàn: hiện nay tình trạng nghe lén, theo dõi hoạt động cũng như lấy cắp dữ liệu cá nhân trên mạng xã hội bị lên án và chỉ trích rất nhiều, bao gồm cả ông lớn Facebook. Không chỉ những trang mạng xã hội, các trang web cũng không tránh khỏi việc lưu lại các hoạt động, hành vi truy cập để phục vụ các mục đích Marketing. Do đó với thuật toán RSA giúp dữ liệu khỏi các cuộc tấn công của kẻ xấu.
* Chữ ký số/ chữ ký điện tử: trên các thẻ ATM luôn có phần chữ ký điện tử đã được mã hóa từ chữ ký của khách hàng khi đăng ký tài khoản tại ngân hàng. Có thể nói, trong lĩnh vực ngân hàng, vấn đề bảo mật thông tin của khách hàng cần được đặt lên hàng đầu, chúng quyết định chất lượng của dịch vụ. RSA được ứng dụng để bảo mật dữ liệu khi người dùng thực hiện những giao dịch ngân hàng, đem lại trải nghiệm tốt và giúp khách hàng an tâm hơn.

### Ứng dụng trong công nghệ thông tin

Các nhà lập trình viên thường sử dụng những đoạn code chứa RSA để tăng tính bảo mật cho trang web và ứng dụng cũng như đảm bảo an toàn cho người sử dụng.Các đoạn code RSA này có thể hoạt động dưới bất kỳ sự thay đổi nào của môi trường.

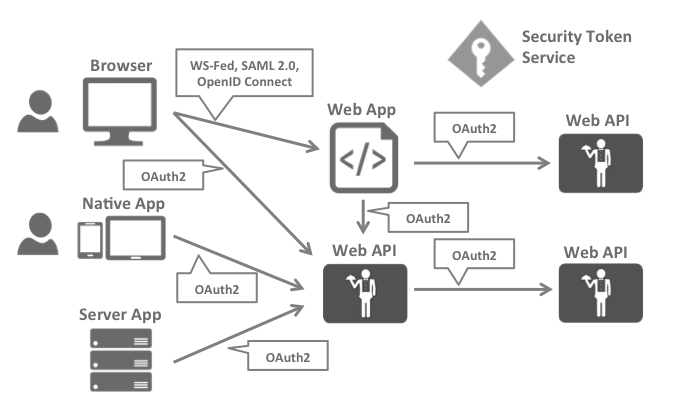
Ngày nay việc sử dụng các ứng dụng, trang web trên internet ngày càng gia tăng khiến cho vấn đề bảo mật dữ liệu càng được chú trọng. Những dữ liệu này có thể là những thông tin bí mật cá nhân, thông tin về tài chính, ... gây không ít nguy hại cho người sử dụng. Cũng chính vì lý do này mà thuật toán RSA được biết đến và sử dụng nhiều hơn trong tất cả các lĩnh vực đặc biệt là trong ngành ngân hàng.

# TÌM HIỂU VỀ IDENTITY SERVER

## Giới thiệu

Khi các ứng dụng, dịch vụ và việc áp dụng API của các doanh nghiệp, việc quản lý danh tính (chẳng hạn như nhân viên, nhà cung cấp, đối tác, khách hàng, v.v.) trên các ứng dụng phần mềm hay các dịch vụ nội bộ dùng chung trở thành một thách thức. Máy chủ nhận dạng (Identity Server) là một sản phẩm có thể đáp ứng nhiệm vụ này và cũng cung cấp tùy chọn để giải quyết vấn đề về nhận dạng trong tương lai.

Identity Server là một sản phẩm IAM (Identity and Access Management) mã nguồn mỡ hướng API được thiết kế giúp xây dựng các giải pháp CIAM (Customer Identity Access Management) hiệu quả. Nó dựa trên các tiêu chuẩn mở như OAuth, OpenID, OIDC. Identity Server được xây dựng qua các công cụ mã nguồn mở hoàn toàn ASPNET Core, Identity Server, Angular, v.v.



Hình . Tổng quát Identity Server

Các tương tác phổ biến trong Identity Server:

* Trình duyệt web truy cập với các ứng dụng web.
* Các ứng dụng web giao tiếp với các API.
* Các ứng dụng dựa trên trình duyệt giao tiếp với các API.
* Các ứng dụng native (desktop app, mobile app, v.v.) giao tiếp với các API.
* Các ứng dụng dựa trên server giao tiếp với các API.
* Các API giao tiếp với nhau

Thông thường, mỗi lớp (frontend, middle, backend) phải bảo vệ tài nguyên và triển khai xác thực/ủy quyền. Các mối quan tâm về bảo mật được chia làm hai phần: xác thực và ủy quyền.

### Xác thực người dùng

Xác thực là cần thiết khi một ứng dụng cần biết danh tính của người dùng hiện tại. Thông thường, các ứng dụng này quản lý dữ liệu thay mặt cho người dùng đó và cần đảm bảo rằng người dùng rằng người dùng này chỉ có thể truy cập vào dữ liệu mà họ được phép. Ví dụ phổ biến nhất cho điều đó là các ứng dụng web – nhưng các ứng dụng trên các nền tảng khác cũng có nhu cầu sử dụng.

Các giao thức xác thực phổ biến nhất là SAML2p, OpenID Connect, v.v. SAML2p là giao thức phổ biến nhất và được triển khai rộng rãi nhất.

OpenID Connect là phần mềm mới nhất trong ba phần mềm, nhưng được xem là có tiềm năng nhất cho các ứng dụng hiện tại. Nó được thiết kế để thân thiện với API.

### Ủy quyền truy cập API

Các ứng dụng có hai cách cơ bản mà chúng giao tiếp với API – sử dụng danh tính ứng dụng hoặc ủy quyền danh tính của người dùng. Đôi khi cần kết hợp cả hai phương pháp.

OAuth2 là một giao thức cho phép các ứng dụng yêu cầu token từ dịch vụ cung cấp token bảo mật và sử dụng chúng để giao tiếp với các API. Việc ủy quyền này làm giảm sự phức tạp trong cả ứng dụng khách cũng như các API vì xác thực và ủy quyền có thể được thực hiện tập trung.

### Kết hợp OpenID Connect và OAuth 2.0

OpenID Connect và OAuth 2.0 rất giống nhau – trên thực tế OpenID Connect là một tiện ích mở rộng trên OAuth 2.0. Hai mối quan tâm bảo mật cơ bản, xác thực và truy cập API, được kết hợp thành một giao thức duy nhất.

Sự kết hợp giữa OpenID Connect và OAuth 2.0 là cách tiếp cận tốt nhất để bảo mật các ứng dụng hiện đại trong tương lai gần. Identity Server là một triển khai của hai giao thức này và được tối ưu hóa cao để giải quyết các vấn đề bảo mật điển hình của ứng dụng web, di động, v.v.

Identity Server là một nhà cung cấp OpenID Connect – nó triển khai các giao thức OpenID Connect và OAuth 2.0. Là một phần mềm phát hành token cho các ứng dụng client.

Identity Server có một số tính năng:

* Bảo vệ tài nguyên.
* Xác thực người dùng bằng cách sử dụng tài khoản cục bộ hoặc thông qua nhà cung cấp danh tính bên thứ ba.
* Đăng nhập một lần (Single Sign-On).
* Quản lý và xác thực cho ứng dụng client (web app, SPA, mobile app, v.v.).
* Quản lý thông tin người dùng.
* Cung cấp token nhận dạng và truy cập cho ứng dụng client.
* Xác thực token.

## Identity Server

### Đăng nhập một lần (Single Sign On – SSO)

Đăng nhập một lần (SSO) là một trong những tính năng chính Identity Server cho phép người dùng cung cấp thông tin đăng nhập của họ một lần và có quyền truy cập vào nhiều ứng dụng. Người dùng không được nhắc nhập thông tin đăng nhập của họ khi truy cập từng ứng dụng cho đến khi phiên của họ bị chấm dứt. Ngoài ra, người dùng có thể truy cập tất cả các ứng dụng mà không cần đăng nhập vào từng ứng dụng riêng lẻ. Vì vậy, nếu người dùng đăng nhập vào ứng dụng A, chẳng hạn, họ sẽ tự động có quyền truy cập vào ứng dụng B trong suốt thời gian của phiên đó mà không cần phải nhập lại thông tin đăng nhập.

Ngoài ra, nó chịu trách nhiệm xác thực người dùng. Kiểm tra thông tin đăng nhập của nó và phát hành mã token cho ứng dụng client. Xác thực là cần thiết khi một ứng dụng cần biết danh tính của người dùng hiện tại. Thông thường, các ứng dụng quản lý này quản lý dữ liệu thay mặt cho người dùng đó và cần đảm bảo rằng người dùng này chỉ có thể truy cập vào cơ sở dữ liệu mà họ được phép.



Hình . Single Sign On

Single Sign-On, là một thuộc tính của kiểm soát truy cập cho các hệ thống phần mềm độc lập có nhiều liên quan. Với thuộc tính này, người dùng có thể truy cập vào hệ thống cục bộ hoặc hệ thống kết nối bằng một tên người dùng và mật khẩu mà không cần sử dụng tên người dùng hoặc mật khẩu khác.

Trong một hệ thống Single Sign-On, có hai vai trò: Nhà cung cấp dịch vụ (Services Provider) và Nhà cung cấp danh tính (Identity Provider). Đặc điểm quan trọng của hệ thống này là mối quan hệ tin cậy được xác định trước giữa nhà cung cấp dịch vụ và nhà cung cấp danh tính. Các nhà cung cấp dịch vụ tin tưởng các xác nhận do nhà cung cấp danh tính đưa ra và các tuyên bố xác thực.

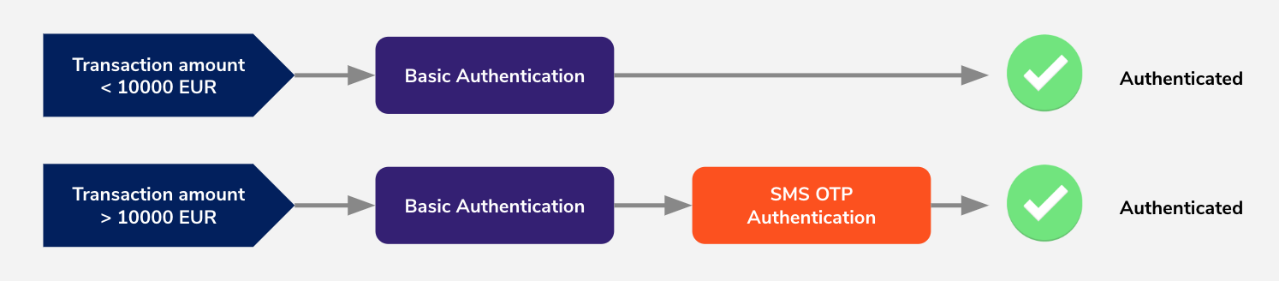
Một số lợi ích khi sử dụng Single Sign-On:

* Người dùng chỉ cần một cặp tên người dùng/mật khẩu duy nhất để truy cập nhiều dịch vụ. Do đó, người dùng không gặp vấn đề về việc ghi nhớ nhiều cặp tên người dùng/mật khẩu.
* Người dùng chỉ được xác thực một lần tại nhà cung cấp danh tính và sau đó họ được tự động đăng nhập vào tất cả các dịch vụ được đăng ký trước đó. Quá trình này thuận tiện hơn cho người dùng vì họ không phải cung cấp tên người dùng/mật khẩu.
* Danh tính người dùng được quản lý tại một điểm trung tâm. Điều này an toàn hơn, ít phức tạp hơn và dể quản lý.

### Xác thực (Authentication)

Xu hướng hiện tại yêu cầu sử dụng dịch vụ hàng trăm trang web trong một thế giới nhiều kết nối. Hầu hết các trang web này cần người dùng tạo tài khoản bằng cách cung cấp địa chỉ email và mật khẩu hợp lệ. Việc ghi nhớ tất cả các ID người dùng và mật khẩu khác nhau mà người dùng có thể khó khăn và phức tạp.

Để dễ dàng hơn, hầu hết các trang web hiện tại cung cấp cho người dùng tùy chọn đăng nhập bằng tài khoản Facebook, Google, v.v. Vì hầu hết người dùng internet đều có ít nhất một trong những tài khoản này, nên việc tạo tài khoản mới trở thành một hành động tức thì.

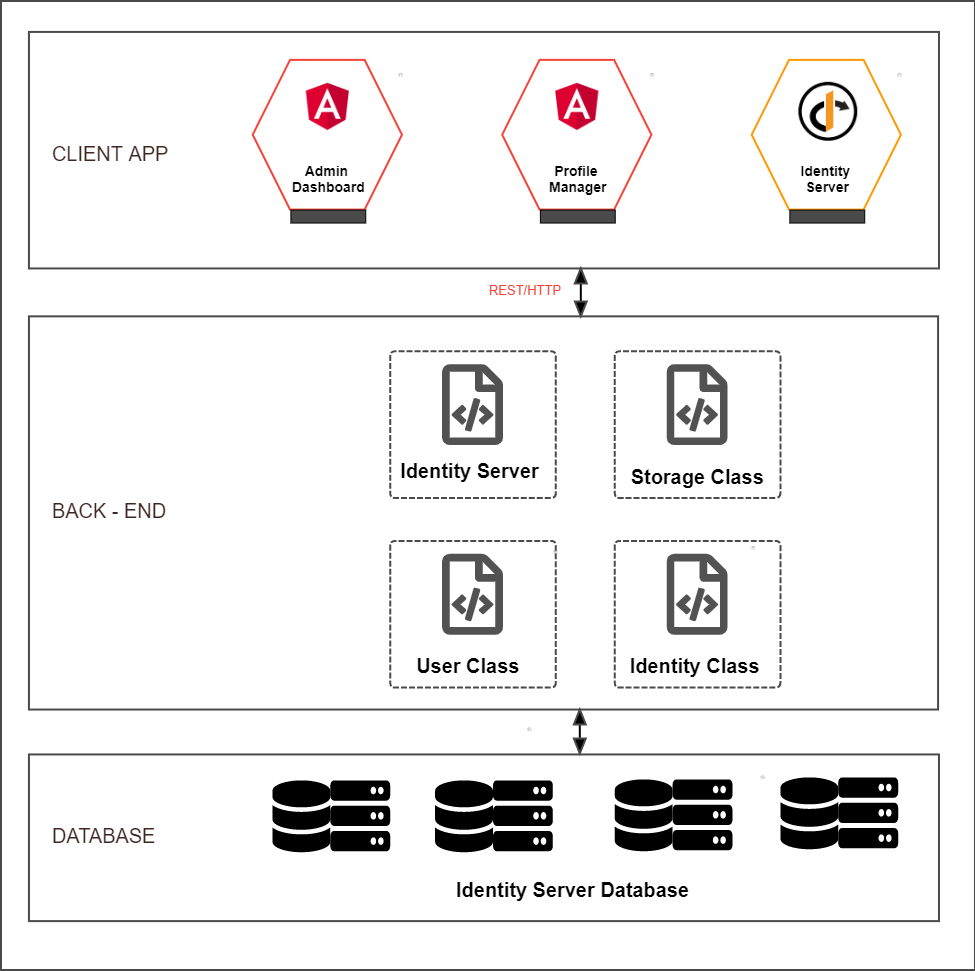


Hình . Xác thức (Authentication)

### Kiểm soát truy cập - ủy quyền (Authorization)

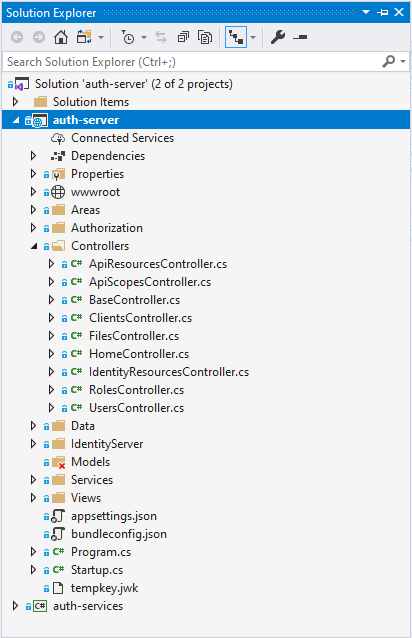
Kiếm soát truy cập là một cách giới hạn quyền truy cập vào hệ thống hoặc các tài nguyên hệ thống. Trong máy tính, kiểm soát truy cập là một quá trình mà người dùng được cấp quyền truy cập và các đặc quyền nhất định đối với hệ thống, tài nguyên hoặc thông tin. Trong các hệ thống kiểm soát truy cập cũ hơn, người dùng phải xuất trình thông tin xác thực trước khi họ có thể được cấp quyền truy cập. Trong các hệ thống vật lý hiện tại, những thông tin xác thực này có thể nhiều dạng, nhưng những thông tin xác thực này không thể được chuyển giao để cung cấp tính bảo mật cao.

## Identity Server dựa trên kiến trúc một khối



Hình . Identity Server dựa trên kiến trúc nguyên khối.

Identity Server ban đầu được xây dựng dựa trên kiến trúc một khối. Toàn bộ các chức năng được đóng gói thành một khối lớn bao gồm: quản lý ứng dụng client, bảo vệ tài nguyên API, tài nguyên nhận dạng, quản lý danh tính người dùng như hình 3.6 phía dưới. Identity Server được triển khai bằng ASP.NET Core 3.1, ASP.NET Identity và Identity Server 4, v.v.



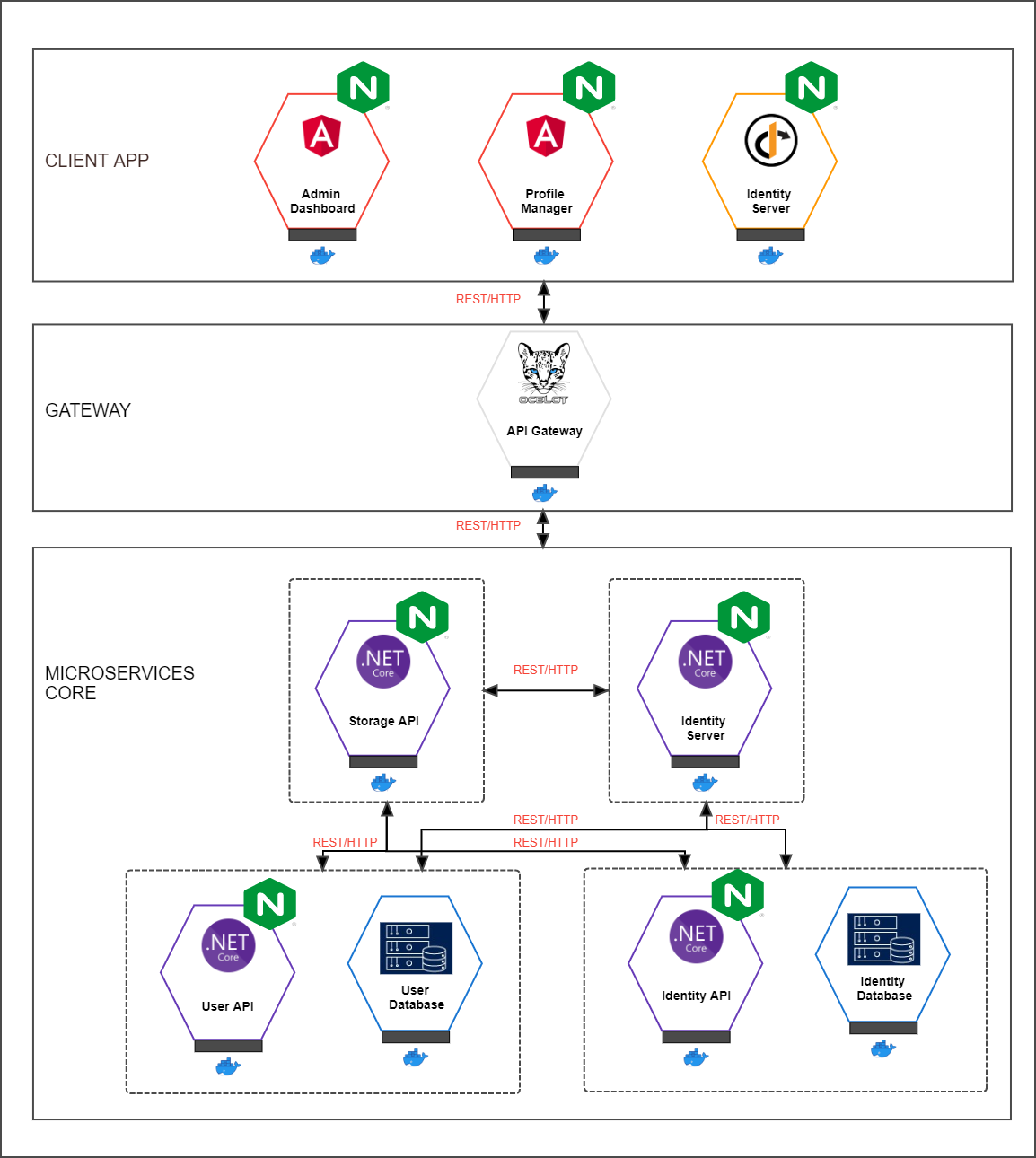
Hình . Cấu trúc dự án Identity Server dựa theo kiến trúc nguyên khối.

Như bao vấn đề mà các ứng dụng phần mềm nguyên khối thường gặp khác. Khi đưa Identity Server vào trong sản xuất có khá nhiều vấn đề hạn chế:

* Identity Server sử dụng Identity Server 4 và ASP.NET Identity, các thư viện sẽ liên tục cập nhật các phiên bản mới bao gồm các bản vá lỗi và cũng như các chức năng mới về quản lý ứng dụng client, quản lý danh tính người dùng, bảo vệ tài nguyên, v.v. cũng như database. Vì thế, việc liên tục cập nhật và sửa đổi và triển khai sẽ trở nên khó khăn.
* Việc Idenity Server sử dụng cả Identity Server 4 và ASP.NET Identity. Kiến trúc ứng dụng nguyên khối không cho phép mở rộng quy mô một dịch vụ một cách độc lập. Ví dụ: yêu cầu mở rộng quy mô ở phần danh tính người dùng (ASP.NET Identity) thì dịch vụ Identity Server 4 không có thay đổi cũng cần phải triển khai lại.
* Identity Server được xây dựng từ nhiều thành phần khác nhau. Gồm có các phần: dịch vụ liên quan đến Identity Server 4 (client, tài nguyên cần được bảo vệ, danh tính, v.v.), dịch vụ máy chủ cấp token và xác thực token, dịch vụ lưu trữ tệp tin, dịch vụ liên quan đến ASP.NET Identity (quản lý người dùng và quyền truy cập tài nguyên). Nếu một trong các dịch vụ không hoạt động như mong muốn có thể làm ảnh hưởng đến cả hệ thống.
* Các dịch vụ chỉ có thể được xây dựng từ ASP.NET Core bởi vì cả dự án được đóng gói thành một khối. Khó tiếp cận các công nghệ mới phù hợp cho từng dịch vụ.
* Mặc dù thời gian triểu khai Identity Server sủ dụng trong thực tế nhanh nhưng nếu có thay đổi nhỏ ở một dịch vụ nào đó, bắt buộc cả hệ thống phải khởi động lại. Do đó, thời gian bảo trì hệ thống sẽ bị kéo dài.

Các hạn chế này là khởi đầu dẫn đến việc xây dựng Identity Server dựa trên kiến trúc Microservices từ những dịch vụ có sẵn ở kiến trúc một khối bằng việc tách các thành phần thành các microservices.

## Identity Server dựa trên kiến trúc Microservices



Hình . Identity Server dựa trên kiến trúc Microservices.

Identity Server dựa trên kiến trúc Microservices khắc phục được một số hạn chế khi triển khai ứng dụng phần mềm dựa trên kiến trúc nguyên khối. Chẳng hạn:

|  |  |
| --- | --- |
| **Identity Server dựa trên kiến trúc nguyên khối** (hình 4.4) | **Identity Server dựa trên kiến trúc Microservice** (hình 4.6) |
| Đóng gói 4 dịch vụ thành một khối duy nhất. Hạn chế cho khả năng triển khai, bảo trì và nâng cấp | Tách biệt 4 dịch vụ này thành các microservice nhỏ như hình 3.8. Thuận lợi cho việc triển khai, bảo trình và nâng cấp |
| Triển khai lại toàn bộ hệ thống khi có một thay đổi nhỏ. | Chỉ triển khai một microservice bị thay đổi. |
| Các dịch vụ được đóng gói thành một khối duy nhất, khi một dịch vụ nào đó gặp lỗi thì cả hệ thống bị ảnh hưởng. | Các microservice được triển khai độc lập với nhau. Khi xảy ra lỗi ở một microservice nào đó thì chỉ ảnh hưởng đến chức năng mà microservice đó đảm nhiệm và kéo theo một số microservice khác có giao tiếp đến nó. Hệ thống vẫn hoạt động bình thường khi mất microservice đó thay vì cả hệ thống bị ảnh hưởng. |
| Khó áp dụng công nghệ mới. | Có thể áp dụng công nghệ mới để phù hợp cho từng microservice. |
| Tất cả các dịch vụ được triển khai thành một khối duy nhất, dẫn đến việc quản lý Logs trở nên khó khăn. Vì tất cả các dịch vụ đều ghi Logs của chính mình vào một file chung duy nhất. | Mỗi microservice có một file Logs riêng, giúp dễ dàng quản lý. |
| Tương tự với dịch vụ, cơ sở dữ liệu tập trung nên khi gặp lỗi sẽ ảnh hưởng toàn bộ hệ thống. | Cơ sở dữ liệu phi tập trung |

Bảng . So sánh Identity Server dựa trên hai kiến trúc.

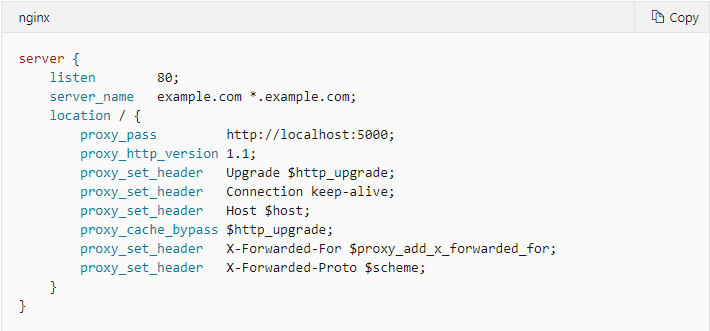
## Xác định, xây dựng Identity Server dựa trên kiến trúc Microservices

### Microservices Core

|  |  |  |
| --- | --- | --- |
| Tên microservice | Chức năng | Ngôn ngữ/Framework |
| Identity Server | Cung cấp token, xác nhận token. Còn có giao diện phục vụ cho dịch vụ Single Sign-On | ASP.NET Core 3.1, Identity Server 4, ASP.NET Identity,  ASP.NET Core Blazor |
| Storage API | Cung cấp API có chức năng lưu trữ và hiển thị các tệp tin do người dùng tải lên | ASP.NET Core 3.1,  Swagger |
| User API | Cung cấp API có chức năng quản lý các tác vụ liên quan đến người dùng | ASP.NET Core 3.1,  ASP.NET Identity, Swagger |
| Identity API | Cung cấp các API có chức năng quản lý các tác vụ liên quan đến client, bảo vệ API, danh tính | ASP.NET Core 3.1,  Identity Server 4,  Swagger |

Bảng . Xác định các microservices và chức năng.

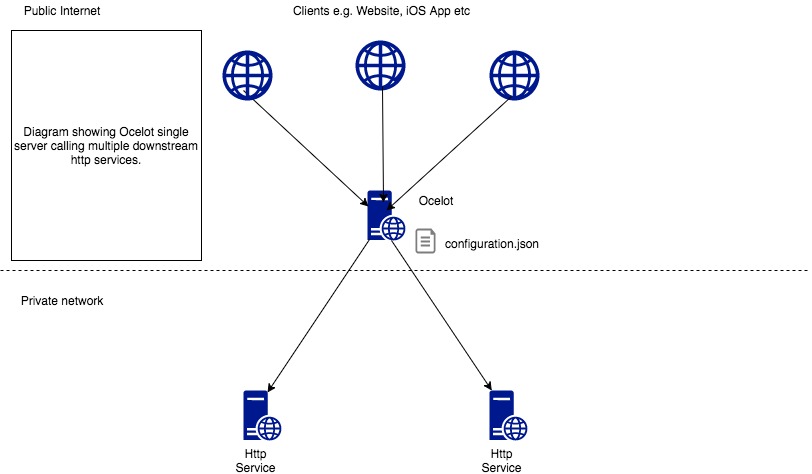
Ngoài ra, các microservice được tích hợp với một ứng dụng web server Nginx. Có chức năng load balancing, cân bằng tải, proxy, bảo mật v.v. Nginx là phần mềm mã nguồn mở và có hiệu năng vô cùng ấn tượng.



Hình . Cấu hình một dịch vụ khi sử dụng Nginx làm chức năng câng bằng tải.

Tất cả Microservice được đóng gói thành Docker Image để triển khai thành các Docker Container thông qua Docker Compose trên cả nền tảng Window, Linux và Mac. Vì thế, việc triển khai các microservice trên tất cả các nền tản là việc vô cùng dể dàng. Ngoài ra, microservice được đóng gói và triển khai trên một Container riêng biệt và độc lập so với các microservice khác, tạo điều kiện cho việc sữa lỗi, nâng cấp và bảo trì.

### API Gateway



Hình . API Gateway Ocelot cho ASP.NET Core

API Gateway làm nhiệm vụ định tuyến các yêu cầu/phản hồi giữa các microservice và ứng dụng client.

Ocelot hướng đến những người sử dụng .NET chạy ứng dụng có kiến trúc hướng dịch vụ SOA hay kiến trúc Microservices cần một điểm vào thống nhất trong hệ thống.

Ocelot là một loạt các phần mềm trung gian theo một thứ tự cụ thể.

Ocelot điều khiển đối tượng HttpRequest thành một trạng thái được chỉ định bởi một cấu hình (khai báo) trước đó của nó cho đến khi nó đạt đến phần mềm trung gian của trình tạo đối tượng HttpRequestMessage được sử dụng để thực hiện yêu cầu đối với các dịch vụ bên dưới (microservcie). Phần mềm trụng gian thực hiện các yêu cầu là điểm cuối cùng trong đường dẫn Ocelot. Nó không gọi phần mềm trung gian tiếp theo. Có một phần mềm trung gian ánh xạ HttpResponseMessage vào đối tượng HttpResponse và được trả lại cho máy khách.



Hình . Khai báo một API với API Gateway Ocelot bằng file Configuration.json.

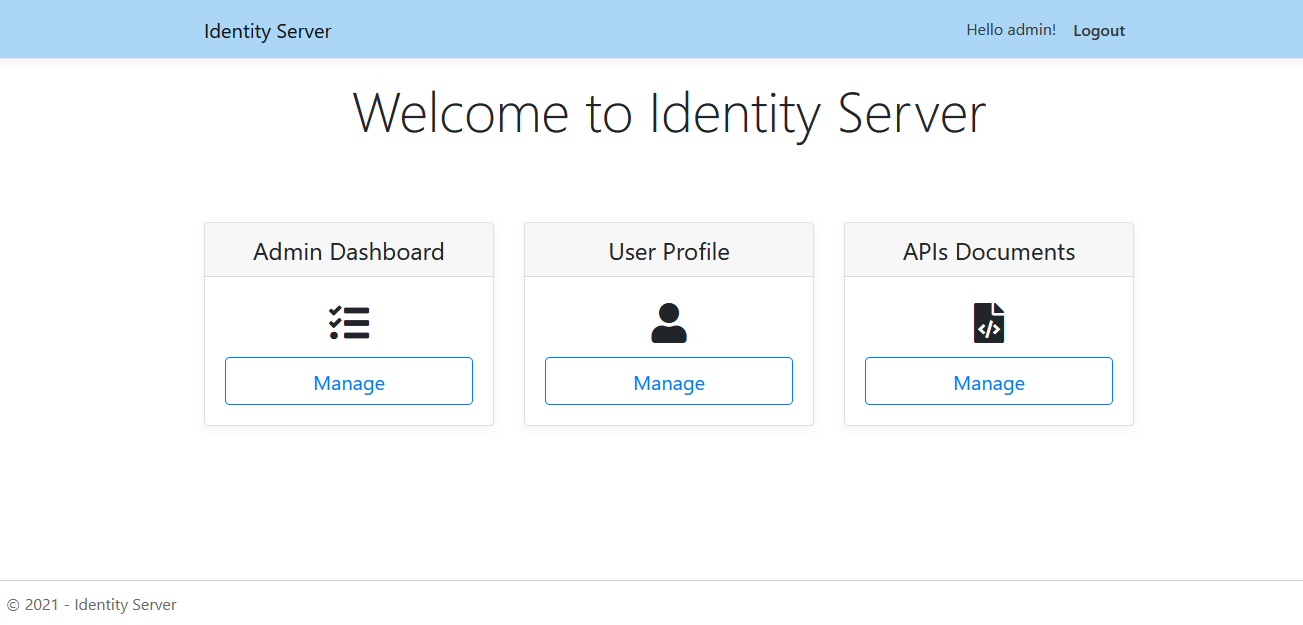
### Các ứng dụng phía client

* Angular - Admin Dashboard: Giao diện quản lý tất cả các dịch vụ liên quan đến xác thực và ủy quyền. Bao gồm: ứng dụng client, tài nguyên API được bảo vệ, các quy định của các tài nguyên, danh tính, thông tin người dùng, các chức năng liên quan đến ủy quyền người dùng, v.v.
* Angular – User Profile: Giao diện cung cấp các chức năng cho từng người dùng riêng biệt có thể xem và chỉnh sửa thông tin cá nhân của chính mình. Ngoài ra còn có thể xóa tài khoản khi không cần dùng đến.
* Blazor – Identity Server: Giao diện đăng nhập và đăng xuất phục vụ chung cho tất cả các ứng dụng client (Single Sign-On).

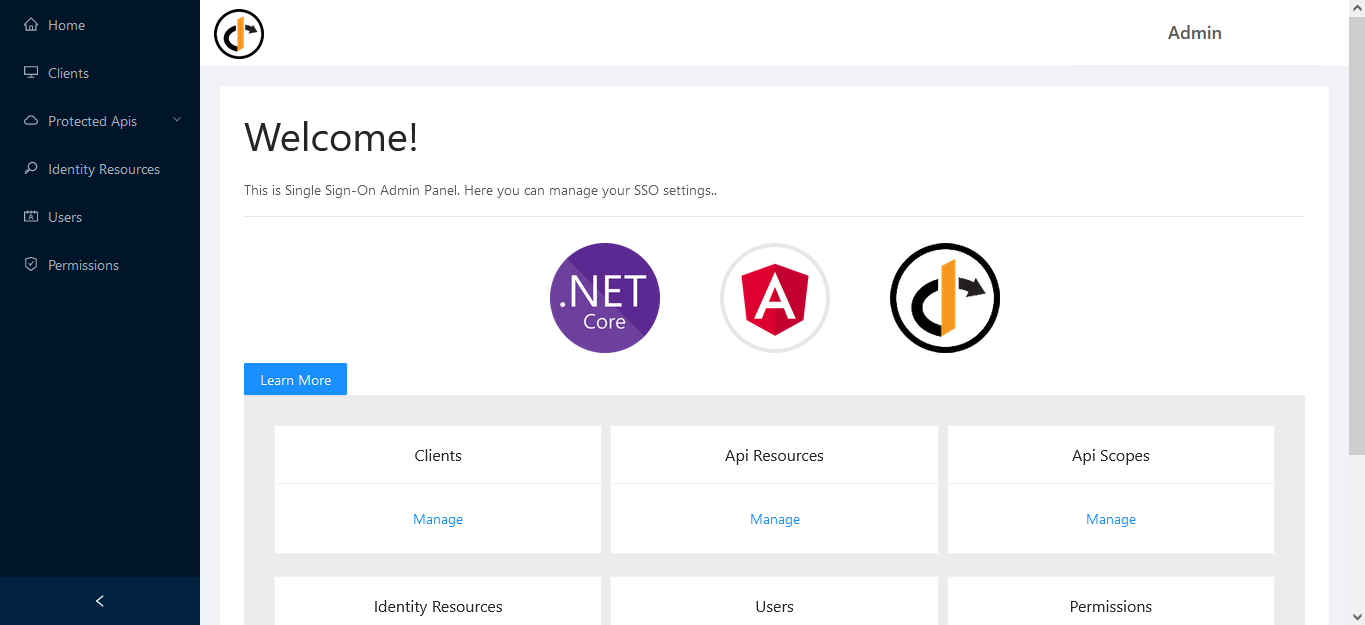
Các ứng dụng client đều được đóng gói với Docker Image và Nginx. Vì thế chúng cũng có khả năng triển khai nhanh gọn và trên tất cả các nền tảng hệ điều hành.

# XÂY DỰNG ỨNG DỤNG

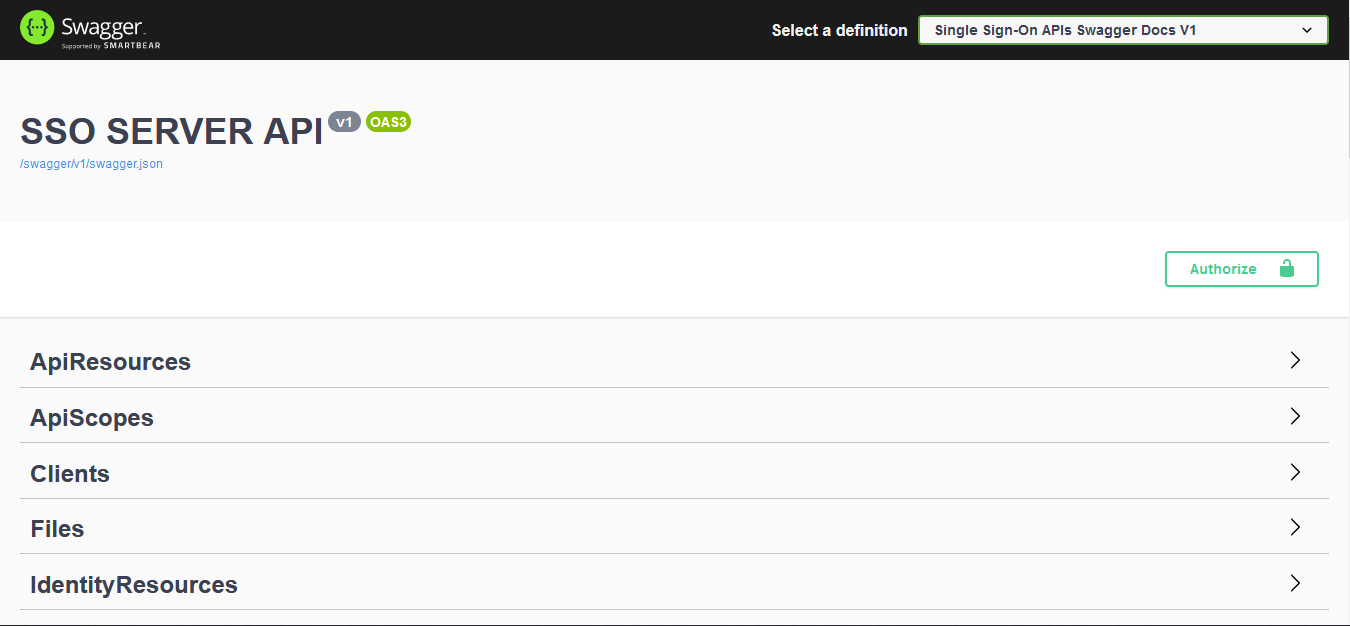
## Giao diện hệ thống



Hình . Trang chủ Identity Server



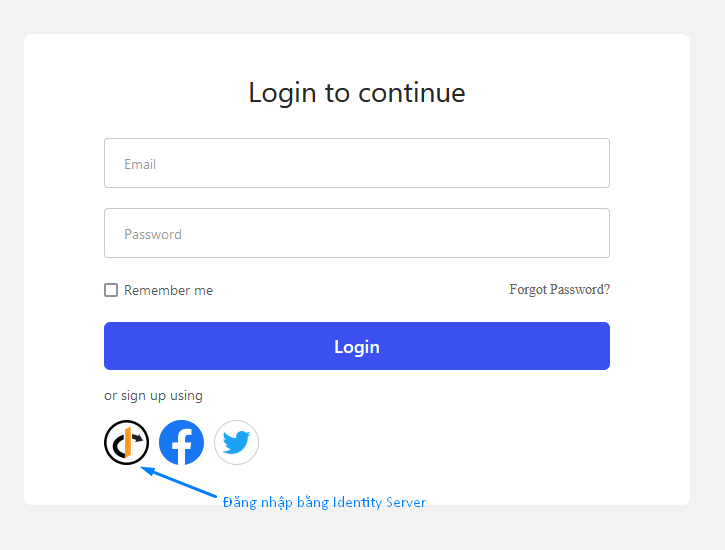
Hình . Trang chủ ứng dụng Admin Dashboard



Hình . Trang Swagger quản lý API Identity Microservice

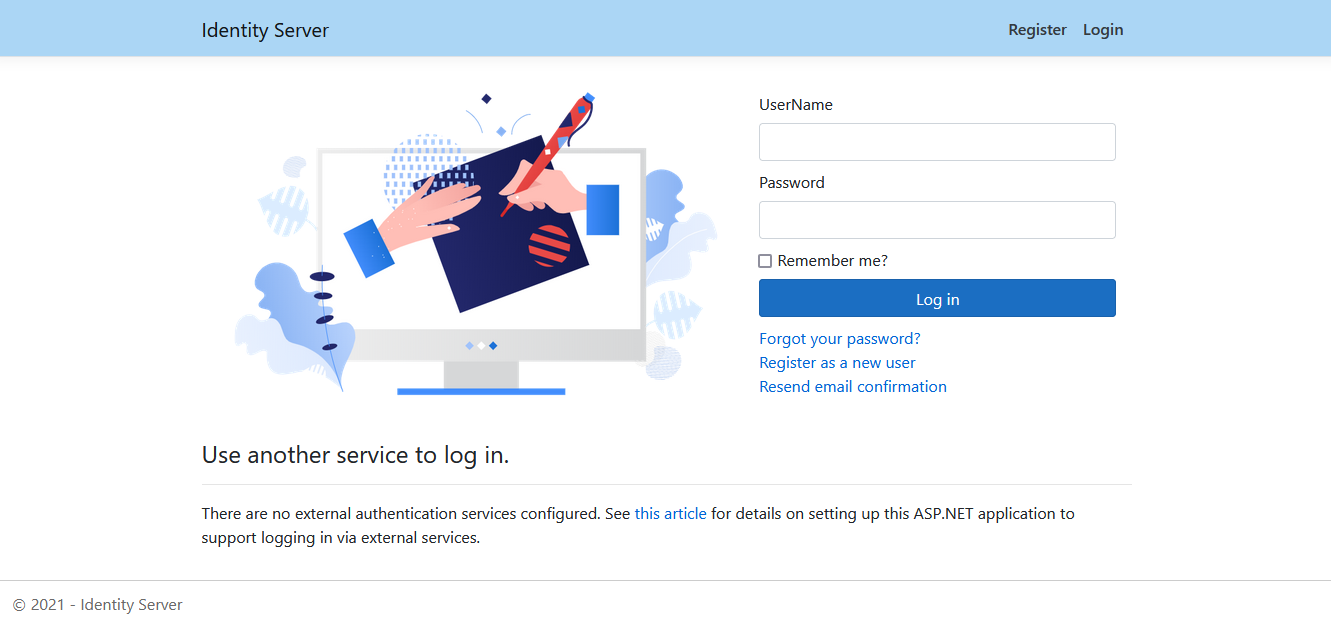
## Đăng nhập vào ứng dụng của hệ thống thông qua Identity Server

* Truy cập các ứng dụng client (Angular – Admin Dashboard, Angular – User Profile) và đăng nhập thông qua Identity Server.



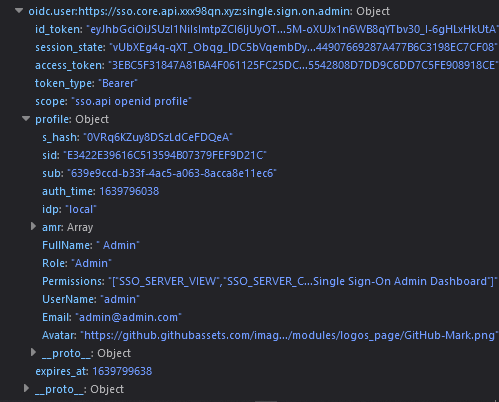
Hình . Đăng nhập vào ứng dụng client

* Hệ thống sẽ tự động chuyển tiếp đến trang đăng nhập của Identity Server (trang đăng nhập SSO), và người dùng cần đăng nhập bằng tài khoản được cấp trước đó hoặc phải tự tạo tài khoản mới



Hình . Đăng nhập vào ứng dụng thông qua Identity Server

* Sau khi đăng nhập thành công, hệ thống tự động quay về ứng dụng trước đó và gửi kèm theo một mã token, các ứng dụng client sử dụng các token này để có quyền truy cập vào các tài nguyên API của hệ thống. Mã token có thể chứa các thông cơ bản và các thông tin bổ sung do nhà phát triển cài đặt thêm.



Hình . Mã Token được gửi về ứng dụng client

## Cấu hình ứng dụng client mới

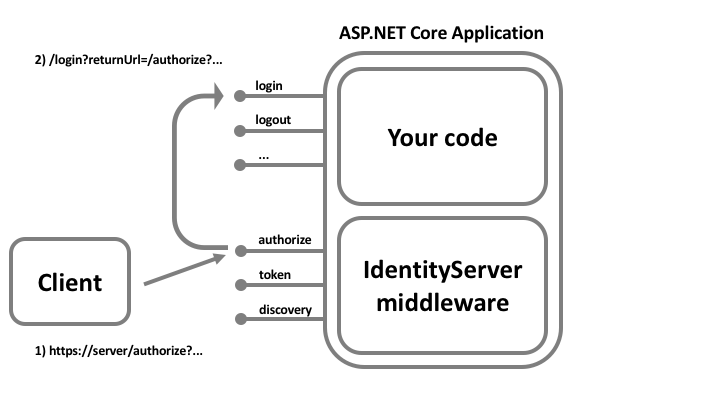
Yêu cầu người dùng có quyền truy cập tất cả các API (mặc định người dùng được phân quyền Admin). Sau khi đăng nhập thành công vào ứng dụng Angular – Admin Dashboard:

* Truy cập vào dịch vụ Clients/Create new Client và tiến hành tạo mới một ứng dụng client với các thông tin cơ bản bao gồm:
* Sau khi tạo thông tin cơ bản thành công. Ứng dụng sẽ chuyển tiếp đến trang cài đặt các thông số chi tiết cho ứng dụng. Để ứng dụng client có thể kết nối được với Identity Server. Người dùng cần cài đặt một số thông số riêng tương ứng với loại ứng dụng đó. Trong bài này sẽ ví dụ đăng ký ứng dụng SPA (Angular) sử dụng OIDC.

|  |  |
| --- | --- |
| Thông tin cơ bản |  |
| Client id | Id duy nhất của ứng dụng client. |
| Enable | Chỉ định ứng dụng được bật hay tắt. |
| Client Secrets | Danh sách khóa bí mật của ứng dụng khách – dùng để truy cập vào máy chủ cung cấp mã token. |
| Require Client Secret (mặc định: true) | Chỉ định xem ứng dụng client này có cần yêu cầu mã bí mật từ máy chủ cung cấp token hay không. |
| Allowed Grant Types | Chỉ định các loại kết nối mà ứng dụng client có thể sử dụng. |
| Required PKCE  (mặc định: true) | Chỉ định ứng dụng client dựa trên Code Flow có phải gửi khóa bằng chứng hay không. |
| Allow Plain Text PKCE  (mặc định: false) | Chỉ định ứng dụng client sử dụng PKCE có thể sử dụng văn bản thuần túy hay không. |
| Redirect Uris | Chỉ định các URL được phép nhận mã token truy cập hoặc mã token ủy quyền. |
| Allowed Scopes | Theo mặc định, ứng dụng client không được phép truy cập bất cứ tài nguyên nào – chỉ định tài nguyên được phép truy cập bằng cách thêm tên tài nguyên tương ứng. |
| Allow Offline Access | Chỉ định ứng dụng client có thể yêu cầu làm mới mã token hay không. |
| Allow Access Tokens Via Browser | Chỉ định ứng dụng client có thể nhận mã token mới thông qua trình duyệt hay không. |
| Properties | Lưu trữ mọi giá trị dành riêng cho ứng dụng khách. |
| Xác thực/Đăng xuất |  |
| Post Logout Redirect Uris | Chỉ định các URL được chuyển hướng đến sau khi đăng xuất. |
| Front Channel Logout Uri | Chỉ định URL chỉ đăng xuất tại ứng dụng client. |
| Front Channel Logout Session Required  (mặc định: true) | Chỉ định xem Id phiên sử dụng của người dùng. |
| Back Channel Logout Uri | Chỉ định URL đăng xuất taị ứng dụng client. |
| Back Channel Logout Session Required  (mặc định: true) | Chỉ định xem Id phiên của người dùng. |
| Enable Local Login  (mặc định là true) | Chỉ định ứng dụng client này có thể sử dụng tài khoản cục bộ hay bên thứ ba. |
| Identity Provider Restrictions | Chỉ định dịch vụ bên thứ ba nào có thể được sử dụng với ứng dụng client này |
| User Sso Lifetime | Thời lượng tối đa (tính bằng giây) kể từ lần cuối cùng người dùng xác thực (mặc định là null) xác định thời gian tồn tại của phiên đăng nhập. |
| Token |  |
| Identity Token Lifetime  (mặc định: 300 giây / 5 phút) | Thời gian tồn tại của mã token nhận dạng tính bằng giây. |
| Allowed Identity Token Signing Algorithms | Danh sách các thuật toán được phép ký cho mã token nhận dạng. Nếu trống, sẽ sử dụng thuật toán ký mặc định của máy chủ. |
| Access Token Lifetime  (mặc định: 3600 giây / 1 giờ) | Thời gian tồn tại của mã token truy cập tính bằng giây. |
| Authorization Code Lifetime  (mặc định: 300 giây / 5 phút | Thời gian tồn tại của mã token ủy quyền tính bằng giây). |
| Absolute Refresh Token Lifetime | Thời gian tồn tại tối đa của mã token làm mới tính bằng giây. |
| Sliding Refresh Token Lifetime | Thời gian trượt của mã token làm mới trong vài giây. |
| Refresh Token Usage | * ReUse xử lý mã token làm mới sẽ giữ nguyên khi làm mới mã token * OneTime xử lý mã token làm mới sẽ được cập nhật khi làm mới mã token. Đây là mặc định. |
| Refresh Token Expiration | * Absolute mã token làm mới sẽ hết hạn vào một thời điểm cố định (được chỉ định bởi Absolute Refresh Token Lifetime) * Sliding khi làm mới mã token, thời gian tồn tại của mã token làm mới sẽ được gia hạn (theo số lượng được chỉ định trong Sliding RefreshToken Lifetime). Thời gian tồn tại sẽ không vượt quá Absolute Refresh Token Lifetime. |
| Update Access Token Claims On Refresh | Nhận hoặc đặt một giá trị cho biết liệu mã token truy cập (và các xác nhận quyền sở hữu của nó) có được cập nhật theo yêu cầu mã token làm mới hay không. |
| Access Token Type  (mặc định: Jwt ) | Chỉ định xem mã token truy cập là mã thông báo tham chiếu hay mã token JWT. |
| Include Jwt Id  (mặc định: true) | Chỉ định xem mã token truy cập JWT có nên có một ID duy nhất được nhúng hay không. |
| Allowed Cors Origins | Nếu được chỉ định, sẽ được sử dụng bởi các triển khai dịch vụ chính sách CORS mặc định để xây dựng chính sách CORS cho các ứng dụng khách JavaScript. |
| Claims | Cho phép xác nhận quyền sở hữu cài đặt cho khách hàng (sẽ được bao gồm trong mã token truy cập). |
| Always Send Client Claims  (mặc định: false) | Nếu được đặt, yêu cầu của khách hàng sẽ được gửi cho mọi luồng. Nếu không, chỉ dành cho luồng thông tin xác thực khách hàng. |
| Always Include User Claims In Id Token  (Mặc định: false). | Khi yêu cầu cả mã thông báo id và mã thông báo truy cập, nếu yêu cầu của người dùng luôn được thêm vào mã thông báo id thay vì yêu cầu khách hàng sử dụng điểm cuối userinfo. |
| Client Claims Prefix | Nếu được đặt, các loại xác nhận quyền sở hữu khách hàng tiền tố sẽ được bắt đầu bằng. Mặc định cho client. Mục đích là để đảm bảo họ không vô tình va chạm với các xác nhận quyền sở hữu của người dùng. |
| Pair Wise Subject Salt | Giá trị được sử dụng trong tạo subjectId theo cặp cho người dùng của ứng dụng client này. |
|  |  |
| Require Consent | Tên hiển thị của ứng dụng client |
| Allow Remember Consent  (mặc định: true) | Chỉ định xem người dùng có thể chọn lưu trữ các quyết định về sự đồng ý hay không. |
| Consent Lifetime  (mặc định: null). | Thời gian tồn tại của sự đồng ý của người dùng trong vài giây. |
| Client Name | Tên hiển thị của ứng dụng client |
| Client Uri | URI để biết thêm thông tin về ứng dụng client |
| Logo Uri | URI cho logo ứng dụng client |
|  |  |
| User Code Type | Chỉ định loại mã người dùng để sử dụng cho máy khách |
| Device Code Lifetime  (mặc định: 300 giây / 5 phút) | Thời gian tồn tại của mã thiết bị tính bằng giây |

Bảng . Thông số của ứng dụng client

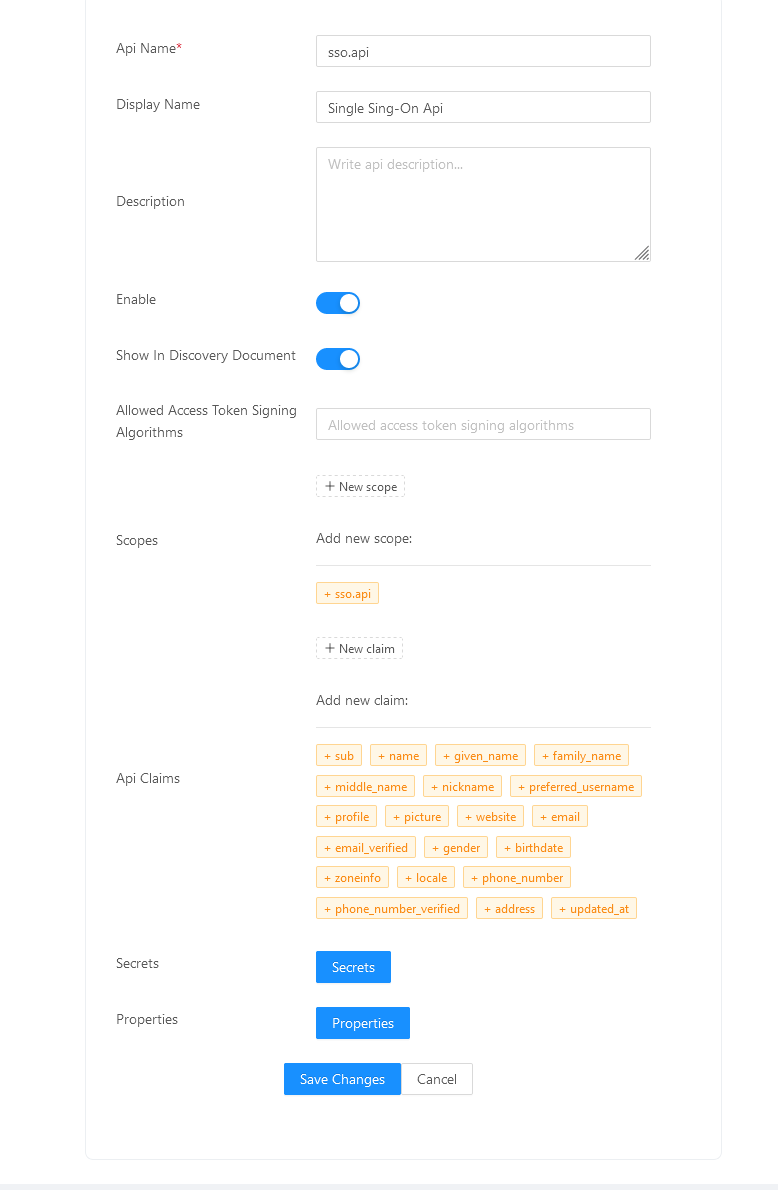
* Sau khi cài đặt các thông số cần thiết, cần thiết lập kết nối từ ứng dụng client với Identity Server sử dụng OIDC Client. Sử dụng tài liệu OpenID Connect.



Hình . Quy trình đăng nhập

## Bảo vệ API bằng Identity Server

### Bảo vệ tài nguyên API



Hình . Bảo vệ API bằng Identity Server

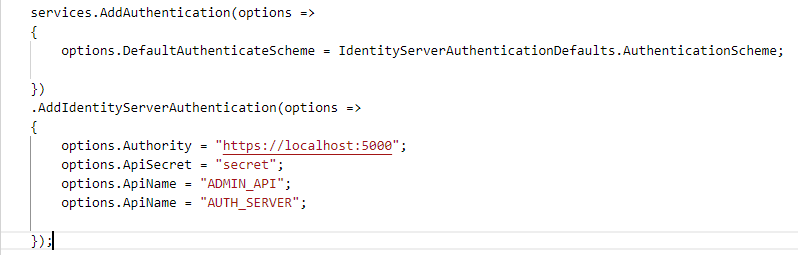
Sử dụng Angular – Admin Dashboard để xác thực các tài nguyên API cần được bảo vệ. Từ dịch vụ Protected APIs/ API Resources có thể thêm mới, xóa phạm vi và chỉnh sửa các thông số chi tiết của API cần bảo vệ. Các thông số kỹ thuật khi bảo vệ API bằng Identity Server:

|  |  |
| --- | --- |
| Enabled  (mặc định: true) | Cho biết nếu tài nguyên này được kích hoạt và có thể được yêu cầu. |
| Name | Tên duy nhất của API. Giá trị này được sử dụng để xác thực với sự xem xét nội tâm và sẽ được thêm vào đối tượng của mã thông báo truy cập gửi đi. |
| Display Name | Giá trị này có thể được sử dụng. |
| Description | Giá trị này có thể được sử dụng. |
| Api Secrets | Bí mật API được sử dụng cho điểm cuối nội quan. API có thể xác thực bằng cách sử dụng tên và bí mật API. |
| Allowed Access Token Signing Algorithms | Danh sách các thuật toán ký được phép cho mã thông báo truy cập. Nếu trống, sẽ sử dụng thuật toán ký mặc định của máy chủ. |
| User Claims | Danh sách các loại xác nhận quyền sở hữu của người dùng được liên kết sẽ được bao gồm trong mã thông báo truy cập. |
| Scopes | Danh sách tên phạm vi API. |

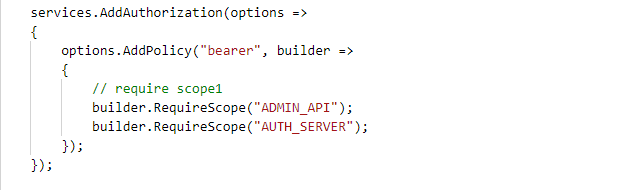
Bảng . Thông số kỹ thuật bảo vệ API bằng Identity Server

Identity Server phát hành mã token truy cập ở định dạng JWT (Json Web Token) theo mặc định.

Mọi nền tảng liên quan ngày nay đều có hỗ trợ xác thực mã JWT. Việc bảo vệ API dựa trên ASP.NET Core chỉ là vấn đề thêm trình xử lý xác thực JWT:

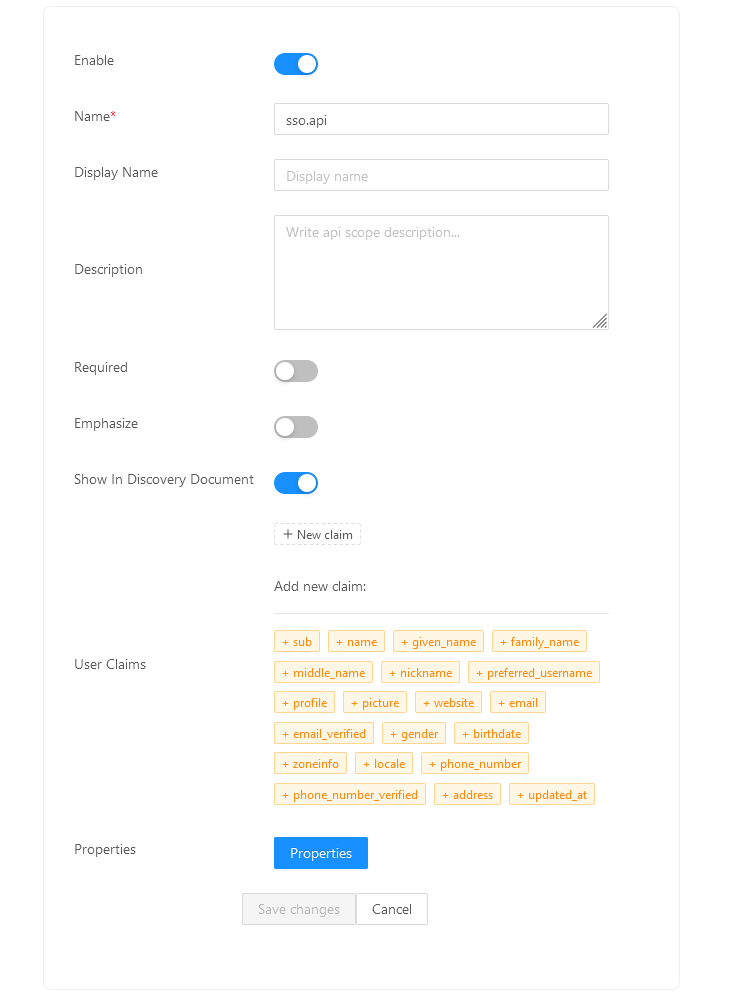


Hình . Trình xử lý mã token xác thực



Hình . Trình xử lý mã token ủy quyền

### Các quy định về phạm vi truy cập tài nguyên API



Hình . Xác định phạm vi API được bảo vệ

Sử dụng Angular – Admin Dashboard để xác thực các tài nguyên API cần được bảo vệ. Từ dịch vụ Protected APIs/ API Scopes có thể thêm mới, xóa phạm vi và chỉnh sửa các thông số chi tiết của phạm vi API cần bảo vệ.

|  |  |
| --- | --- |
| Enabled  (Mặc định: true) | Cho biết tài nguyên này được kích hoạt và có thể truy cập. |
| Name | Tên duy nhất của API. Giá trị này được sử dụng để xác thực với sự xem xét nội tâm và sẽ được thêm vào đối tượng của mã token truy cập gửi đi. |
| Display Name | Giá trị này có thể được sử dụng, ví dụ như trên màn hình chấp thuận. |
| Description | Giá trị này có thể được sử dụng, ví dụ như trên màn hình chấp thuận. |
| User Claims | Danh sách các loại xác nhận quyền sở hữu của người dùng được liên kết sẽ được bao gồm trong mã thông báo truy cập. |

Bảng . Các thông số xác định phạm vi truy cập API

## Tài nguyên nhận dạng

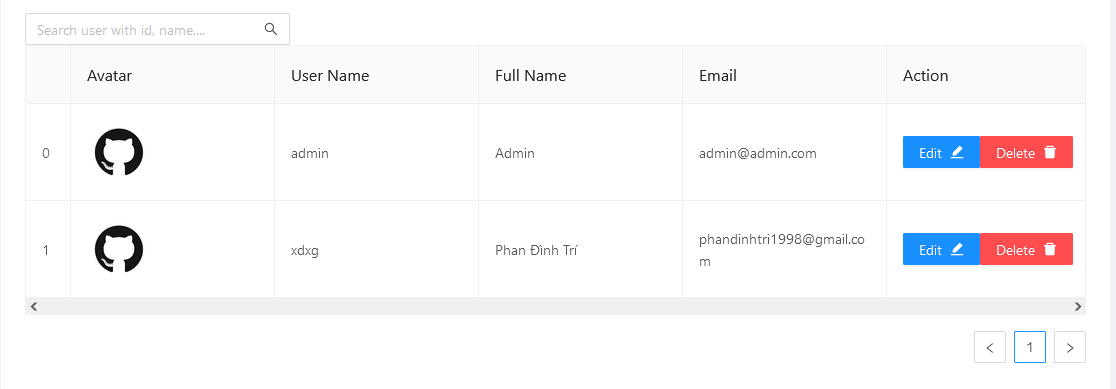
Tài nguyên nhận dạng là một nhóm xác nhận quyền sở hữu được đặt tên có thể được yêu cầu bằng cách sử dụng các tham số liên quan đến phạm vi.

Thông số kỹ thuật OpenID Connect gợi ý một vài tên phạm vi tiêu chuẩn để xác nhận loại ánh xạ có thể hữu ích hoặc có thể tự do thiết kế chúng.

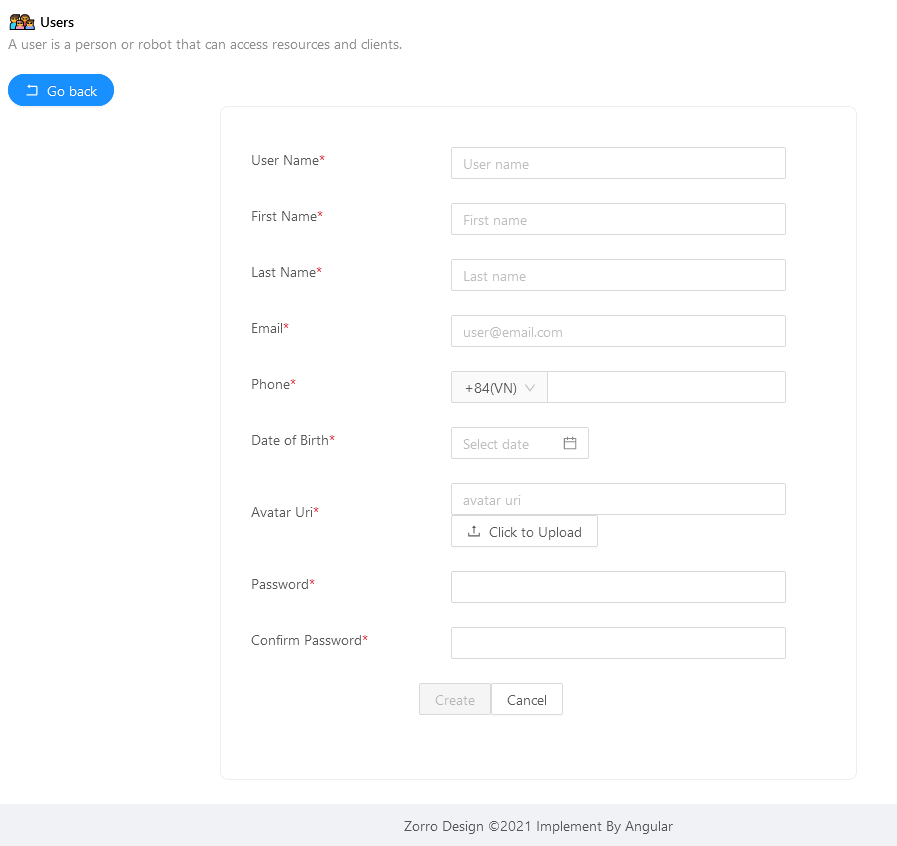
Một trong số chúng thực sự bắt buộc, phạm vi “openid”, yêu cầu nhà cung cấp trả lại các yêu cầu phụ trong mã token

## Quản lý người dùng

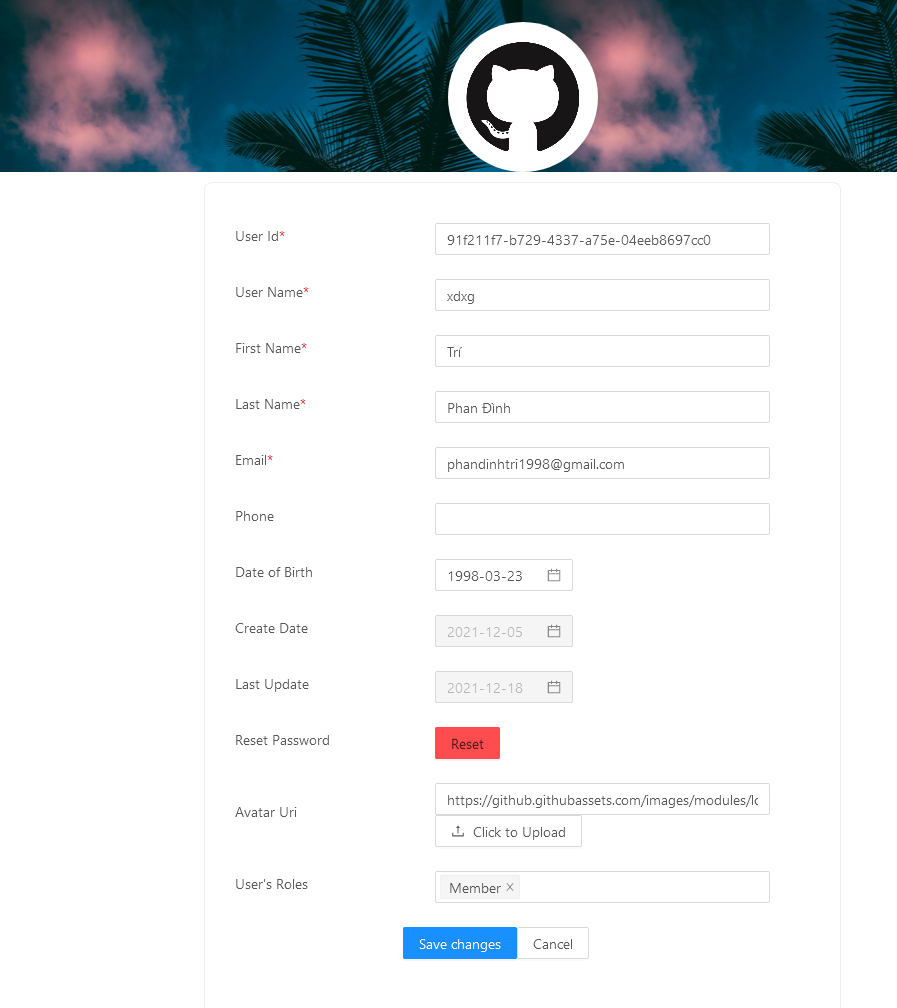
Angular – Admin Dashboard cung cấp khả năng quản lý tất cả các tác vụ liên quan đến người dùng. Identity Server sử dụng thư viện ASP.NET Identity, cung cấp mọi chức năng cơ bản mà một ứng dụng quản lý người dùng thường có và nhà phát triển có thể bổ những yêu cầu riêng.



Hình . Danh sách thông tin người dùng



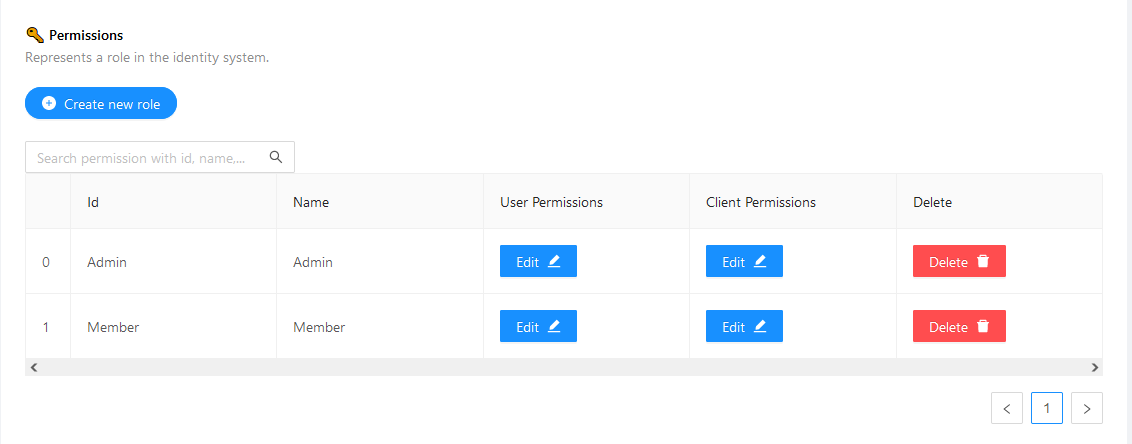
Hình . Thêm mới người dùng



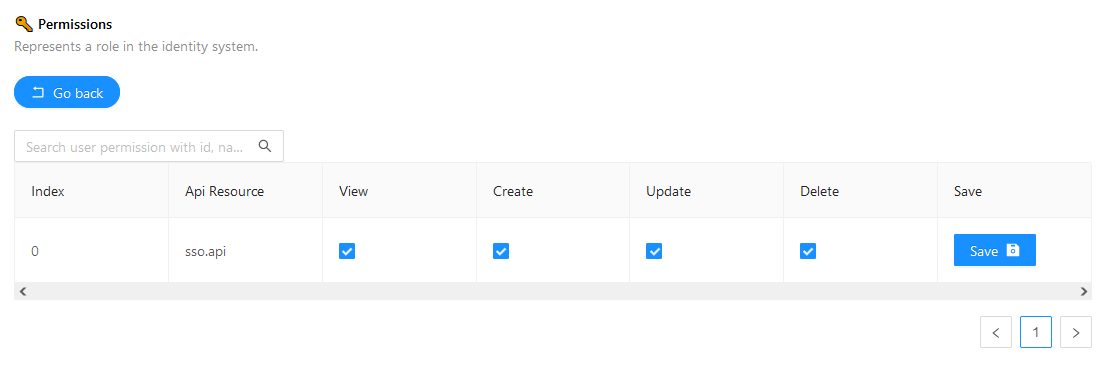
Hình . Chỉnh sửa thông tin người dùng

## Quản lý phân quyền người dùng

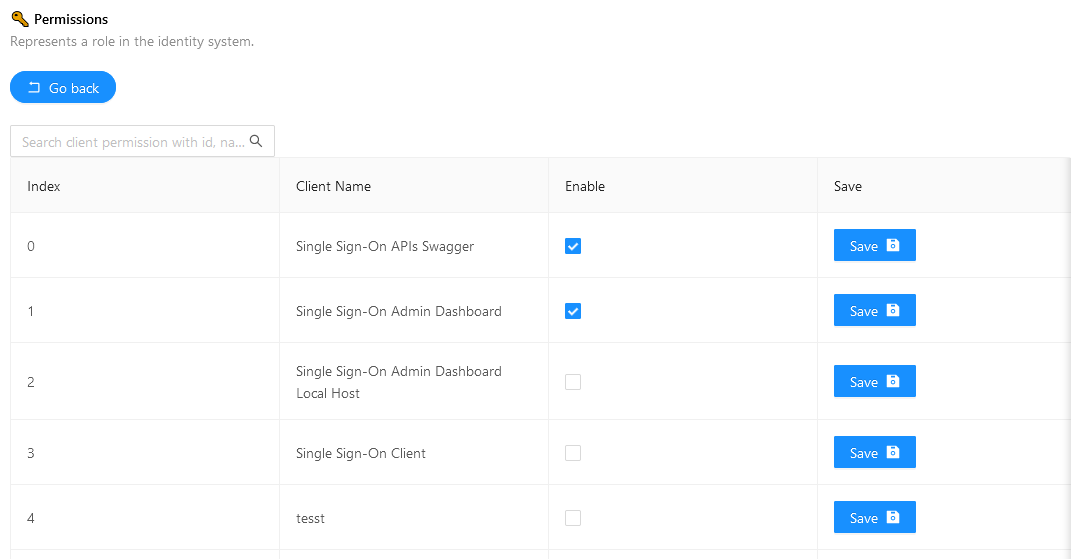
Angular – Admin Dashboard cung cấp các chức năng quản lý ủy quyền cho từng người dùng truy cập vào các API, ứng dụng client. ASP.NET Identity cung cấp các khả năng có sẵn và có thể tùy biến thêm theo ý của nhà phát triển.



Hình . Quản lý thông tin các nhóm phân quyền



Hình . Chỉnh sửa thông tin chi tiết ủy quyền truy cập API



Hình . Chỉnh sửa thông tin chi tiết ủy quyền truy cập ứng dụng client

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết quả đạt được

* Xác định được các ưu điểm và nhược điểm kiến trúc phần mềm nguyên khối, kiến trúc ứng dụng phần mềm hướng dịch vụ và kiến trúc Microservices.
* Áp dụng kiến trúc Microservices hướng API vào trong các ứng dụng hiện đại.
* Phân tách ứng dụng nguyên khối thành ứng dụng có kiến trúc Microservices.
* Thực hành với các công nghệ phổ biến của Microservice bao gồm: Docker, RESTful API, API Gateway, v.v.
* Thực hành với ứng dụng web Angular 10.
* Xây dựng máy chủ nhận dạng cung cấp dịch vụ xác thực và ủy quyền tập trung.

## Các vấn đề còn tồn tại

* Giao diện người dùng không được đẹp mắt người dùng. Chỉ tập trung vào các chức năng là chính.
* Xây dựng và thiết kế API ở một số dịch vụ chưa tuân thủ theo tiêu chuẩn REST
* Báo cáo còn nhiều sai sót.
* Tốc độ hệ thống cần được cải thiện.

## Đề xuất hướng phát triển

Trong quá trình làm đồ án, bản thân đã tự học hỏi, tìm hiểu trau dồi kiến thức từ việc tra cứu đọc tài liệu, cũng như các kiến thức đã học tập được từ quý thầy cô giáo. Mong muốn trong tương lai sắp tới, đồ án sẽ được mở rộng theo các hướng sau:

* Tiếp tục nghiên cứu phát triển hệ thống, và tích hợp vào các hệ thống của các doanh nghiệp.
* Tối ưu hệ thống, tối ưu giao diện, hoạt động tốt trên các nền tảng web.

**TÀI LIỆU THAM KHẢO**

1. RSA:https://vi.wikipedia.org/wiki/RSA\_(m%C3%A3\_h%C3%B3a)
2. Angular: <https://angular.io/>
3. Ant Design of Angular: <https://ng.ant.design/docs/introduce/en>
4. Swagger for Netcore: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio>
5. ASPNET Identity: <https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity#:~:text=ASP.NET%20Identity%20does%20not,users%20in%20the%20web%20site.>
6. Identity Server 4: <https://identityserver4.readthedocs.io/en/latest/>
7. JP Project for SSO: <https://github.com/brunohbrito/JPProject.IdentityServer4.SSO>
8. Boostrap for Angular: <https://ng-bootstrap.github.io/#/home>
9. vietname-dev: <https://github.com/vietnam-devs>
10. OAuth 2.0: <https://www.oauth.com/>
11. OpenID Connect: <https://openid.net/connect/>
12. OIDC client: <https://github.com/IdentityModel/oidc-client-js>
13. How Single Sign-On work: <https://www.onelogin.com/learn/how-single-sign-on-works>
14. Microservices in Practice: <https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/>
15. Microservices architecture style: <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>
16. Nguyên tắc thiết kế REST API: <https://medium.com/eway/nguy%C3%AAn-t%E1%BA%AFc-thi%E1%BA%BFt-k%E1%BA%BF-rest-api-23add16968d7>
17. Monolithic vs Microservices architecture: <https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/?ref=rp>
18. Fullstack mark blog: <https://fullstackmark.com/>
19. RSA https://vi.wikipedia.org/wiki/RSA\_(m%C3%A3\_h%C3%B3a)