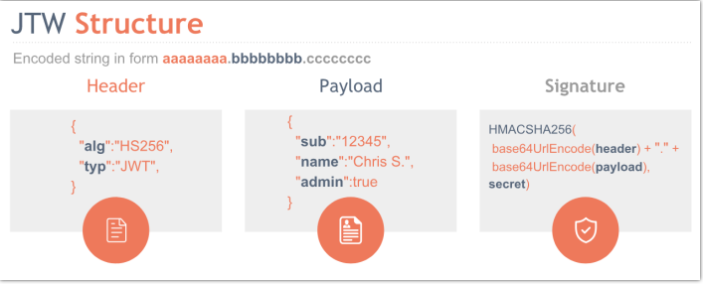
## OpenID Connect

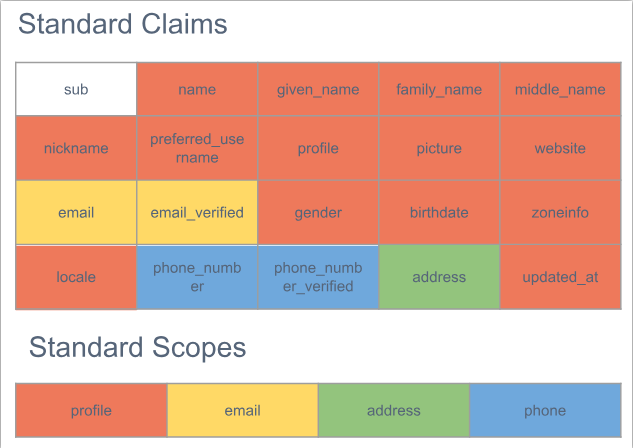
Khi mô tả OAuth 2.0 chúng ta đã nói rằng mục đích của nó là để phát access tokens để cung cấp limited access tới protected resources. Nói cách khác thì OAuth 2.0 cung cấp authorization nhưng nó không cung cấp authentication

### ID Token & JWT



### Claims Scopes

Claim là một đơn vị thông tin cá nhân trong một cặp key-value. Scopes được sử dụng để yêu cầu chính xác các tập claims. OpenId scope bắt buộc scope để chỉ định cái mà OpenID Connect sẽ được sử dụng. OpenID Connect định nghĩa một chuẩn các tập của profile claims



### OAuth 2.0 & OpenID Connect Endpoints

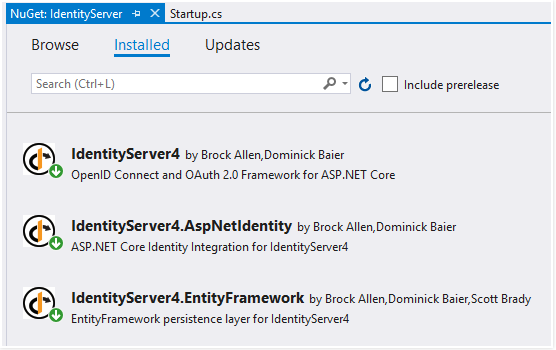
* Authorization endpoit – Được sử dụng bởi client để lấy được authorization từ resource owner thông qua user-agent redirection. Thực hiện Authentication của End-User cái mà đã được chuyển hướng thông qua User-Agent. Đây là endpoint nơi mà bạn chuyển hướng khi click Login
* Token endpoint: Được sử dụng bởi client để trao đổi một authorization grant cho access token. Nó trả về access token, id token nếu trong trường hợp nó là OpenID Connect, và một tùy chọn refresh token
* UserInfo endpoint – Được thêm vào trong trường hợp sử dụng OpenID Connect với mục đích để lấy được claims của end-user đã xác thực.
* Client endpoint – Thằng này là một endpoint thuộc về client. Nó được sử dụng thông qua bởi authorization server để trả về phản hồi quay lại client thông qua user-agent

## IdentityServer4

Nó sẽ mất rất nhiều công sức để thực thi tất cả specs đã định nghĩa bởi OAuth 2,0 và OpenID Connect bởi chính ta, may mắn thay, ta không cần làm điều này bởi vì ta có IdentityServer. Tất cả những thức mà IdentityServer làm là thêm các spec OpenID Connect và OAuth 2.0 endpoints tới ASP.NET Core application thông qua middleware. Điều này có nghĩa rằng bằng việc thêm middleware của nó tới pipeline của application ta nhận được authorization và token endpoints chúng ta đã nói tới và tất cả tính năng core cần thiết (redirecting, granting access, token validation, etc...) cho việc thực thi spec. Tất cả thứ mà ta cần làm là cung cấp các basic pages như Login, Logout views.

### Authorization Server Setup

Tạo một project .NET Core Web Application. Sau đó install các package như sau

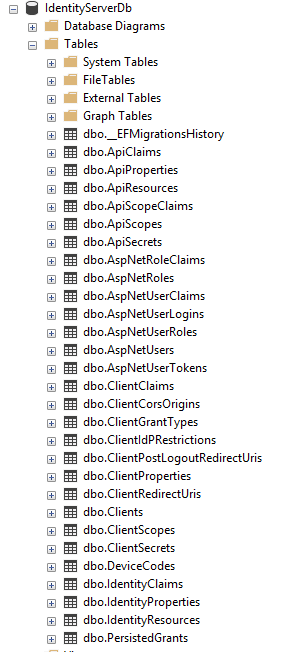


**IdentityServer4.AspNetIdentity** cung cấp config API để sử dụng ASP.NET Identity management library.

**IdentityServer4.EntityFramework** cung cấp một thực thi EF cho việc config và hoạt động lưu trữ trong IdentityServer.

* Configuration data: Data cho việc định nghĩa resources và clients
* Operational data: Data đã sản xuất bởi IdentityServer, như là tokens, codes and consents

Khi bạn tích hợp EntityFramework nó có nghĩa rằng database sẽ chứa tất cả các tables được yêu cầu để IdentityServer làm việc



Chúng được xử lý bởi 2 DbContext classes là PersistedGrantDbContext và ConfigurationDbContex

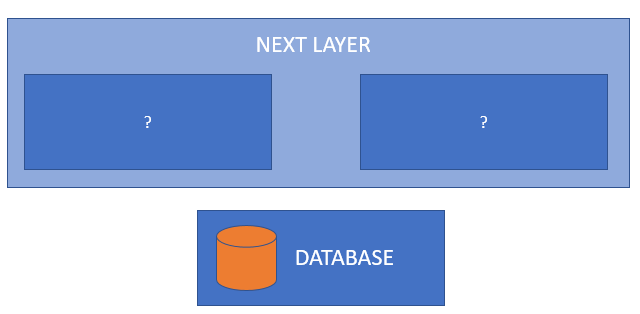
# ASP.NET Core Identity Series – Getting Started

ASP.NET Core Identity là thành viên trong hệ thống của Microsoft về việc quản lý application users. Mọi thức cái mà làm việc với một user account như là create one, login, resetting password, using external login providers hoặc cung cấp khả năng truy cập tới resource nhất định.

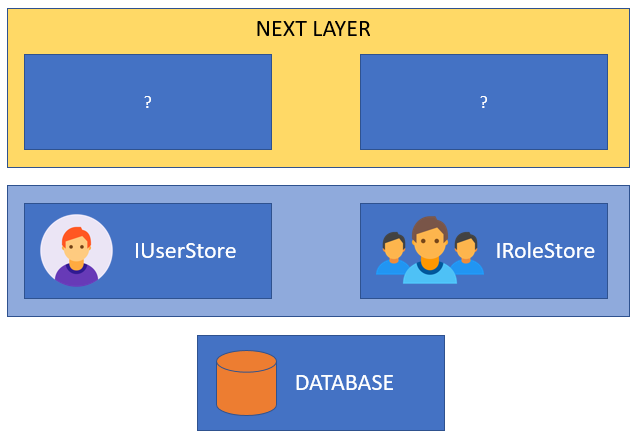
Chúng ta sẽ học ASP.NET Core Identity library’s core components và kiến trúc của chúng và user management

## ASP.NET Core Identity Basics

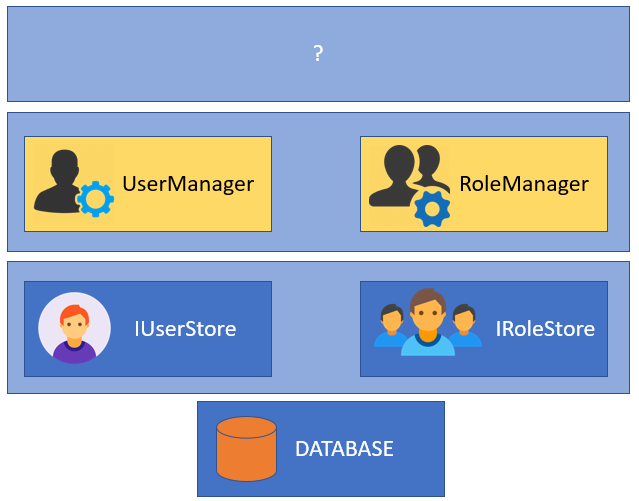
Trong thực tế nhiều developers nhầm lần về vai trò của ASP.NET Core Identity trong stack bởi việc nghĩ rằng nó là một authentication library. Thực sự không phải thế, nhưng mà từ đâu mà có giả thuyết này? Câu trả lời được ẩn bên dưới kết cấu của library vì vậy ta sẽ điều tra nó từ dưới lên trên. Tại bên dưới cùng tầng kiến trúc là một store đây thường là database



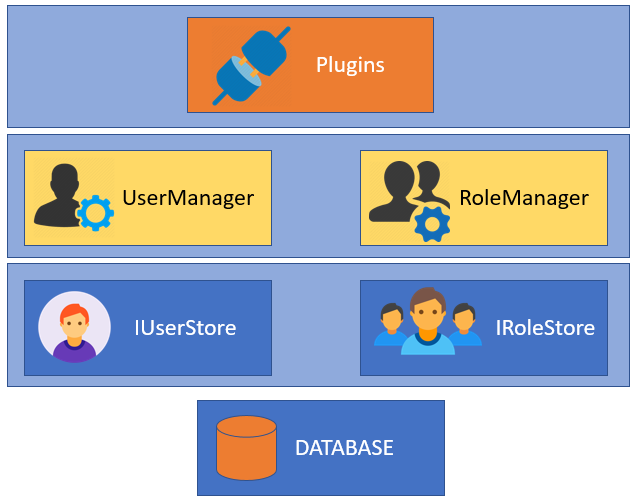
Đây là nơi mà dữ liệu user được chứa như username, email, hash passwords. Layer tiếp theo là **data access** đây là nơi mà thực thi của IUserStore và IRoleStore interfaces



Đây là nơi bạn có lẽ viết data access layer cái mà lưu và quản lý users trên kho lưu trữ và custom schema. IUserStore là một dependency được yêu cầu cho thằng layer tiếp theo để làm việc, có nghĩa là ta phải luôn cung cấp implementation cho library để làm việc. Trong trường hợp đẹp, EF cung cấp một IUserStore implementation trong đó model một user là một IdentityUser trong database. Layer tiếp theo là một business layer.



Các managers này giữ toàn bộ business logic như là validating user passwords dựa trên configuration hoặc kiểm tra một user với username không tồn tại trong database trong khi đăng kí. Bên dưới nắp của managers tạo các cuộc gọi tới data access layer. Layer cuối cùng là một tập các extensions cái mà sử dụng để gọi plugins. Phần lớn plugin được sử dụng là SignInManager cái mà quản lý sign in cho các users



Các extensions này ngồi ngay bên trên managers layer. Việc một user sử dụng external login thì là một vies đụ đơn giản của các extensions. Vai trò của ASP.NET Core không xác thực users nhưng quản lý chúng. Nhưng các extensions có thể làm và thêm tính năng như xác thực users