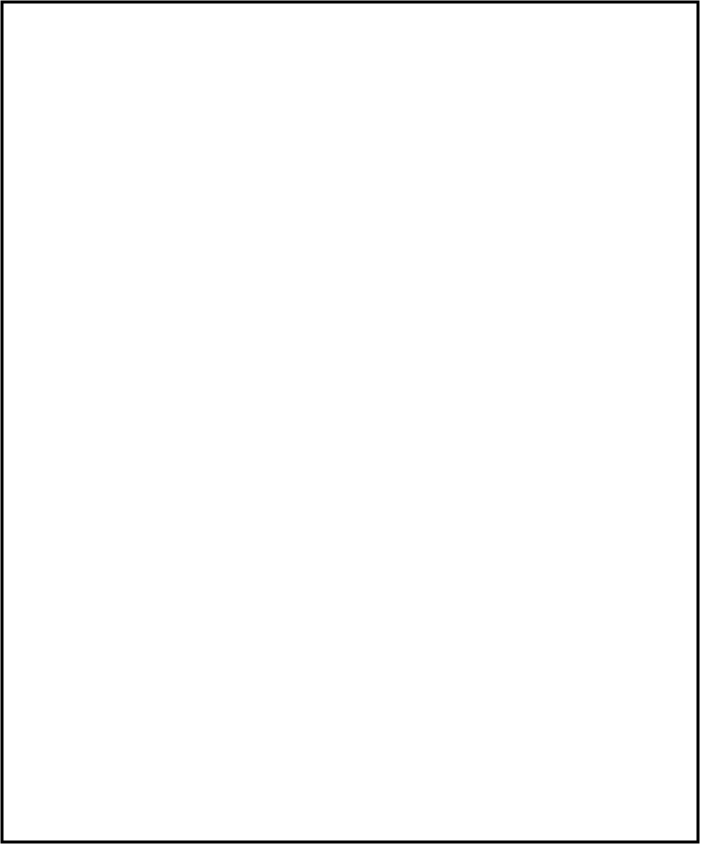
****

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

**KHOA CÔNG NGHỆ PHẦN MỀM**

--------------------------------



**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH**

**NGHIÊN CỨU ASP.NET CORE VÀ TẠO WEBSITE MINH HỌA**

Sinh viên thực hiện: Nguyễn Tiến Đạt

MSSV: 16520199

Lớp: KTPM2016

Người hướng dẫn: Thầy Nguyễn Công Hoan

*Thành phố Hồ Chí Minh – Ngày 1 Tháng 7 Năm 2020*

**Mục lục**

[**1.** **Giới thiệu đề tài** 1](#_Toc46155862)

[**1.1** **Dẫn nhập:** 1](#_Toc46155863)

[**1.2** **Bài toán:** 1](#_Toc46155864)

[**1.3** **Vai trò, xu thế:** 2](#_Toc46155865)

[**1.4** **Giới hạn/ Phạm vi:** 4](#_Toc46155866)

[**1.5** **Các nội dung nghiên cứu và thực hiện:** 4](#_Toc46155867)

[**2.** **Công Nghệ Đã Sử Dụng:** 7](#_Toc46155868)

[**2.1** **Frontend:** 7](#_Toc46155869)

[**2.2** **Backend:** 13](#_Toc46155870)

[2.2.1. ASP.NET core là gì: 13](#_Toc46155871)

[**2.2.2. Lịch sử ra đời:** 13](#_Toc46155872)

[2.2.3. **Kiến trúc**: 16](#_Toc46155873)

[2.2.4. **Tổ chức mã trong dự án**: 20](#_Toc46155874)

[2.2.5. **Tính năng Dependence Injection được hỗ trợ sẵn**: 22](#_Toc46155875)

[2.2.6. Quản lý vòng đời (lifetime) Service: 23](#_Toc46155876)

[2.2.7. **MIDDLEWARE**: 23](#_Toc46155877)

[2.2.8. **Hệ Thống Log**: 25](#_Toc46155878)

[**2.2.9. WebAPP với Razor:** 28](#_Toc46155879)

[**2.2.10. WebAPP với MVC:** 30](#_Toc46155880)

[**2.2.11. WebAPP với Blazor:** 31](#_Toc46155881)

[**2.2.12. WebAPI app:** 31](#_Toc46155882)

[**2.2.13. Quản lý API với Swagger:** 32](#_Toc46155883)

[**2.2.14. RealTime APP:** 33](#_Toc46155884)

[**2.2.15. gPRC:** 35](#_Toc46155885)

[**2.2.16. Testing trong ASP.NET core:** 38](#_Toc46155886)

[**2.2.17. DATA ACCESS:** 39](#_Toc46155887)

[**2.2.18. Host và Deploy:** 41](#_Toc46155888)

[**2.2.19. Sercurity và Identity:** 43](#_Toc46155889)

[**2.2.20. Data Protection:** 47](#_Toc46155890)

[**2.2.21. Một số tính năng nâng cao:** 51](#_Toc46155891)

[**2.3** **Restful API:** 53](#_Toc46155892)

[**2.4** **Sercurity:** 55](#_Toc46155893)

[2.4.1 Giới thiệu Json Web Token (JWT) 55](#_Toc46155894)

[**3.** **XÂY DỰNG ỨNG DỤNG** 57](#_Toc46155895)

[**3.1** **Tổng quan khảo sát các ứng dụng liên quan** 57](#_Toc46155896)

[**3.2** **Cách thiết kế database, cách cài đặt phần mềm** 58](#_Toc46155897)

[3.2.1 Database: 58](#_Toc46155898)

[3.2.2 FrontEnd: 61](#_Toc46155899)

[3.2.3 BackEnd: 61](#_Toc46155900)

[**3.3** **Usecase Diagram** 62](#_Toc46155901)

[**3.4** **Sequence Diagram** 62](#_Toc46155902)

[3.4.1. Login: 62](#_Toc46155910)

[3.4.2 Xem thông tin: 63](#_Toc46155911)

[3.4.3 Chuyển khoản: 63](#_Toc46155912)

[3.4.4 Đăngký: 64](#_Toc46155913)

[3.4.5 Xem thông tin giao dịch: 64](#_Toc46155914)

[3.4.6 Đổi mật khẩu, Đổi tên tài khoản: 65](#_Toc46155915)

[**3.5** **Ứng dụng của công nghệ vào trong ứng dụng:** 66](#_Toc46155916)

[**4** **THỬ NGHIỆM VÀ ĐÁNH GIÁ** 67](#_Toc46155917)

[**4.1** **Môi trường thử nghiệm:** 68](#_Toc46155920)

[**4.2** **Giao diện:** 68](#_Toc46155921)

[**4.2.1** **Đăng ký:** 68](#_Toc46155922)

[**4.2.2** **Đăng nhập:** 69](#_Toc46155923)

[**4.2.3** **Thông tin tài khoản:** 69](#_Toc46155924)

[**4.2.4** **Chuyển khoản:** 70](#_Toc46155925)

[**4.2.5** **Đổi mật khẩu:** 70](#_Toc46155926)

[**4.2.6** **Lịch sử giao dịch:** 71](#_Toc46155927)

[**4.2.7** **Đổi tên đăng nhập:** 71](#_Toc46155928)

[**4.3** **Kết quả đạt được:** 71](#_Toc46155929)

[**4.4** **Hạn chế và hướng phát triển:** 72](#_Toc46155930)

[**5** **TÀI LIỆU THAM KHẢO:** 72](#_Toc46155931)

[**Tiếng việt:** 72](#_Toc46155932)

[**Tiếng anh:** 73](#_Toc46155933)

# **LỜI CẢM ƠN**

Trong thời gian làm đồ án chuyên ngành, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến Thầy Nguyễn Công Hoan, người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình hoàn thành đồ án.

Em cũng xin chân thành cảm ơn các thầy cô giáo trong trường ĐH Công Nghệ Thông Tin nói chung, các thầy cô trong Khoa Công Nghệ Phần Mềm nói riêng đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập.

Cuối cùng, em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên em trong suốt quá trình học tập và hoàn thành khoá đồ án chuyên ngành.

**Sinh Viên Thực Hiện**

***Nguyễn Tiến Đạt***

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

1. **Giới thiệu đề tài**
   1. **Dẫn nhập:**

E-Banking là chữ viết tắt của Electronic-banking (dịch vụ ngân hàng điện tử), một công cụ tiện ích cung cấp các sản phẩm và dịch vụ bán lẻ mới lẫn truyền thống của ngân hàng thông qua các phương tiện điện tử và các kênh truyền thông tương tác khác, bao gồm:

\* Tiến hàng giao dịch ngân hàng.

\* Kiểm tra tài khoản.

\* Thanh toán các hóa đơn điện tử.

\* Cung cấp sản phẩm và dịch vụ thanh toán điện tử khác như tiền điện tử.

* 1. **Bài toán:**

Trước đây, khi nói đến ngân hàng thì người ta thường nghĩ đến thủ tục hành chính rườm rà, phức tạp, phải ký đủ thứ giấy tờ liên quan để đảm bảo quyền lợi giữa ngân hàng và người giao dịch. Hơn nữa, khi giao dịch tại các chi nhánh thì rủi ro mất mát, nguy hiểm khi giao dịch với số lượng lớn là điều hoàn toàn có thể xảy ra bất cứ lúc nào.

Hệ thống ngân hàng thương mại (NHTM) tại Việt Nam trong thời gian qua đã có nhiều thay đổi tích cực trong việc phát triển các dịch vụ của mình để phục vụ ngày càng tốt hơn nhu cầu của khách hàng. Việc đưa các sản phẩm dịch vụ của ngân hàng điện tử vào ngân hàng làm đa dạng hóa các sản phẩm, dịch vụ ngân hàng truyền thống, mở ra nhiều cơ hội mới, khả năng cạnh tranh mới. Đặc biệt, sự xuất hiện của dịch vụ Internet banking là một thành quả hữu hiệu, đã phá vỡ những rào cản và giới hạn về không gian, thời gian trong việc cung cấp các dịch vụ ngân hàng. Dịch vụ Internet banking ra đời là một xu thế tất yếu, đáp ứng được nhu cầu thanh toán qua mạng internet của người dân Việt Nam. Tuy nhiên, do thói quen tiêu dùng tiền mặt cũng như tâm lý còn e ngại việc bảo mật thông tin cá nhân, và sợ bị mất cấp thông tin tài khoản khi sử dụng dịch vụ Internet banking của người dân Việt Nam nên dịch vụ ngân hàng này chưa phát triển mạnh.

* 1. **Vai trò, xu thế:**

Theo Hiệp hội doanh nghiệp phần mềm Việt Nam (VINASA), số lượng người dùng Internet tại Việt Nam có tốc độ tăng trưởng rất nhanh trong giai đoạn 2006-2010. Cụ thể, năm 2006, số lượng người dùng Internet tại Việt Nam là 6,3 triệu, nhưng đã tăng lên 20,7 triệu, đưa Việt Nam trở thành một trong 20 nước có số người dùng mạng nhiều nhất thế giới. Và đặc biệt, tính đến 31/12/2013, số lượng người dùng Internet tại Việt Nam đã tăng lên thành 33 triệu (Bộ thông tin và truyền thông, 2014). Rõ ràng, khi mà Internet đã xuất hiện hầu hết ở mọi cơ quan, công sở trong thời gian vừa qua thì dịch vụ ngân hàng điện tử (E-banking) trở nên một xu hướng phát triển tất yếu, nhiều tiềm năng tại Việt Nam. Ngân hàng điện tử là một trong nhiều ứng dụng công nghệ hiện đại của ngân hàng với khả năng xử lý thông tin trực tuyến, dịch vụ ngân hàng điện tử cung cấp các dịch vụ thanh toán và truy vấn trực tuyến cho mọi đối tượng khách hàng cá nhân và doanh nghiệp. Các dịch vụ này mang lại nhiều thuận tiện cho người sử dụng, giúp khách hàng tiết kiệm thời gian và giao dịch đơn giản. Internet banking được xây dựng như một kênh giao dịch tài chính - ngân hàng thông qua Internet dành cho mọi đối tượng khách hàng cá nhân và doanh nghiệp; giúp khách hàng làm chủ nguồn tài chính mọi lúc, mọi nơi. Với số lượng người sử dụng Internet ngày càng tăng như trên là dấu hiệu đáng mừng không những cho ngành công nghệ thông tin mà cho còn tạo cơ hội cho ngành ngân hàng để phát triển dịch vụ Internet banking. Thực tế ở Việt Nam hiện nay cho thấy, các NHTM đã nhận thức được lợi ích to lớn của Internet banking nên rất quan tâm tới sự phát triển dịch vụ này, các ngân hàng đã và đang chạy đua quyết liệt để chiếm lĩnh thị phần. Internet banking là một dịch vụ được cung ứng khá sớm ở các nước trên thế giới, năm 1980 dịch vụ này được cung ứng bởi một ngân hàng ở Scotland. Tuy nhiên, dịch vụ này chính thức được cung ứng bởi các ngân hàng vào năm 1990, và ngày càng mở rộng và phát triển. Hiện nay, Internet banking đã và đang trở thành xu hướng chính của các ngân hàng tại Mỹ, Châu Âu, Anh, Châu Âu, Châu Á - Thái Bình Dương. - Tại Mỹ, các giao dịch dựa trên Internet banking đã trở nên kinh tế và hiệu quả cho ngành ngân hàng Mỹ trong việc giảm chi phí và cải thiện chất lượng dịch vụ cho các khách hàng hiện tại và tiềm năng của ngân hàng ở phạm vi toàn cầu. Với Internet banking, các NHTM có thể cung cấp cho khách hàng mọi thông tin cần thiết về ngân hàng và có thể thực hiện dễ dàng các chương trình giới thiệu sản phẩm, dịch vụ mới, hoặc chương trình khuyến mãi. + Internet banking giúp các NHTM mở rộng phạm vi hoạt động, tăng khả năng cạnh tranh. Internet banking là một giải pháp của NHTM để nâng cao chất lượng dịch vụ và hiệu quả hoạt động, qua đó nâng cao khả năng cạnh tranh. Hơn nữa, nó còn giúp các NHTM thực hiện chiến lược toàn cầu hóa mà không cần mở chi nhánh ở nước ngoài. - Đối với khách hàng, Internet banking đã mang lại những giá trị mới cho khách hàng, tiết kiệm thời gian, chi phí, tiện lợi thực hiện mọi lúc mọi nơi, nhanh chóng và hiệu quả, theo đó: + Internet banking giúp cho khách hàng có thể liên lạc với ngân hàng một cách nhanh chóng, thuận tiện để thực một số dịch vụ ngân hàng tại bất kỳ thời điểm nào (24 giờ mỗi ngày, 7 ngày trong tuần) và ở bất cứ nơi đâu. Đặc biệt, điều này rất có ý nghĩa đối với các khách hàng có ít thời gian đi đến văn phòng để giao dịch trực tiếp với ngân hàng, đối với các doanh nghiệp nhỏ và vừa, và đối với khách hàng cá nhân có số lượng giao dịch với ngân hàng ít, số tiền mỗi giao dịch không lớn. + Internet banking cho phép khách hàng thực hiện và xác nhận các giao dịch với độ chính xác cao, rất nhanh chỉ trong vài giây. + Chi phí cho các giao dịch trên mạng ít hơn rất nhiều so với giao dịch trực tiếp tại các chi nhánh ngân hàng do khách hàng không phải mất chi phí đi lại cũng như không phải trả phí phục vụ cho ngân hàng. + Khách hàng có thể truy cập và quản lý tất cả các tài khoản ngân hàng chỉ trong một trang web. Phần lớn, các trang web của dịch vụ Internet banking cung cấp cho khách hàng thêm một số dịch vụ khác, chẳng hạn như: báo giá chứng khoán, thông báo lãi suất, quản lý danh mục đầu tư. Ngoài ra, với các tiêu chuẩn đã được chuẩn hóa, khách hàng được phục vụ một cách chính xác thay vì phải tùy thuộc vào thái độ phục vụ khác nhau của các nhân viên ngân hàng (Nguyễn Thị Thanh Thúy, 2008). Với những lợi ích này, dịch vụ Internet banking là xu hướng phát triển tất yếu của các NHTM Việt Nam nói riêng và các ngân hàng trên thế giới nói chung.

* 1. **Giới hạn/ Phạm vi:**

Đối tượng nghiên cứu: Khách hàng có khó khăn với những thủ tục tại ngân hàng

Không gian nghiên cứu: Trường đại học Công Nghệ Thông Tin

Thời gian nghiên cứu: 15/05/2019 – 11/07/2019

* 1. **Các nội dung nghiên cứu và thực hiện:**

Trong các nghiên cứu về các yếu tố trở ngại đối với việc sử dụng Internet banking của khách hàng, có các nghiên cứu sau đây: - Đối với các nghiên cứu về nhận thức của khách hàng đối với việc bảo mật thông tin khi sử dụng Internet banking: + Nguyễn Hoàng Bảo Khánh (2014) đã chỉ ra rằng khách hàng cảm thấy lo sợ và e ngại khi sử dụng dịch vụ Internet banking bởi vì họ sợ tài khoản ngân hàng và thông tin cá nhân bị đánh cấp với hacker. Nguyễn Duy Thanh và Cao Hào Thi (2011) cũng cho rằng, yếu tố rủi ro, bảo mật trong giao dịch là một trong những yếu tố quan trọng khiến khách hàng cân nhắc nên chấp nhận sử dụng Internet banking hay không bởi vì họ sợ thông tin bị mất cấp. + Safeena và các cộng sự (2011) đã nghiên cứu về sự cảm nhận của khách hàng khi sử dụng Internet banking của các ngân hàng ở Ấn Độ, kết quả nghiên cứu này chỉ ra rằng: Khách hàng vẫn còn chưa sẵn lòng sử dụng dịch vụ Internet banking bởi vì lý do bảo mật thông tin. Vì vậy, nghiên cứu này đưa đề nghị ngân hàng cần áp dụng công nghệ cao để bảo mật thông tin cho khách hàng khi họ sử dụng Internet banking. Trong một nghiên cứu về tác động của Internet banking đối với chất lượng dịch vụ của ngân hàng ở Pakistan, Moinuddin (2013) đã chỉ ra rằng: khách hàng vẫn còn đối mặt với tình trạng thiếu việc bảo mật thông tin từ phía ngân hàng. Cụ thể hơn, khách hang thường có tâm lý e ngại tài khoản của mình bị đánh cấp bởi các hacker, vì vậy, họ không dám giao dịch trên mạng internet. Nghiên cứu này đề nghị ngân hàng cần cung cấp chế độ bảo mật tốt nhất để khách hàng có thể an tâm khi sử dụng Internet banking. - Đối với các nghiên cứu về thói quen thích sử dụng tiền mặt trong giao dịch hơn dịch vụ ngân hàng điện tử của khách hàng: + Nông Thị Như Mai (2015) đã cho rằng thói quen sử dụng tiền mặt của khách hàng vẫn chưa thể thay đổi được. Doanh số dùng thẻ để rút tiền mặt qua ATM chiếm tỷ trọng quá cao (hơn 83,2% các giao dịch qua thẻ ATM là rút tiền mặt), doanh số thanh toán qua đơn vị chấp nhận thẻ và dịch vụ Internet banking chiếm tỷ lệ quá ít. NGHIÊN CỨU KHOA HỌC 44 SỐ 10 - THÁNG 02/2016 + Matt Keating (2013), Giám đốc các kênh ngân hàng trực tiếp của Ngân hàng Quốc tế (VIB), ông cho rằng, thói quen sử dụng tiền mặt của khách hàng Việt Nam đã cản trở trong quá trình phát triển dịch vụ internet banking, và ông cũng lý giải tại sao người dùng Việt Nam tại sao thích sử dụng tiền mặt trong giao dịch là vì họ cảm giác an toàn khi cầm, giữ được trong tay. - Đối với các nghiên cứu về sự hỗ trợ dịch vụ Internet banking trực tuyến trong việc tiếp nhận và phản hồi các yêu cầu từ khách hàng: Nimako và các cộng sự (2013) đã nghiên cứu sự thõa mãn của khách hàng đối với dịch vụ chất lượng dịch vụ internet banking của ngành ngân hàng ở quốc gia Ghana. Kết quả nghiên cứu này chỉ ra rằng, khách hàng chưa hài lòng đối với việc tiếp nhận và hồi đáp các yêu cầu từ khách hàng, việc hướng dẫn trực tuyến cho khách hàng chậm chạp, kém. - Đối với các nghiên cứu về vai trò của internet banking, các nghiên cứu sau đây chỉ ra vai trò của internet banking đối với khách hàng và ngân hàng: + Dwumfuo và Dankwah (2013) đã nghiên cứu việc sử dụng internet banking tại các ngân hàng ở Ghana, kết quả nghiên cứu này chỉ ra rằng: internet banking đã mang lại nhiều lợi ích cho ngân hàng, như: nâng cao hiệu quả hoạt động, nâng cao dịch vụ khách hàng, nâng cao sự hài lòng của khách hàng, nâng cao năng lực cạnh tranh cho ngân hàng. + Nguyễn Minh Loan (2014) đã cho rằng, internet banking là kênh giao dịch tài chính-ngân hàng quan trọng, thông qua mạng internet, nó giúp khách hàng làm chủ nguồn tài chính mọi lúc, mọi nơi. Khách hàng của ngân hàng có thể thực hiện được tất cả các giao dịch và trao đổi thông tin với ngân hàng thông qua các thiết bị điện tử (như máy tính, điện thoại di động) có kết nối mạng Internet mà không cần đến các quầy giao dịch của ngân hàng. Dịch vụ internet banking đang đóng một vai trò quan trọng trong hoạt động dịch vụ, góp phần mang lại lợi nhuận, đồng thời tăng hình ảnh, thương hiệu cho các ngân hàng. Tóm lại, các nghiên cứu ở trên đã chỉ ra vai trò quan trọng của Internet banking đối với khách hàng cũng như ngân hàng và cũng nêu lên những trở ngại của việc phát triển Internet banking. Sau đây là bảng tóm tắt kết quả chủ yếu của các nghiên cứu:

|  |  |
| --- | --- |
| Năm | Tóm tắt kết luận sau nghiên cứu |
| 2011-2014 | Khách hàng e ngại việc bảo mật thông tin nên gây trở ngại cho việc phát triển dịch vụ internet banking |
| 2013 | Sự hỗ trợ dịch vụ internet banking trực tuyến (online) trong việc tiếp nhận và phản hồi các yêu cầu từ khách hàng kém, chậm chạp |
| 2013-2014 | Thói quen thích sử dụng tiền mặt của khách hàng nên gây trở ngại cho việc phát triển dịch vụ internet banking |
| 2013-2015 | Vai trò của internet banking đối với khách hàng và ngân hàng là rất quan trọng. |

Bảng tóm tắt nghiên cứu

1. **Công Nghệ Đã Sử Dụng:**
   1. **Frontend:**

Angular là một javascript framework do google phát triển để xây dựng các **Single Page Application** (SPA) bằng JavaScript , HTML và TypeScript . Angular cung cấp các tính năng tích hợp cho animation , http service và có các tính năng như auto-complete , navigation , toolbar , menus ,… Code được viết bằng TypeScript , biên dịch thành JavaScript và hiển thị tương tự trong trình duyệt. Và trong đồ án này em sẽ sử dụng**Angular 8.**

**Angular Module**

Ứng dụng Angular bao gồm một số các khối như Component, Service và Directive. Chúng ta tạo ra nhiều block để phát triển ứng dụng lớn dần. Angular đưa ra một cách tốt để tổ chức các khối này sử dụng một khái niệm gọi là Angular Module.

Chúng ta tạo ra các component, service, và directive sau đó đặt chúng trong Angular Module. Chúng ta sử dụng một directive đặc biệt gọi là @NgModule để tạo module. Angular Module cũng được gọi là NgModule.

Sử dụng Angular Module (hoặc ngModule) để tổ chức code ứng dụng Angular với một ứng dụng Angular lớn hơn. Ứng dụng sẽ gộp lại bởi nhiều Angular Module. Mỗi Module triển khai một tính năng cụ thể voặc các tính năng của ứng dụng.

**Lệnh import**

Lệnh import chỉ ra cho Angular biết tìm các function đó ở đâu và sử dụng trong module của chúng ta. Tất cả các thành phần bên ngoài giống như một thư viện của third party, các module của Angular phải được import. Một lệnh import là một phần của đặc tả ES2015. Nó tương tự như như lệnh import của Java hay lệnh Using của C#.

Ta chỉ có thể import chỉ các member được export từ module khác.

**Class**

Class chứa logic của ứng dụng. Nó có thể chứa method và properties như là C# hoặc Java class. Class phải được định nghĩa với từ khoá export nếu bạn muốn sử dụng class từ module khác.

**Class Decorator**

Nếu không có class decorator, AppComponent chỉ là một class. Không có gì liên quan Angular. Nó là decorator chỉ ra cho Angular biết cách sử dụng class đó.

Ví dụ, @Component decorator chỉ ra cho Angular biết class là một Component. Tương tự thế, một @Directive kể cho Angular biết class đó là Directive. Angular hiện tại có các class decorator:

1. @Component
2. @Directive
3. @Injectable
4. @NgModule
5. @Pipe

**Xây dựng các khối của ứng dụng Angular**

**Nhìn chung trong ứng dụng có thể định danh 7 khối của Angular như sau:**

**1.Component**

**2.Templates**

**3.Metadata**

**4.Data Binding**

**5.Directives**

**6.Services**

**7.Dependency Injection**

**Component**

Angular Component là một class nó được bổ trợ bởi @Component decorator. Component điều khiển một phần của UI. Ứng dụng Task List ở trên có 3 component. TaskComponent hiển thị danh sách task, còn TaskAddComponent giúp chúng ta tạo mới task. RootComponent là component cha của các component này chỉ hiển thị tên của ứng dụng. Component được tạo ra trong ứng dụng mới nhìn sẽ như sau:



Cấu trúc component

Component có 4 phần như sau:

1. Import statement
2. Class
3. Template
4. Metadata

Lệnh import các thành phần liên quan được yêu cầu bởi Component này. Class chứa các logic của ứng dụng. Nó được bổ sung thông tin bởi @Component decorator.

**Template**

Component cần một view để hiển thị. Template định nghĩa view. Template chỉ là một tập con của HTML, nó chỉ cho Angular biết làm sao để hiển thị view. Nó là một trang HTML chuẩn sử dụng các thẻ H1, H2...Nó cũng sử dụng các lệnh của Angular như {}, []...

**Metadata**

Metadata chỉ cho Angular làm sao để xử lý class. Chúng ta thêm Metadata vào class sử dụng class decorator. Khi chúng ta thêm @Component decorator vào class nó sẽ trở thành component.

Class decorator sử dụng đối tượng cấu hình, cung cấp thông tin Angular cần để tạo component. Ví dụ @Component directive đi với cấu hình như selector, templateURL, directive...

**Data Binding**

Angular sử dụng Data Binding để lấy dữ liệu từ Component đến View. Nó được hoàn thành sử dụng cú phpas HTML đặc tả được biết đến là Template Syntax

Angular hỗ trợ 4 kiểu của Data binding:

1. Interpolation: Data được bind ra từ component sang view
2. Property Binding: Data bind ra từ component sang thuộc tính HTML trong view.
3. Event Binding: Các sự kiện DOM được bind ra từ view ra method của Component
4. Two-way binding/Model binding: Luồng dữ liệu cả hai chiều từ view sang component và ngược lại.

**Directive**

Directive giúp chúng ta thao tác với View. Một directive là một class, chúng ta tạo ra sử dụng @Directive nó chứa metadata và logic thao tác trên DOM.

View được tạo ra bởi Angular sử dụng template metadata được định nghĩa trong Component. Các template là động và được chuyển đổi sử dụng directive.

Angular hỗ trợ 2 loại directive là Structural directive sẽ thay đổi cấu trúc của view. Còn loại kia là attribute directive sẽ chỉnh sửa style của view.

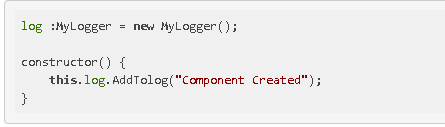
**Services**

Service cung cấp dịch vụ cho các Components hoặc các service khác. Angular không có bất cứ đặc tả nào cho Service nó chỉ là các class có export method và chứa một số task. Bạn không cần làm gì cả



cấu trúc services

Và trong bất cứ component nào, chúng ta chỉ cần gọi nó:



**Gọi services từ component**

Nó là các Javascript module thuần không có gì đặc biệt. Angular đã làm gì để các service này có sẵn cho component? Nó gọi là cơ chế dependecy injection.

**Dependency Injection**

Dependency Injection là một phương thức khi tự động tạo một instance của một service được inject vào một component hoặc một service khác khi nó được yêu cầu.

Denpendency injection được dùng hầu hết khi inject service vào component hoặc service khác.

Angular làm điều này sử dụng injector. Khi một component được tạo, Angular sẽ tìm Component metadata các service mà component này yêu cầu. Injector sẽ tạo ra instance của service đó và inject vào component sử dụng constructor của nó.

Nếu service đã tạo thì injector sẽ không tạo mà sử dụng lại. Service cần chỉ ra cho Angular biết nó sẵn sàng được inject vào bất cứ component nào yêu cầu nó bằng cách sử dụng @Injectable decorator.



Ví dụ Dependency Injection

* 1. **Backend:**
     1. ASP.NET core là gì:

ASP.NET Core là một open-source mới và framework đa nền tảng (cross-platform) cho việc xây dựng những ứng dụng hiện tại dựa trên kết nối đám mây, giống như web apps, IoT và backend cho mobile.

.net core gồm:

1. MVC

2. ASP.NET Core Web API

3. backend mobile (Xamarin)

* + 1. **Lịch sử ra đời:**

Bản phát hành đầu tiên của ASP.NET đã xuất hiện cách đây 15 năm trước, nó là một phần của .NET Framework. Từ đó, hàng triệu lập trình viên đã sử dụng nó để xây dựng những ứng dụng web tuyệt vời, và trên những năm đó Microsoft đã phát triển thêm nhiều tính năng mới.

| **Version** | **Original Release Date** | **Latest Patch Version** | **Patch Release Date** | **End of Support** |
| --- | --- | --- | --- | --- |
| .NET Core 3.1 | December 3, 2019 | 3.1.5 | June 9, 2020 | December 3, 2022 |
| .NET Core 3.0 | September 23, 2019 | 3.0.3 | February 18, 2020 | March 3, 2020 |
| .NET Core 2.2 | December 4, 2018 | 2.2.8 | November 19, 2019 | December 23, 2019 |
| .NET Core 2.1 | May 30, 2018 | 2.1.19 | June 9, 2020 | August 21, 2021 |
| .NET Core 2.0 | August 14, 2017 | 2.0.9 | July 10, 2018 | October 1, 2018 |
| .NET Core 1.1 | November 16, 2016 | 1.1.13 | May 14, 2019 | June 27 2019 |
| .NET Core 1.0 | June 27, 2016 | 1.0.16 | May 14, 2019 | June 27 2019 |

Một vài framework backend được nhiều người sử dụng khác:

- Ruby on Rails

- Spring

- Django

- Laravel

Bảng so sánh ưu và khuyết điểm :

Ưu:

|  |  |  |
| --- | --- | --- |
| .NET core | Spring boot | Rails |
| Hoàn hảo để lập trình backend ( phát triển frontend nhanh ) | Mạnh mẽ và tiện dụng | Phát triển dự án nhanh |
| Đa nền tảng | Đa nền tảng | Cộng đồng lớn |
| Phát triển dự án nhanh | Dễ cài đặt | Dễ cài đặt |
| Cung cấp kiến trúc codebase và tổ chức code tốt hơn | Có hệ sinh thái Spring | Phát triển web tốt |
| Hiệu năng cao | Quản lý cache tốt | Quản lý cache tốt |
|  |  |  |

Nhược:

|  |  |  |
| --- | --- | --- |
| .NET core | Spring boot | Rails |
|  | Bản cài đặt rất nặng | Hiệu năng kém |
|  | Cần quá nhiều file Config | Tốc độ boot lâu |

Tại sao nên sử dụng .net core:

* Có thể phát triển và chạy trên Windows, macOS và Linux.
* Mã nguồn mở và tập trung vào cộng đồng.
* Nhẹ, hiệu năng cao, module hóa HTTP request pipeline
* Hợp nhất việc xây dựng web UI và web APIs
* Mã ít hơn Một vài ví dụ về ít mã hơn bao gồm:

+ [LINQ (Language INtegrated Query](https://msdn.microsoft.com/en-us/library/bb308959.aspx) : loại bỏ RẤT NHIỀU vòng lặp for..next trong mã của bạn để giúp viết vòng lặp một lớp dễ dàng hơn.

+ [Toán tử ternary nội tuyến (?)](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator) : Loại bỏ sự cần thiết của một  câu lệnh *if* đơn giản  cũng làm cho nó trở thành một lớp lót.

* Mã ít hơn => dễ bảo trì hơn

| **App type** | **Scenario** | **Tutorial** |
| --- | --- | --- |
| Web app | New server-side web UI development | [Get started with Razor Pages](https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-3.1) |
| Web app | Maintaining an MVC app | [Get started with MVC](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-3.1) |
| Web app | Client-side web UI development | [Get started with Blazor](https://dotnet.microsoft.com/learn/aspnet/blazor-tutorial/intro) |
| Web API | RESTful HTTP services | [Create a web API](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1)† |
| Remote Procedure Call app | Contract-first services using Protocol Buffers | [Get started with a gRPC service](https://docs.microsoft.com/en-us/aspnet/core/tutorials/grpc/grpc-start?view=aspnetcore-3.1) |
| Real-time app | Bidirectional communication between servers and connected clients | [Get started with SignalR](https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr?view=aspnetcore-3.1) |
| Microservices |  |  |
| Mobile app with Xamairin |  |  |

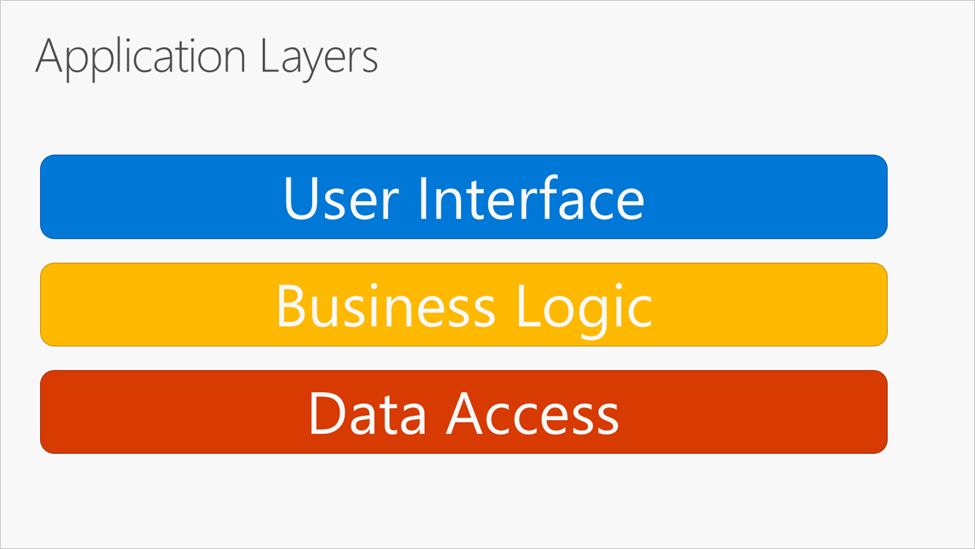
* + 1. **Kiến trúc**:

Kiến trúc asp.net core:

- Trước khi nói đến kiến trúc asp.net core thì chúng tên đọc lại định nghĩa về kiến truc đa lớp (N – layer).

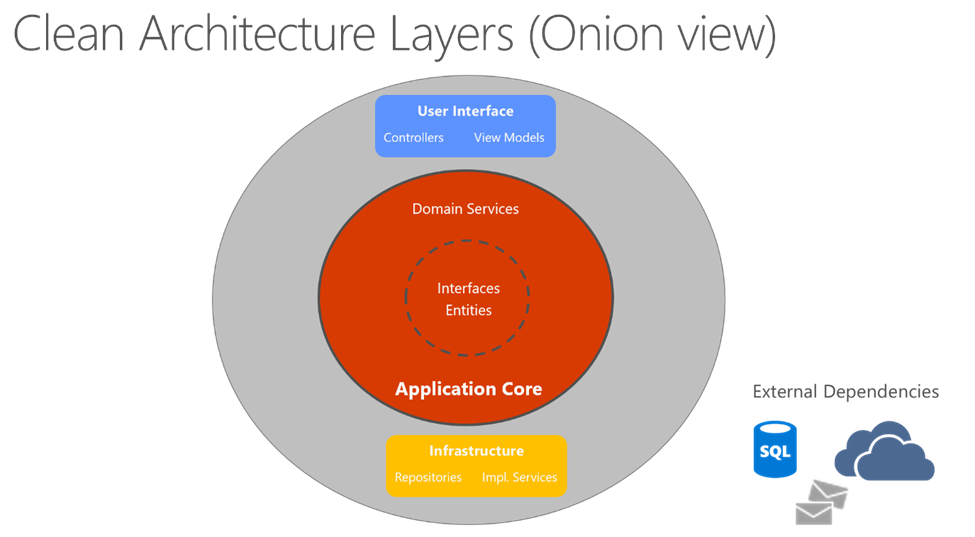
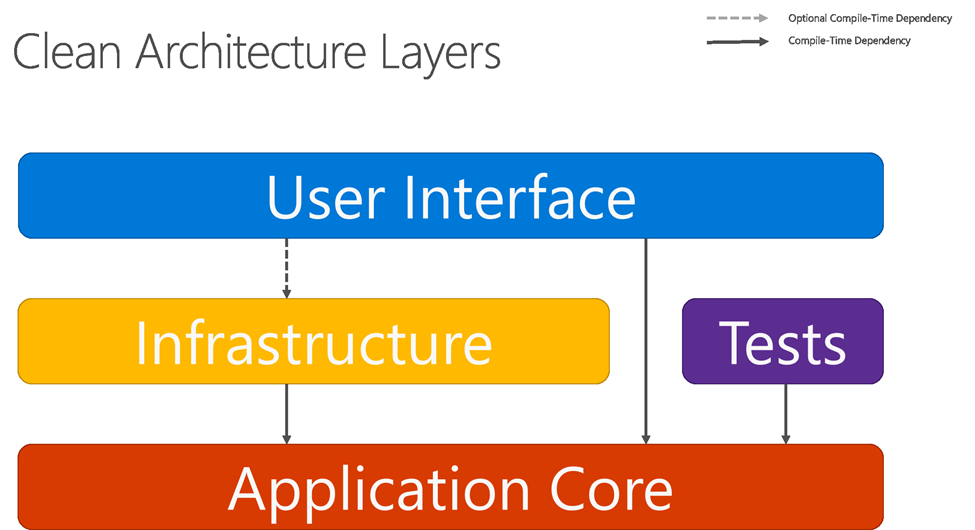
Kiến trúc đa lớp (N – layer) : Khi một ứng dụng bắt đầu trở nên phức tạp, một trong những cách tổ chức quản lý là tách nó thành các project riêng. Tách một project lớn thành nhiều project nhỏ nằm trong cùng một solution có lợi cho tổ chức code. Bạn không còn phải làm việc với những project quá phức tạp với rất nhiều file và thư mục. Mỗi project con sẽ đơn giản gọn gàng hơn.

Lưu ý rằng, layer thể hiện sự phân chia về mặt logic của ứng dụng. Nghĩa là, dù tách thành nhiều layer, mỗi layer biên dịch về một assembly riêng, nhưng khi thực thi, các assembly đều tải vào cùng một tiến trình. Do vậy, loại ứng dụng này vẫn là ứng dụng monolith.



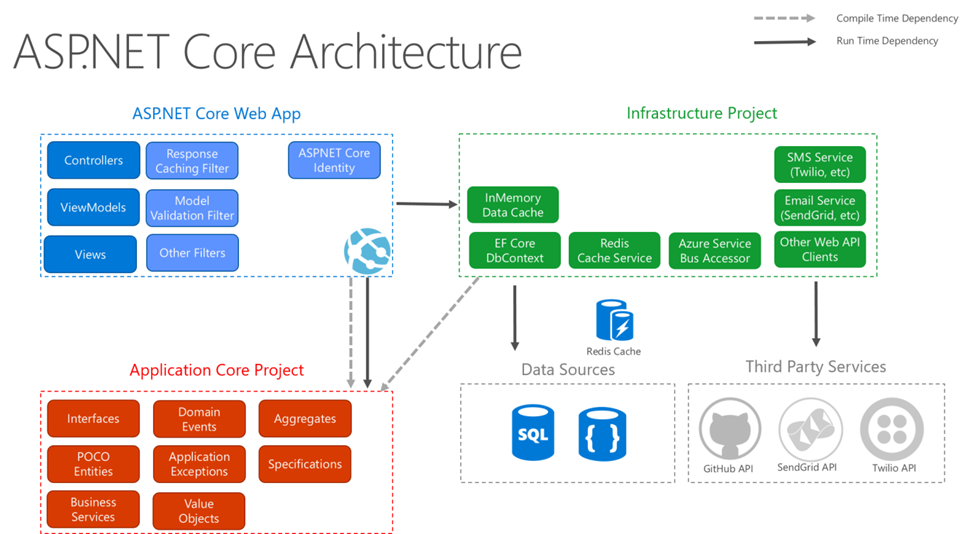
- Trong dự án ASP.NET Core (MVC hoặc Razor Pages) bạn có thể vận dụng một biến thể của nó gọi là clean architecture. Kiến trúc này đôi khi còn được gọi là kiến trúc hành tây (onion architecture). Theo kiến trúc này, bạn chia ứng dụng làm 3 layer: Application Core, Infrastructure, User Interface.

Có hai cách nhìn khác nhau đối với kiến trúc này: onion view và layer view.



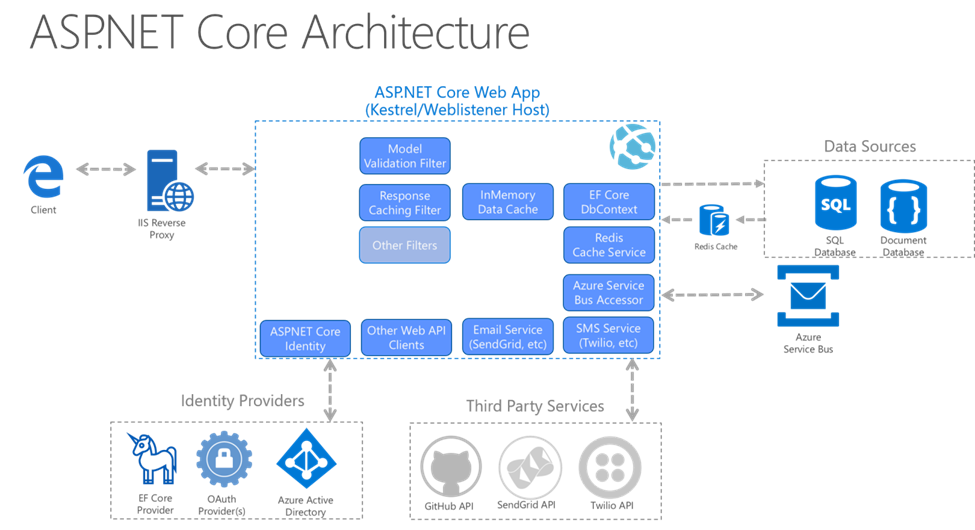
Trong sơ đồ này, các phụ thuộc chảy về vòng tròn trong cùng. Và bạn có thể thấy trên sơ đồ rằng Application Core không có sự phụ thuộc vào các lớp ứng dụng khác. Entites và Interfaces của ứng dụng là trung tâm. Sau đó, nhưng vẫn trong Application Core, là các dịch vụ miền, thường triển khai các giao diện được xác định trong vòng tròn bên trong. Bên ngoài Application Core, cả hai layer UI và Infrastructure đều phụ thuộc vào Application Core, nhưng không phụ thuộc vào nhau.

Hình sau cho thấy một cái nhìn chi tiết hơn về kiến trúc của ứng dụng ASP.NET Core khi được xây dựng theo các khuyến nghị này.



Vì layer UI không có bất kỳ sự phụ thuộc trực tiếp nào vào các loại được xác định trong dự án Infrastructure, do đó, rất dễ dàng để trao đổi các triển khai, để thuận tiện cho việc thử nghiệm hoặc đáp ứng các yêu cầu khi ứng dụng thay đổi. Việc sử dụng và hỗ trợ tích hợp của ASP.NET Core cho DI làm cho kiến trúc này trở thành cách thích hợp nhất để cấu trúc các ứng dụng nguyên khối.

Đối với các ứng dụng nguyên khối, các dự án Application, Infrastructure và UI đều chạy dưới dạng một ứng dụng. Kiến trúc ứng dụng thời gian chạy có thể trông giống như hình sau.



* + 1. **Tổ chức mã trong dự án**:

Trong một giải pháp kiến trúc sạch, mỗi dự án có trách nhiệm rõ ràng. Như vậy, một số loại nhất định thuộc về từng dự án và bạn sẽ thường xuyên tìm thấy các thư mục tương ứng với các loại này trong dự án phù hợp.

Application Core giữ mô hình kinh doanh (business), bao gồm các entity, services và interfaces. Các giao diện này bao gồm các trừu tượng cho các hoạt động sẽ được thực hiện bằng Infrastructure, chẳng hạn như truy cập dữ liệu, truy cập hệ thống tệp, cuộc gọi mạng... Đôi khi các dịch vụ hoặc giao diện được xác định ở layer này sẽ cần phải hoạt động với các loại không phải entity, không phụ thuộc vào UI hoặc Infrastructure. Chúng có thể được định nghĩa là các đối tượng truyền dữ liệu đơn giản (Data Tranfer Objects - DTOs).

Các loại Application Core:

- Entities (các lớp mô hình kinh doanh vẫn tồn tại)

- Interfaces

- Services

- DTOs

Dự án Infrastructure thường bao gồm triển khai truy cập dữ liệu. Trong một ứng dụng web ASP.NET Core điển hình, các triển khai này bao gồm DbContext Entity Framework (EF), bất kỳ đối tượng Migration EF Core nào đã được xác định và các lớp triển khai truy cập dữ liệu. Cách phổ biến nhất để mã triển khai truy cập dữ liệu trừu tượng là thông qua việc sử dụng mẫu thiết kế Respository.

Ngoài việc triển khai truy cập dữ liệu, dự án Infrastructure nên chứa các triển khai dịch vụ phải tương tác với các mối quan tâm về cơ sở hạ tầng. Các dịch vụ này sẽ triển khai các giao diện được xác định trong Application Core và vì vậy Infrastructure nên có một tham chiếu đến dự án Application Core.

Các loại Infrastructure:

- Các loại của EF Core ( DbContext, Migration)

- Các kiểu triển khai truy cập dữ liệu (Respository)

- Các dịch vụ dành riêng cho cơ sở hạ tầng (ví dụ: FileLogger hoặc SmtpNotifier)

Layer UI trong ứng dụng ASP.NET Core MVC là điểm vào cho ứng dụng. Dự án này nên tham chiếu dự án Application Core và các loại của nó sẽ tương tác với Infrastructure một cách nghiêm ngặt thông qua các giao diện được xác định trong Application Core. Không được phép khởi tạo trực tiếp hoặc gọi tĩnh đến các loại lớp Infrastructure trong lớp UI.

Các loại UI

- Controllers

- Filters

- Views

- ViewModels

- Startup

Lớp Startup chịu trách nhiệm cấu hình ứng dụng và kết nối các kiểu triển khai với các giao diện, cho phép DI hoạt động chính xác trong thời gian chạy.

Để kết nối nội dung phụ thuộc vào ConfigureService trong tệp Startup.cs của dự án UI, dự án có thể cần tham chiếu dự án Infrastructure. Sự phụ thuộc này có thể được loại bỏ, dễ dàng nhất bằng cách sử dụng container DI tùy chỉnh. Đối với mục đích của mẫu này, cách tiếp cận đơn giản nhất là cho phép dự án UI tham chiếu dự án Infrastructure.

* + 1. **Tính năng Dependence Injection được hỗ trợ sẵn**:

public class Car

{

Horn horn; // horn là một Dependecy của Car

public Car(Horn horn) => this.horn = horn; // horn được Inject (bơm vào) bằng hàm khởi tạo

public void Beep()

{

horn.Beep();

}

}

* **Inject thông qua phương thức khởi tạo:** cung cấp các Dependency cho đối tượng thông qua hàm khởi tạo

Horn horn = new Horn();

var car = new Car(horn);

car.Beep();

* **Inject thông qua setter:**tức các Dependency như là thuộc tính của lớp, sau đó inject bằng gán thuộc tính cho Depedency object.denpendency = obj;
* **Inject thông qua các Interface** - áp dụng Interface cho các Dependency - nó có thể dùng setter, hàm tạo ...

public class Car

{

IHorn horn; // horn là một Dependecy của Car, triển khai từ Interface IHorn

public Car(IHorn horn) => this.horn = horn; // Inject từ hàm tạo

public void Beep() => horn.Beep();

}

public class HeavyHorn : IHorn

{

public void Beep() => Console.WriteLine("(kêu to lắm) BEEP BEEP BEEP ...");

}

public class LightHorn : IHorn

{

public void Beep() => Console.WriteLine("(kêu bé lắm) beep bep bep ...");

}

Car car1 = new Car(new HeavyHorn());

car1.Beep(); // (kểu to lắm) BEEP BEEP BEEP ...

Car car2 = new Car(new LightHorn());

car2.Beep(); // (kểu bé lắm) beep bep bep ...

* + 1. Quản lý vòng đời (lifetime) Service:

Bất cứ khi nào chúng ta yêu cầu service, DI Container sẽ quyết định xem có tạo mới một thể hiện (instance) hay sử dụng lại thể hiện đã tạo từ trước đó. Vòng đời của Service phụ thuộc vào khi khởi tạo thể hiện và nó tồn tại bao lâu. Chúng ta định nghĩa vòng đời khi đăng ký service.

Chúng ta đã học cách sử dụng DI ở bài trước. Có 3 mức độ vòng đời, bằng cách này chúng ta có thể quyết định mỗi service có vòng đời ra sao.

1. Transient: Một thể hiện mới luôn được tạo, mỗi khi được yêu cầu.

2. Scoped: Tạo một thể hiện mới cho tất cả các scope (Mỗi request là một scope). Trong scope thì service được dùng lại

3. Singleton: Service được tạo chỉ một lần duy nhất.

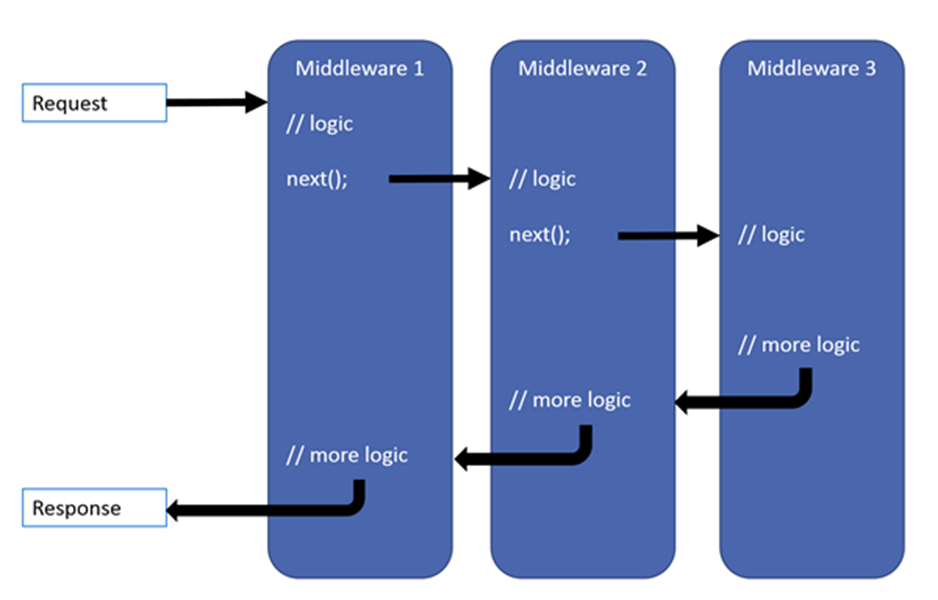
* + 1. **MIDDLEWARE**:

**Request Pipeline**

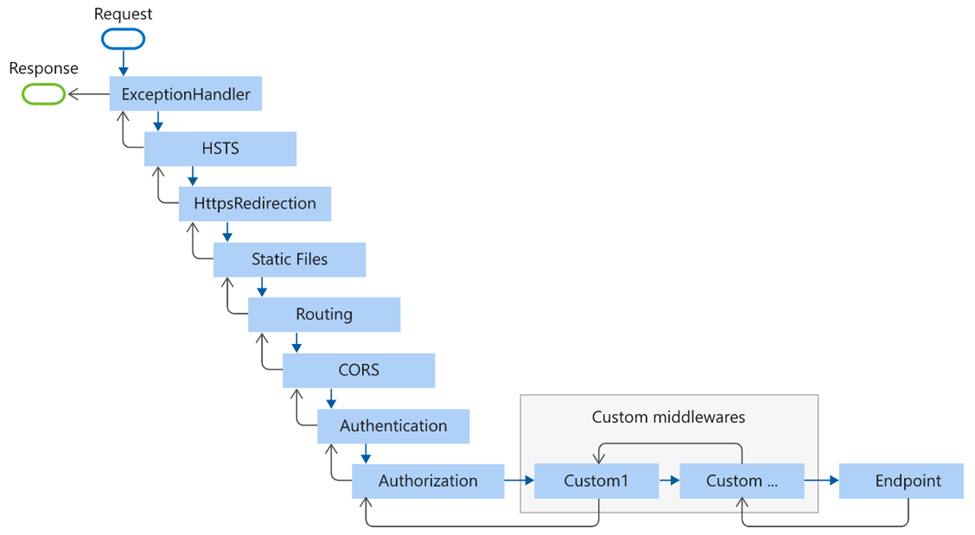
Request Pipeline là cơ chế bắt đầu khi các request bắt đầu được xử lý với một đối tượng Request đầu vào và kết thúc với đầu ra là một response. Pipeline chỉ ra cách mà ứng dụng phản hồi với HTTP Request. Request đến từ trình duyệt đi qua pipeline và quay trở lại khi xử lý xong để trả về client. Các thành phần đơn lẻ tạo nên pipeline này được gọi là middleware.

**Middleware là gì?**

Middleware là thành phần của phần mềm đóng vai trò tác động vào request pipeline (luồng request) để xử lý chúng và tạo ra response phản hồi lại client. Mỗi một tiến trình middleware thao tác với các request nhận được từ middleware trước nó. Nó cũng có thể quyết định gọi middleware tiếp theo trong pipeline hoặc trả về response cho middleware ngay trước nó. (ngắt pipeline).



Thứ tự của middleware trong pipeline của asp.net core và Razor page app



trình tự các middleware thực thi khi một request gửi đến

Ở trên là trình tự các middleware thực thi khi một request gửi đến. Một số thì không cần theo trình tự trên nhưng một số lại bắt buộc phải tuân theo như:

- UseCors, UseAuthentication, và UseAuthorization phải theo trình tự trên

Một số build-in middleware được sử dụng với một số mục đích cũng như trong những kịch bản sau:

1. Exception/error handling:

- Khi ứng dụng chạy trong môi trường phát triển: Developer Exception Page Middleware (UseDeveloperExceptionPage) báo lỗi app runtime. Database Error Page Middleware báo lỗi database runtime.

- Khi ứng dụng chạy trong môi trường Product: Middleception Handler Middleware (UseExceptionHandler) bắt các ngoại lệ được ném trong các middleware sau. HTTP Strict Transport Security Protocol (HSTS) Middleware (UseHsts) thêm Strict-Transport-Security vào header.

2. HTTPS Redirection Middleware (UseHttpsRedirection) chuyển hướng HTTP requests đến HTTPS.

3. Cookie Policy Middleware (UseCookiePolicy) khiến ứng dụng tuân tuân thủ ứng dụng theo quy định của Quy định bảo vệ dữ liệu chung (GDPR) của EU.

4. Routing Middleware (UseRouting) đến các route request.

5. Authentication Middleware (UseAuthentication) xác thực người dùng trước khi họ được phép truy cập vào tài nguyên được bảo mật.

6. Authorization Middleware (UseAuthorization) ủy quyền cho người dùng truy cập các tài nguyên được bảo mật

7. Session Middleware (UseSession) thiết lập và duy trì trạng thái session

8. Endpoint Routing Middleware (UseEndpoints with MapRazorPages) thêm Razor page endpoint vào pipelines.

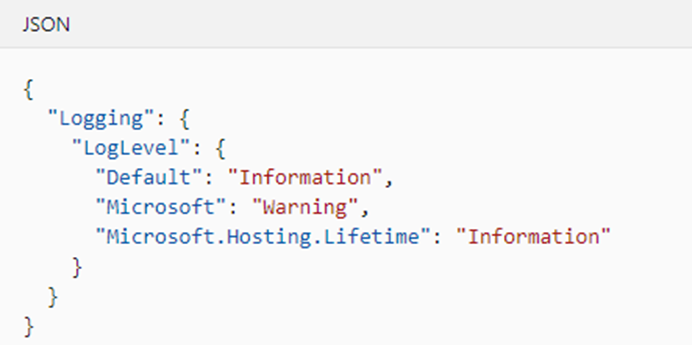
* + 1. **Hệ Thống Log**:

ASP.NET Core khá linh hoạt, có thể tùy chỉnh và có thể mở rộng. Bạn có thể thay đổi hầu hết mọi thứ. Ngay cả việc đăng nhập. Nếu bạn không thích đăng nhập tích hợp, bạn có thể cắm vào trình ghi nhật ký của riêng bạn hoặc trình ghi nhật ký hiện có như log4net, NLog, Elmah.

Dưới đây là các cấp độ của log:

| **LogLevel** | **Value** | **Method** | **Description** |
| --- | --- | --- | --- |
| [Trace](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 0 | [LogTrace](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.logtrace) | Contain the most detailed messages. These messages may contain sensitive app data. These messages are disabled by default and should ***not*** be enabled in production. |
| [Debug](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 1 | [LogDebug](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.logdebug) | For debugging and development. Use with caution in production due to the high volume. |
| [Information](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 2 | [LogInformation](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.loginformation) | Tracks the general flow of the app. May have long-term value. |
| [Warning](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 3 | [LogWarning](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.logwarning) | For abnormal or unexpected events. Typically includes errors or conditions that don't cause the app to fail. |
| [Error](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 4 | [LogError](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.logerror) | For errors and exceptions that cannot be handled. These messages indicate a failure in the current operation or request, not an app-wide failure. |
| [Critical](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 5 | [LogCritical](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loggerextensions.logcritical) | For failures that require immediate attention. Examples: data loss scenarios, out of disk space. |
| [None](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.loglevel) | 6 |  | Specifies that a logging category should not write any messages. |

Trong file appsetting.json thì cấu hình log được tạo sẵn khi khởi tạo dự án:



"Default", "Microsoft", và "Microsoft.Hosting.Lifetime" là các loại được cấu hình log. Trong đó Default mặc định loglevel đều log mọi thứ với mức độ information hoặc hơn. “Microsoft” đại diện cho tất cả cái loại bắt đầu bằng Microsoft, vd: “Microsoft.AspNetCore.Routing.EndpointMiddleware” với mức Warning hoặc cao hơn. "Microsoft.Hosting.Lifetime" là loại riêng biệt hơn, nó chỉ rõ loại và được cấu hình loglevel Information.

Chúng ta có thể cấu hình lại loglevel như hình dưới:



Chỉnh sửa thông tin trong Logging.{providername}.LogLevel sẽ đè lên thông tin trong Logging.LogLevel. Ví dụ: Thông tin cấu hình trong Debug.LogLevel.Default sẽ đè lên thông tin loglevel trong LogLevel.Default.

Một số bí danh provider được xây dựng sẵn mà ta có thể tùy chỉnh như:

• Console

• Debug

• EventSource

• EventLog

• AzureAppServicesFile

• AzureAppServicesBlob

• ApplicationInsights

Chúng ta còn có thể custom log với các cách như:

- Chỉnh sửa màu của từng loglevel

- Chỉnh sửa template của log’s output

Ngoài ra chúng ta còn thể sử dụng có một số logging provider được cộng đồng phát triển:

• elmah.io (GitHub repo)

• Gelf (GitHub repo)

• JSNLog (GitHub repo)

• KissLog.net (GitHub repo)

• Log4Net (GitHub repo)

• Loggr (GitHub repo)

• NLog (GitHub repo)

• PLogger (GitHub repo)

• Sentry (GitHub repo)

• Serilog (GitHub repo)

• Stackdriver (Github repo)

Điểm mạnh hơn so với spring boot về loggin:

- Có thể tùy chỉnh loglevel của từng loại riêng biệt

- Chỉnh màu của từng loglevel giúp nhìn dễ dàng hơn

- Có thể dùng third-party provider được cộng đồng phát triển

* + 1. **WebAPP với Razor:**

Razor Pages là một trong các framework dành cho phát triển ứng dụng web xây dựng trên ASP.NET Core. Razor Pages thể hiện cách tiếp cận và mô hình lập trình riêng trong phát triển ứng dụng web: lấy trang làm trung tâm (page centric), vận dụng mô hình MVVM trong web.

Razor Pages là một trong các framework dành cho xây dựng ứng dụng web bên trên ASP.NET Core. Razor Pages cho phép trộn HTML và C# (gọi là cú pháp Razor) vào cùng một file (có đuôi cshtml) để dễ dàng tạo ra HTML theo logic của chương trình. Nhờ vậy dữ liệu HTML tạo ra “động” chứ không cố định như ở các trang web “tĩnh” thiết kế sẵn.

@page

@{

Layout = null;

ViewData["Title"] = "About";

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>@ViewData["Title"] - Tự học ICT</title>

</style>

</head>

<body>

<h1 class="header">Razor Pages tutorial</h1>

<hr />

<h2>@SayHello("Covid")</h2>

<p>It's @DateTime.Now.ToLongDateString()</p>

<p>ASP.NET Core Razor Pages is a page-focused framework for building dynamic, data-driven web sites with clean separation of concerns. Based on the latest version of ASP.NET from Microsoft - ASP.NET Core, Razor Pages supports cross platform development and can be deployed to Windows, Unix and Mac operating systems.</p>

<hr />

<h2 class="footer">©tuhocict.com - @DateTime.Now.Year</h2>

</body>

</html>

@{

string SayHello(string name) {

return $"Hello, {name}. Welcome to Razor Pages!";

}

}

(1) Trang nội dung bắt buộc phải có @page ở dòng đầu tiên. Thiếu dòng này page sẽ không hoạt động. @page được gọi là directive.

(2) Bạn cũng có thể thấy, trong trang nội dung trộn lẫn code C# và markup của HTML. Trong đó, code C# được đặt bên trong khối @{ }. Ngoài khối này đều là mã HTML. Loại cú pháp sử dụng trong file này gọi là cú pháp Razor

* + 1. **WebAPP với MVC:**

ASP.NET Core MVC là một framework “nhẹ”, opensource, giúp tối ưu hóa hiệu năng của ứng dụng với ASP.NET Core .

ASP.NET Core MVC cung cấp các tính năng dựa trên mô hình xây dựng website động cho phép phân chia rõ ràng các khối lệnh. Nó cung cấp cho bạn toàn quyền kiểm soát đánh dấu, hỗ trợ phát triển với TDD-friendly và sử dụng các tiêu chuẩn web mới nhất.

Tính năng của ASP.NET Core MVC bao gồm:

• Routing

• Model binding

• Model validation

• Dependency injection

• Filters

• Areas

• Web APIs

• Testability

• Razor view engine

• Strongly typed views

• Tag Helpers

• View Components

* + 1. **WebAPP với Blazor:**

Blazor là một framework miễn phí mã mở và là một bộ công cụ mới của Microsoft giúp phát triển client web (chạy trên trình duyệt) sử dụng ngôn ngữ C# (thay cho JavaScript truyền thống). Bản release chính thức của Blazor mới phát hành vào tháng 9/2019 cùng với Asp.net Core 3.0.

Blazor cung cấp khả năng giống như của các JavaScript framework thông dụng như Angular hay React.

Không chỉ vậy, Blazor thậm chí còn cung cấp khả năng tương tác với JavaScript. C# code có thể gọi JavaScript code để tận dụng các thư viện JavaScript khổng lồ!

Blazor được phát triển làm một bộ phận của Asp.net Core, cùng hoạt động song song với MVC và Razor page đã có từ trước. Do vậy, những kinh nghiệm và kỹ thuật đã biết với nền tảng này có thể tiếp tục được tận dụng với Blazor. Blazor cũng có khả năng tái sử dụng code và thư viện như các ứng dụng .NET.

Blazor có hai mô hình hoạt động:

(1) chạy trên server, gọi là Blazor Server app: Blazor tách riêng việc update rendering logic với UI update. Blazor hỗ trợ hosting razor component trên server trong .net core app còn UI update được giao cho kết nối với SignalR.

(2) chạy trên WebAssembly, gọi là Blazor Assembly: Là một single-page APP framework được xây dựng phía client -side để tương tác với .net

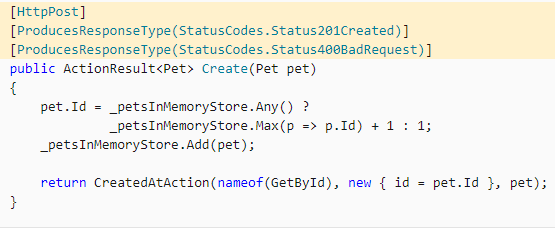
Tương tự như blazor, .net core cũng hỗ trợ cho các SPA framework khác như Angular, React, React with Redux

* + 1. **WebAPI app:**

Ngoài việc hỗ trợ tạo webapp, .net core còn hỗ trợ tạo RESTful services hay còn gọi là APIs. Để xử lý các request thì .net core sử dụng controller là những lớp kế thừa từ class có sẵn tên là Controllerbase.



Bằng cách thêm vào các thuộc tính cho nó ta có thể cấu hình lại hành vi của nó. Hình dưới dây minh họa việc sử dụng thuộc tính để cấu hình HTTP action được nhận, và HTTP status code trả về



Ngoài ra nó còn có thể binding các dữ liệu từ các dạng khác nhau thông qua tham số truyền vào như sau:

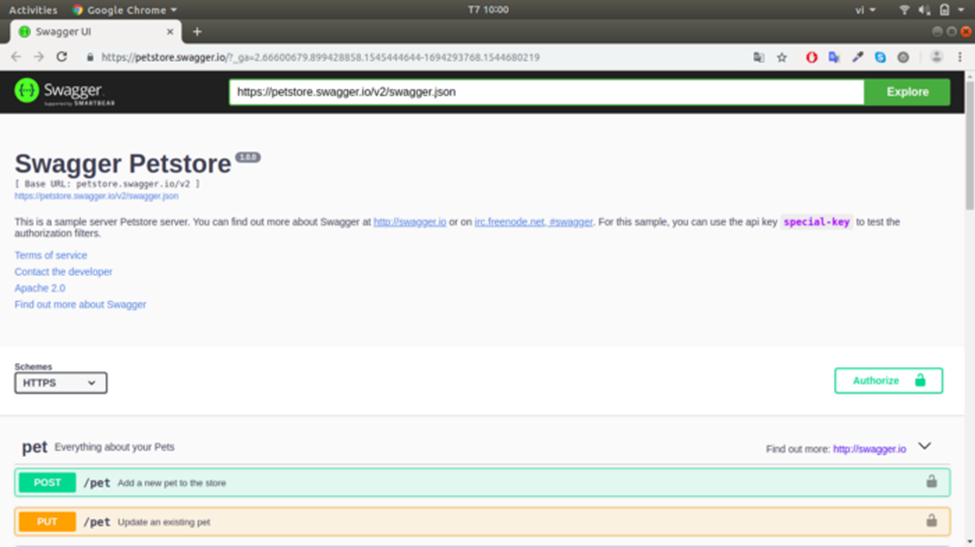
| **Attribute** | **Binding source** |
| --- | --- |
| [[FromBody]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.frombodyattribute) | Request body |
| [[FromForm]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.fromformattribute) | Form data in the request body |
| [[FromHeader]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.fromheaderattribute) | Request header |
| [[FromQuery]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.fromqueryattribute) | Request query string parameter |
| [[FromRoute]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.fromrouteattribute) | Route data from the current request |
| [[FromServices]](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection?view=aspnetcore-3.1#action-injection-with-fromservices) | The request service injected as an action parameter |

* + 1. **Quản lý API với Swagger:**

Swagger là 1 open source dùng để phát triển, thiết kế, xây dựng và làm tài liệu cho các hệ thống RESTfull Web Service

Swagger cung cấp những công cụ hỗ trợ việc tạo doc: Swagger UI, Swagger Editor, Swagger Codegen, Swagger Hub, Swagger Inspector. Trong đó 3 công cụ đầu tiên là open source, Swagger Hub và swagger Inspector là những công cụ cao cấp hơn nhưng sẽ phải trả phí, tuy nhiên chúng ta có thể dùng free trong vòng 30 ngày. Vậy để cho thuận tiện, chúng ta sẽ tìm hiểu các viết doc APIs bằng SwaggerUI và sơ lược về Swagger Hub.

Swagger UI là 1 công cụ giúp gen 1 trang html css mô tả về các APIs được cấu hình bởi 1 file .yaml. Ngoài ra, công cụ này còn cho phép ta mockup đến api đó để xem kết quả

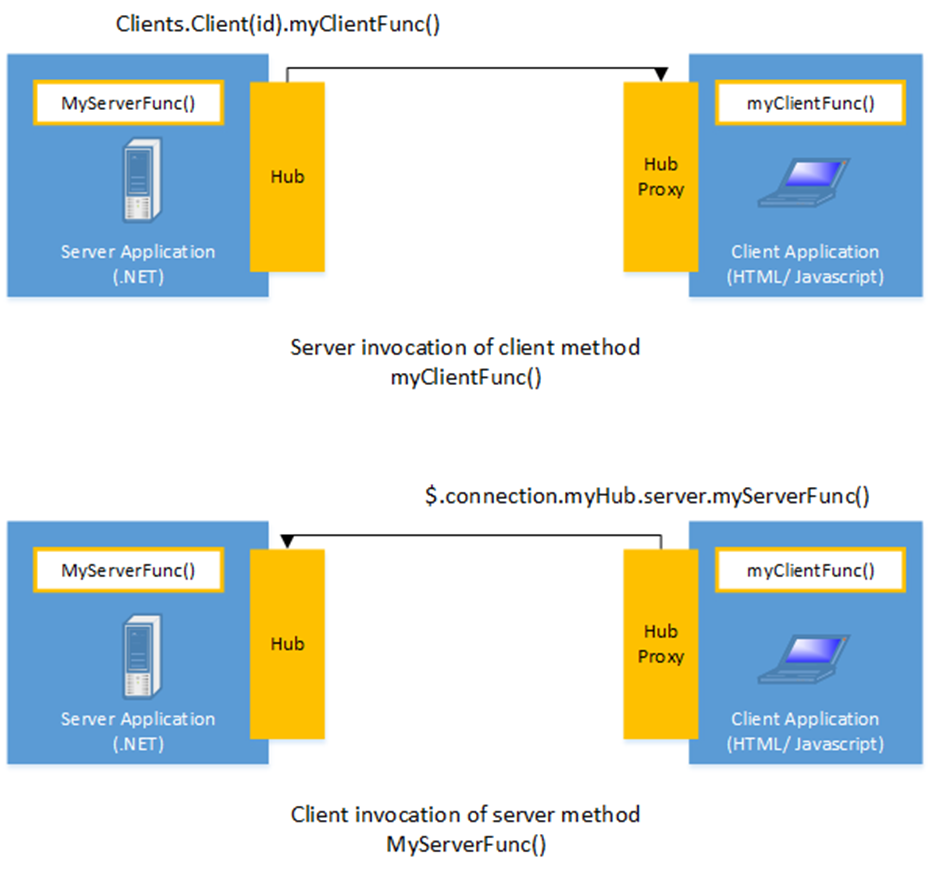


* + 1. **RealTime APP:**

ASP.NET SignalR là một thư viện cho các lập trình viên Asp.Net đơn giản hóa quá trình thêm chức năng web real-time trong phát triển ứng dụng. Real-time web functionality là gì ? Đó là khả năng server đẩy những nội dung tới client đã được kết nối một cách tức thì. Nó khác với giao thức HTTP thông thường: server đợi những yêu cầu từ client và trả về nội dung tương ứng.

SignalR có thể sử dụng trong bất kì chức năng web real-time nào. Trong đó ứng dụng chat trên web là một ví dụ điển hình. Ngoài ra, các ứng dụng cho dashboards, monitoring, collaborative là những gợi ý cho việc sử dụng SignalR.

SignalR cung cấp một API đơn giản cho việc tạo server-to-client remote procedure call (RPC) để gọi những hàm javascript trong trình duyệt (và những nền tảng khác) từ code .Net của server-side. SignalR cũng bao gồm API cho việc quản lý kết nối (connect và disconnect events) và những kết nối nhóm.



SignalR xử lý quản lý kết nối một cách tự động, và cho bạn truyền đi thông điệp tới tất cả các client đã được kết nối một cách đồng loạt, giống như một chat room. Bạn cũng có thể gửi những thông điệp tới những client được xác định. Kết nối giữa client và server là liên tục, không giống như kết nối HTTP cổ điển, cái mà sẽ thành lập lại kết nối cho mỗi lần giao tiếp.

Hiện nó hỗ trợ hầu như mọi nền tảng web như Javascript client(Chorme, safari,..), .NET client(mọi nền tảng .NET core hỗ trợ như Xamarin).

Connection và hub: SignalR API chứa hai chế độ cho việc giao tiếp giữa client và server: Persistent Connection và Hubs.

Persistent Connection: Một kết nối đại diện một endpoint đơn giản cho việc gửi single-recipient, grouped, hoặc broadcast messages. Persistent Connection API (được biểu diễn trong .Net code bởi PersistentConnection class) đưa lập trình viên truy cập trực tiếp tới low-level của giao thức giao tiếp mà SignalR trình bày ra. Việc sử dụng chế độ giao tiếp kết nối sẽ là quen thuộc với những lập trình viên mà đã sử dụng connection-based APIs giống như WCF.

Một Hub là một high-level đã xây dựng dựa trên Connection API mà cho phép client và server gọi những methods của nhau một cách trực tiếp. SignalR xử lý việc điều phối qua biên giới máy như ảo thuật, cho phép clients gọi các methods trên server một cách dễ dàng như các methods cục bộ và ngược lại. Việc sử dụng chế độ giao tiếp Hubs sẽ là quen thuộc với lập trình viên mà đã sử dụng APIs triệu gọi từ xa giống như .Net Remoting. Sử dụng Hub cũng cho phép bạn truyền "strongly typed parameters" tới methods, enabling model binding.

Sercurity: Sử dụng các phương thức như cookie, bearer token, window authentication để bảo mật và cũng hỗ trợ lập trình viên phân quyền.



* + 1. **gPRC:**

[**gRPC**](https://grpc.io/docs/guides/) (trang chủ là <https://grpc.io/>) là một framework ngôn ngữ bất khả tri, framework Remote Procedure Call (RPC) hiệu suất cao.

Những lợi ích chính của gRPC là:

- Framework RPC hiện đại, nhẹ, hiệu suất cao.

- Phát triển API theo hợp đồng, sử dụng Protocol Buffers (**Protobuf**) theo mặc định, cho phép triển khai ngôn ngữ bất khả tri.

- Công cụ có sẵn cho nhiều ngôn ngữ để tạo servers và clients được gõ mạnh (strongly-typed).

- Hỗ trợ client, server và các cuộc gọi stream hai chiều.

- Giảm sử dụng mạng với serialization Protobuf nhị phân.

Những lợi ích này làm cho gRPC lý tưởng cho:

- Microservices nhẹ và cần hiệu suất cao.

- Hệ thống polyglot nơi cần nhiều ngôn ngữ để phát triển.

- Các service thời gian thực point-to-point cần xử lý các stream request hoặc response.

Mặc dù triển khai C# hiện có sẵn trên trang chủ gRPC, nhưng triển khai hiện tại phụ thuộc vào thư viện gốc được viết bằng C (gRPC C-core). Họ hiện đang được tiến hành các công việc để cung cấp một triển khai mới dựa trên máy chủ Kestrel HTTP và ngăn xếp ASP.NET Core được quản lý hoàn toàn.

**So sánh gRPC services với HTTP APIs**

Công nghệ được sử dụng để cung cấp API cho ứng dụng là một lựa chọn quan trọng và gRPC cung cấp các lợi ích độc đáo so với HTTP APIs (bao gồm ASP.NET Core Web APIs). Phần này giới thiệu về những điểm mạnh và điểm yếu của gRPC và đề xuất các kịch bản sử dụng gRPC so với các công nghệ khác.

| **Feature** | **gRPC** | **HTTP APIs with JSON** |
| --- | --- | --- |
| Contract | Required (*.proto*) | Optional (OpenAPI) |
| Protocol | HTTP/2 | HTTP |
| Payload | [Protobuf (small, binary)](https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1#performance) | JSON (large, human readable) |
| Prescriptiveness | [Strict specification](https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1#strict-specification) | Loose. Any HTTP is valid. |
| Streaming | [Client, server, bi-directional](https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1#streaming) | Client, server |
| Browser support | [No (requires grpc-web)](https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1#limited-browser-support) | Yes |
| Security | Transport (TLS) | Transport (TLS) |
| Client code-generation | [Yes](https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.1#code-generation) | OpenAPI + third-party tooling |

**Điểm mạnh của gRPC**

**Hiệu suất cao**:

- Tin nhắn gRPC được tuần tự hóa bằng [Protobuf](https://developers.google.com/protocol-buffers/docs/overview), một định dạng tin nhắn nhị phân hiệu quả. Protobuf tuần tự hóa rất nhanh trên máy chủ và máy khách. Kết quả tuần tự hóa Protobuf dẫn đến tải trọng tin nhắn nhỏ, quan trọng trong các tình huống băng thông hạn chế như ứng dụng di động.

**- gRPC** được thiết kế cho HTTP/2, một phiên bản chính của HTTP cung cấp các lợi ích hiệu suất đáng kể so với HTTP 1.x: Truyền dữ liệu nhị phân được nén. Giao thức HTTP/2 nhỏ gọn và hiệu quả cả trong việc gửi và nhận. Ghép kênh (Multiplexing) của nhiều cuộc gọi HTTP/2 qua một kết nối TCP. Ghép kênh giúp loại bỏ [**head-of-line blocking**](https://en.wikipedia.org/wiki/Head-of-line_blocking).

**Tạo mã tự động:**

- Tất cả các framework gRPC cung cấp hỗ trợ để tạo mã hạng nhất. Một tệp cốt lõi để phát triển gRPC là tệp [**\*.proto**](https://developers.google.com/protocol-buffers/docs/proto3), định nghĩa hợp đồng của các service và tin nhắn gRPC. Từ tệp này, các framework gRPC sẽ mã tạo ra một lớp service cơ sở, các tin nhắn và một máy khách hoàn chỉnh.

- Bằng cách chia sẻ tệp **\*.proto** giữa máy chủ và máy khách, tin nhắn và mã máy khách có thể được tạo từ đầu đến cuối. Tạo mã của máy khách giúp loại bỏ trùng lặp các tin nhắn trên máy khách và máy chủ và tạo ra một máy khách được gõ mạnh cho bạn. Không phải viết một client giúp tiết kiệm đáng kể thời gian phát triển trong các ứng dụng có nhiều dịch vụ.

**Đặc điểm kỹ thuật nghiêm ngặt:**

- Một đặc tả chính thức cho HTTP API với JSON không tồn tại. Các nhà phát triển tranh luận về định dạng tốt nhất của URL, động từ HTTP và mã phản hồi.

- Đặc [tả gRPC](https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md) là quy định về định dạng mà dịch vụ gRPC phải tuân theo. gRPC loại bỏ tranh luận và tiết kiệm thời gian của nhà phát triển vì gPRC nhất quán trên các nền tảng và triển khai.

**Streaming:**

- HTTP/2 cung cấp nền tảng cho các luồng giao tiếp thời gian thực, tồn tại lâu dài. gRPC cung cấp hỗ trợ để stream qua HTTP/2 hạng nhất.

- Dịch vụ gRPC hỗ trợ tất cả các stream kết hợp:

**Unary (không stream)**

**Stream từ máy chủ đến máy khách**

**Stream từ máy khách đến máy chủ**

**Stream hai chiều**

**Deadline/timeouts and cancellation:**

**-**gRPC cho phép client chỉ định thời gian họ sẵn sàng chờ đợi RPC hoàn thành. Deadline được gửi đến máy chủ và máy chủ có thể quyết định hành động nào cần thực hiện nếu vượt quá thời hạn. Ví dụ: máy chủ có thể hủy các yêu cầu gRPC/HTTP/database đang thực hiện khi hết thời gian.

Dealine và cancellation thông qua các cuộc gọi gRPC con giúp thực thi các giới hạn sử dụng tài nguyên.

**Điểm yếu:**

- **Hỗ trợ trình duyệt hạn chế**. Hiện nay không thể gọi trực tiếp dịch vụ gRPC từ trình duyệt. gRPC sử dụng nhiều tính năng HTTP/2 và không có trình duyệt nào cung cấp mức độ kiểm soát cần thiết đối với các request web để hỗ trợ máy khách gRPC. Ví dụ: các trình duyệt không cho phép người gọi yêu cầu sử dụng HTTP/2 hoặc cung cấp quyền truy cập vào các frames HTTP/2 bên dưới.

[**gRPC-Web**](https://grpc.io/docs/tutorials/basic/web.html) là một công nghệ bổ sung từ nhóm gRPC để khắc phục hạn chế của gRPC trong trình duyệt. gRPC-Web bao gồm hai phần: máy khách JavaScript hỗ trợ tất cả các trình duyệt hiện đại và proxy gRPC-Web trên máy chủ. Máy khách gRPC-Web gọi proxy và proxy sẽ chuyển tiếp các request gRPC đến máy chủ gRPC.

Không phải tất cả các tính năng của gRPC đều được gRPC-Web hỗ trợ. Stream từ client và hai chiều không được hỗ trợ và có hỗ trợ cho stream từ máy chủ, nhưng còn hạn chế.

- **Không thể đọc được.** Các yêu cầu HTTP APIs được gửi dưới dạng văn bản và có thể được đọc và tạo bởi con người. Các tin nhắn gRPC được mã hóa theo Protobuf theo mặc định. Mặc dù Protobuf có hiệu quả để gửi và nhận, định dạng nhị phân của nó không thể đọc được. Protobuf yêu cầu mô tả giao diện của tin nhắn được chỉ định trong tệp **\*.proto** để deserialize chính xác. Công cụ bổ sung là cần thiết để phân tích tải trọng Protobuf và để tạo request thủ công.

* + 1. **Testing trong ASP.NET core:**

**Unit test là gì?**

Unit test đơn giản giúp chúng ta kiểm tra tính đúng đắn của một đơn vị code (hầu hết là các method) có làm việc như mong muốn không? Nó cho phép bạn kiểm tra các phương thức độc lập và kiểm tra các logic với các điều kiện dữ liệu khác nhau.

.net core hỗ trợ tạo ra các bộ unit test,intergration test, end-to-end cho các phần sau:

- Razor pages

- Controller

- Middleware

Ngoài hỗ trợ automatic testing như trên thì asp.net core còn có thể thực hiện load and stress testing nhờ vào các phần mềm bên thứ 3.

**Load testing** là một quá trình thêm nhu cầu vào một hệ thống hoặc thiết bị và đo lường phản ứng của nó. Load testing được thực hiện để xác định ứng xử của hệ thống trong các điều kiện tải bình thường và cao hơn điều kiện tải dự kiến. Nó giúp xác định công suất vận hành tối đa của một ứng dụng như các điểm “thắt cổ chai” (bottleneck) và xác định phần tử nào là nguyên nhân gây ra điều đó. Load test được thực hiện vào cuối vòng đời phát triển phần mềm.

**Stress test** được sử dụng để kiểm tra sự ổn định và độ tin cậy của hệ thống. Kiểm thử này chủ yếu xác định tính mạnh mẽ và việc xử lý lỗi hệ thống trong điều kiện tải cực kỳ nặng. Nó thậm chí test vượt ra ngoài mức hoạt động bình thường và đánh giá làm thế nào hệ thống hoạt động trong những điều kiện khắc nghiệt. Stress test được thực hiện để đảm bảo rằng hệ thống sẽ không bị rơi vào tình huống khủng hoảng.

* + 1. **DATA ACCESS:**

Có thể dùng EF core hoặc EF

EF hay EF Core là framework (thư viện khung) để ánh xạ các đơn vị dữ liệu mô tả bằng lớp (đối tượng) vào cơ sở dữ liệu quan hệ, nó cho phép ánh xạ vào các bảng CSDL, tạo CSDL, truy vấn với LINQ, tạo và cập nhật vào database.

Điểm tốt hơn của .net core so với spring boot:

ORM:

Spring boot:

package stackjava.com.springhibernate.entities;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "customer")

public class Customer {

@Id

@Column(name = "id")

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name")

private String name;

@Column(name = "address")

private String address;

public Customer() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

.net core:

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

public class Order

{

public int CustomerId { get; set; }

public string name { get; set; }

public Address Items { get; set; }

}

Khai báo entities của Spring boot mất nhiều thời gian và code hơn của .net core

* + 1. **Host và Deploy:**

Ứng dụng Asp.net core là một console app phải chạy khi server khởi động và khởi động lại nếu bị crash, để tự động hóa chuyện này chúng ta có những process manager phổ biến cho từng hệ điều hành như:

- Linux : nginx, apache

- Window: IIS , window services

Thiết lập Reverse proxy: nếu ứng dụng sử dụng Kestrel server thì Nginx, Apache, hoặc IIS có thể sử dụng làm reverse proxy. Reverse proxy server sẽ nhận HTTP request từ phía server và chuyển chúng tới kestrel.

Sau đó chúng ta còn có thể host trên IIS,WebFarm hoặc Docker.

**Kestrel** là một webserver cho asp.net core đa nền tảng, khi khởi tạo ứng dụng asp.net core thì nó được mặc định sẵn là webserver của ứng dụng

**IIS** là viết tắt của từ (Internet Information Services ), được đính kèm với các phiên bản của Windows. Internet Information Services (IIS) chính là các dịch vụ dành cho máy chủ chạy trên nền hệ điều hành Window nhằm cung cấp và phân tán các thông tin lên mạng, nó bao gồm nhiều dịch vụ khác nhau như Web Server, FTP Server…

**Webfarm** là một môi trường host mà trong đó nó chứa nhiều web server. Thông thường một bộ load balancer sẽ liên kết các web server với nhau, giúp định tuyến các request đến các server trong web farm.

Việc setup và deploy application lên một hoặc nhiều server rất vất vả từ việc phải cài đặt các công cụ, môi trường cần cho application đến việc chạy được ứng dụng chưa kể việc không đồng nhất giữa các môi trường trên nhiều server khác nhau. Chính vì lý do đó **Docker** được ra đời để giải quyết vấn đề này. Docker là một nền tảng cho developers và sysadmin để develop, deploy và run application với container. Nó cho phép tạo các môi trường độc lập và tách biệt để khởi chạy và phát triển ứng dụng và môi trường này được gọi là container. Khi cần deploy lên bất kỳ server nào chỉ cần run container của Docker thì application của bạn sẽ được khởi chạy ngay lập tức.

**Azure App services:**

Một số khái niệm của Azure:

- Resource Group: Là một nhóm các dịch vụ khác nhau để dễ quản lý. Có thể dễ hiểu là mỗi ứng dụng bạn tạo mới 1 cái group này. Ví dụ khách hàng A có 1 web app hoá đơn điện tử, mình tạo 1 Resource Group riêng cho anh ấy. Chị B thuê mình làm ứng dụng rao vặt, thì mình tạo Resource Group riêng cho chị.

- App Services: Ở dưới localhost, bạn dùng máy tính của bạn, thì coi như bạn trả cho Azure một khoảng tiền để mua cái App Services này. App Services giống như 1 cái hosting nhưng có nhiều tính năng hơn như load balancing, autoscaling, hỗ trợ continuous deployment qua Github, Azure DevOps

- App Services Plan: Chi tiết cấu hình và giá mà bạn muốn cho cái App Services ở trên. Ví dụ như Ram mấy GB, CPU mấy core, máy chủ được đặt ở đâu, vv

- SQL Server (SQL Logic Server): Là "logical grouping of multiple Azure SQL Databases". Mình cũng không biết dịch tiếng Việt là gì. Nhưng một SQL Server sẽ có nhiều SQL Database con. Các SQL database con này sẽ dùng chung Server Admin username và password. Azure sẽ không tính phí khi tạo SQL Server vì mức phí sẽ quy định khi tạo SQL Database con.

Vậy để deploy một ứng dụng cơ bản, ta cần làm các bước sau (Linux hoặc MacOS):

1. Tạo Resource Group

2. Tạo App Services Plan

3. Tạo App Services

4. Tạo SQL Server

5. Tạo SQL Database

6. Config các kiểu

7. Push code lên App Services

Còn đối với Window thì visual studio hỗ trợ việc deploy trong sẵn bên trong nó.

* + 1. **Sercurity và Identity:**

**Authentication**:

Là quá trình xác thực danh tính người dùng, xem xét có cho người dùng truy cập đến tài nguyên hay không. Trong asp.net core, authentication được xử lý bởi authentication [middleware](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1).

Các cách authenticate thường được xử dụng:

* JWT
* Cookies

Ngoài ra chúng ta còn có thể sử dụng các dịch vụ bên thứ 3 để authenticate để đăng nhập nhanh chóng hơn ( social authentication ):

* Google
* Facebook
* Microsoft
* Twitter
* …

Đối với các ứng dụng được host bởi IIS, Kertel, HTTP.sys thì có Window authentication hỗ trợ. Bạn có thể sử dụng window authentication khi mạng lưới công ty sử dụng Active Directory Domains Identities hoặc tài khoản microsoft. Window authentication phù hợp sử dụng cho mạng nội bộ nơi mà người dùng, client app, web server đều nằm trên một Window domain.

**Active Directory (AD)** là một kiến trúc độc quyền của Microsoft. Đây là một kiến trúc không thể thiếu được trên Windows Server, được hiểu nôm na là một dịch vụ thư mục. Active Directory là một hệ thống được chuẩn hóa với khả năng quản trị tập trung hoàn hảo về người dùng cũng như các nguồn tài nguyên trong một hệ thống mạng. Active Directory được sử dụng trong mô hình mạng “Server - Client”.

Ngoài ra asp.net core còn hỗ trợ xác thực nhiều yếu tố ( **Multi-factor authentication )** như**:**

* **MFA, 2FA:** MFA yêu cầu 2 hoặc nhiều hơn các loại xác thực danh tính như: mã pin, password, sinh trắc học. 2FA thì chỉ cần 2 loại xác thực trong số trên.
* **MFA TOTP (Time-based One-time Password Algorithm):** Là loại xác thực sử dụng thuật toán TOPT. Chúng ta có thể sử dụng nó cùng với những authenticate app khác như: Microsoft authentication App, Google authentication app. Ví dụ như tạo ra mã QR code để đăng nhập, OTP code.( Vonage API, Twilio, ASPSMS)
* **MFA FIDO2 or passwordless:** là một chuẩn bảo mật mở được phát triển bởi liên minh FIDO.ASP.net code vẫn chưa hỗ trợ nhưng chúng ta có Azure Active Directory hỗ trợ chúng ta việc này.
* **MFA SMS:** Điều này tăng cường bảo mật hơn so với password authentication (single factor). Nhưng hiện nay có rất nhiều phương thức tấn công với dạng xác thực này nên bây giờ nó không còn được khuyến nghị sử dụng.

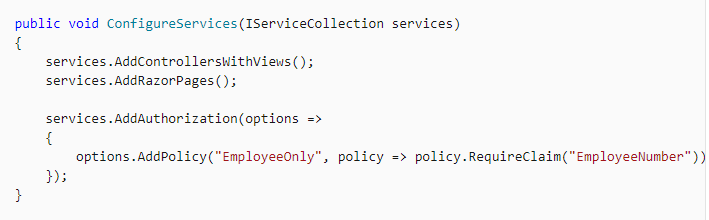
**Athorization**: xác định khả năng của bạn để truy cập hệ thống và ở mức độ nào. Khi danh tính của bạn được hệ thống xác minh sau khi xác thực thành công, bạn sẽ được phép truy cập tài nguyên của hệ thống. Hiện asp.net core có những cách sau đối với vấn đề này:

* Role-based authorization: dựa trên vai trò của người dùng mà từ đó phân quyền họ được vào những controller nào.

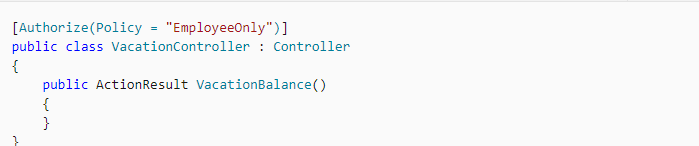


Như hình trên chỉ có người dùng với quyền Administratior mới có quyền vào controller này.

* Claims-based: claim là một cặp tên giá trị để thể hiện subject đó là gì, chứ không phải nó có thể làm gì. Đơn giản asp.net kiểm tra cặp giá trị đó và cho phép truy cập tài nguyên dựa vào nó. Một danh tính người dùng có thể có nhiều claims với nhiều giá trị và cũng có thể có claim cùng loại.

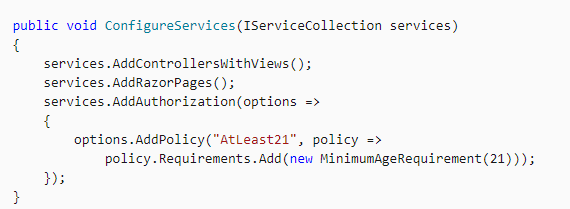


Hình trên thể hiện việc đăng kí policy



Chỉ có người dùng với identity là EmployeeOnly mới có thể truy cập Controller

* Policy-based: Yêu cầu một hoặc nhiều hơn một yếu tố xác thực.



Hình trên thể hiện việc đăng kí policy với yêu cầu số tuổi nhỏ nhất là 21

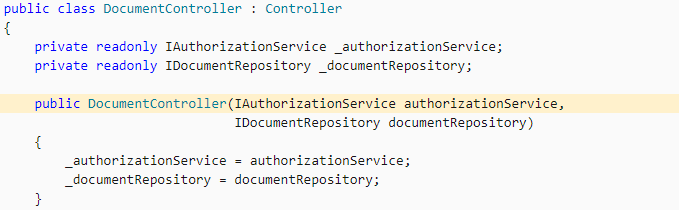


Có thể kết hợp thêm với claims-based authorization

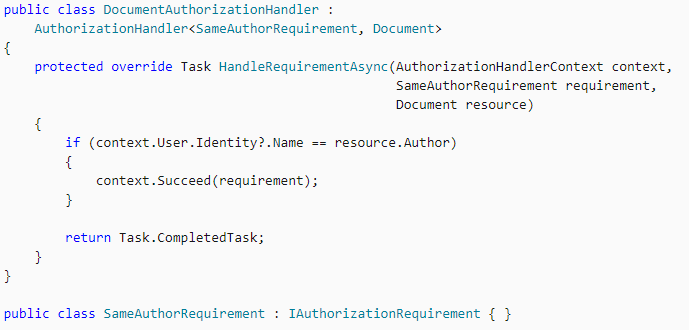


Thêm policy vào để phân quyền cho controller

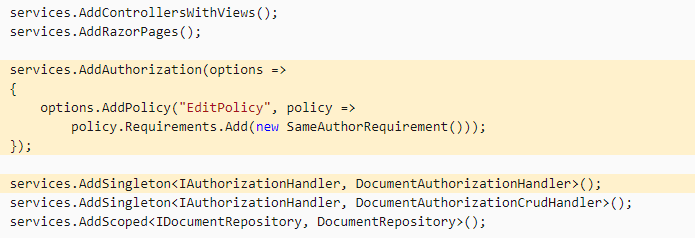
* Resource-based: Là một cách xác định resource nào đang được truy cập. Chỉ có tác giả mới có quyền chỉnh sửa resource, vì vậy phương thức phân quyền bình thường không thể sử dụng trong trường hợp này. Thay vào đó ta phải gọi một phương thức tùy chỉnh gọi là imperative authorization.



Sử dụng DJ cài đặt  [IAuthorizationService](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authorization.iauthorizationservice) vào controller

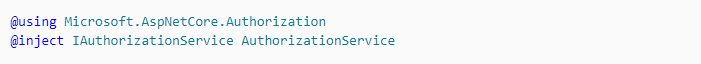


Viết một resource-base handler

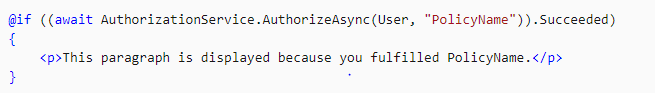


Cấu hình lại file Startup

* View-based: Được sử dụng trong MVC. Chúng ta sử dụng DI thêm phân quyền vào file cshtml



Nếu như muốn phân quyền vào mọi view thì chúng ta DI vào file *\_ViewImports.cshtml.*



Có thể kết hợp với resource-base để phân quyền.

* + 1. **Data Protection:**

**Data Protection**: Các ứng dụng web thường phải lưu trữ những dữ liệu bảo mật cho nên ASP.net core đã cung cấp một cryptographic API cho các lập trình viên để bảo vệ dữ liệu bao gồm cả key managerment và rotation.

Các APIs được chia theo nhu cầu của các lập trình viên gồm 3 nhóm sau:

* [Consumer APIs Overview](https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/overview?view=aspnetcore-3.1) hướng đến nhóm phát triển ứng dụng và framework

"I don't want to learn about how the stack operates or about how it's configured. I simply want to perform some operation in as simple a manner as possible with high probability of using the APIs successfully."

* [configuration APIs](https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/configuration/overview?view=aspnetcore-3.1) hướng đến nhóm phát triển ứng dụng và quản trị hệ thống

"I need to tell the data protection system that my environment requires non-default paths or settings."

* Extensibility APIs hướng đến nhóm chịu trách nhiệm về việc cài đặt các custom policy.

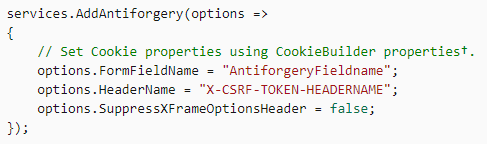
"I need to replace an entire component within the system because I have truly unique behavioral requirements. I am willing to learn uncommonly-used parts of the API surface in order to build a plugin that fulfills my requirements."

Chống các loại tấn công:

* **Cross-Site Request Forgery (XSRF/CSRF) attacks:** Antiforgery middleware hỗ trợ việc tránh tấn công dạng này. Đối với asp.net core 2.0 trở về sau thì khi sử dụng formtaghelper trong cshtml, nó tự động tạo antiforgery tokens.



Ngoài ra chúng ta có thể cấu hình việc này trong file startup hoặc trong controller.



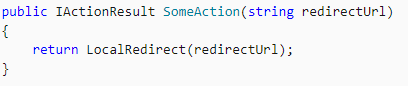
*File Startup*



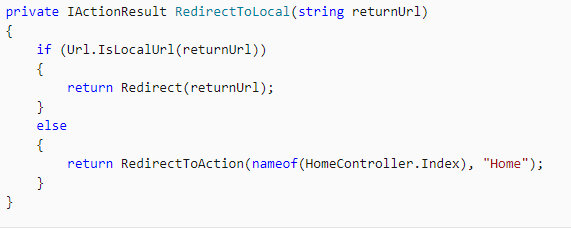
*File Controller*

* Open redirect Attack: Asp.net hỗ trợ các function kiểm tra giúp nhận diện website redirect tới có an toàn không như:

LocalRidirect

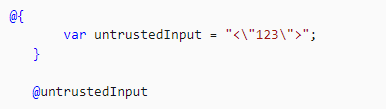


IsLocalUrl



* Cross-Site Scripting (XSS): có các cách như HTML encoding, JavaScript encoding sử dụng Razor hoặc sử dụng trong Controller.

HTML



JavaScript

Controller



* + 1. **Một số tính năng nâng cao:**

1. **Model binding**: Model binding là quá trình tự động trích dữ liệu từ truy vấn HTTP và biến đổi thành object .NET cung cấp cho action. Model binding diễn ra ngay sau khi routing lựa chọn được action để thực thi.

2. **Model validation**: Model cần kiểm tra dữ liệu đầu vào xem có hợp lệ không. Quá trình kiểm tra này có thể được hoàn thành bởi client trước khi gửi lên server hoặc server kiểm tra sau khi nhận được từ client. Cơ chế validation phía client (client-side validation) rất quan trọng vì nó giúp tăng trải nghiệm người dùng khi kiểm tra dữ liệu mà không cần chờ đến server nhưng phía server lại nên đảm bảo một lần nữa để các dữ liệu không hợp lệ không thể đưa vào hệ thống.

3. **Migration**: Migration là kỹ thuật trong việc tương tác với cơ sở dữ liệu, theo đó việc thay đổi về cấu trúc CSDL ở code sẽ được cập nhật lên CSDL đảm bảo dữ liệu đang tồn tại không bị mất, lịch sử (phiên bản) cập nhật được lưu lại sau mỗi lần cập nhật. Với migration khi bạn cập nhật Model, yêu cầu database cập nhật thì nó sẽ lưu thông tin phiên bản hiện tại của cấu trúc Model (database) ở Server DB - ví dụ phiên bản a, sau đó thay đổi các Model, lại yêu cầu cập nhật thì nó sẽ đọc thông tin phiên bản cuối trên DB, so sánh sự khác biệt và cập nhật sự khác biệt đó để lên phiên bản mới, phiên bản b.

Đối với Spring boot thì chúng ta phải sử dụng thư viện ngoài và cấu hình nên các bước setup để sử dụng rất tốn thời gian.

Tạo migration trong .NET Core

Add-Migration [NameMigration]

Cập nhật migration tới Database

Update-Database

Cập nhật database về migration cũ

Update-Database -Migration [OldNameMigration]

4. **Nâng cấp version**: Khi có những version mới ổn định và nhiều tính năng hơn thì chúng ta có thể dễ dàng nâng cấp phiên bản hiện tại lên với những bước đơn giản sau:

- Sửa tên .NET Core SDK version trong file global.json

- Sửa tên TargerFrameWork version trong file project

- Sửa tên PackageReference version trong file project

- Rebuild project

5. **ASP.NET core CLI**:

.NET Core command-line interface (CLI ) là một công cụ hỗ trợ đa nền tảng. Với nó chúng ta có thể sử dụng những IDE khác ngoài visual studio để khởi tạo , chạy, phát triển và deploy các ứng dụng .net core thông qua các lệnh.

Các lệnh đều theo cấu trúc sau:

dotnet <command> <argument> <option>

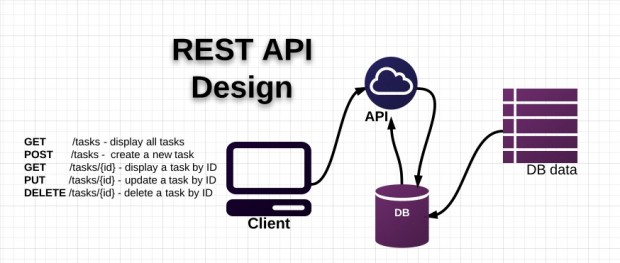
| Basic Commands | Description |
| --- | --- |
| [new](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-new) | Creates a new project, configuration file, or solution based on the specified template. |
| [restore](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-restore) | Restores the dependencies and tools of a project. |
| [build](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-build) | Builds a project and all of its dependencies. |
| [Run](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-run) | Runs source code without any explicit compile or launch commands. |
| [publish](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-publish) | Packs the application and its dependencies into a folder for deployment to a hosting system. |
| [test](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-test) | Executes unit tests. |
| [vtest](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-vstest) | Runs tests from the specified files. |
| [pack](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-pack) | Packs the code into a NuGet package. |
| [clean](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-clean) | Cleans the output of a project. |
| [sln](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-sln) | Modifies a .NET Core solution file. |
| [help](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-help) | Display help on the specified command |
| [store](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-store) | Stores the specified assemblies in the runtime package store. |

| Project Modification Commands | Description |
| --- | --- |
| [add package](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-package) | Adds a package reference to a project. |
| [add reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-add-reference) | Adds project-to-project (P2P) references. |
| [remove package](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-remove-package) | Removes package reference from the project. |
| [remove reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-remove-reference) | Removes project reference |
| [list reference](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-list-reference) | Lists all project-to-project references |

| Advanced Commands | Description |
| --- | --- |
| [nuget delete](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-delete) | Deletes or unlists a package from the server. |
| [nuget locals](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-locals) | Clears or lists local NuGet resources. |
| [nuget push](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-nuget-push) | Pushes a package to the server and publishes it. |
| [msbuild](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-msbuild) | Builds a project and all of its dependencies. |
| [dotnet install script](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-install-script) | Script used to install the .NET Core CLI tools and the shared runtime. |

* 1. **Restful API:**

**RESTful API**là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.



**Tổng quan restful api**

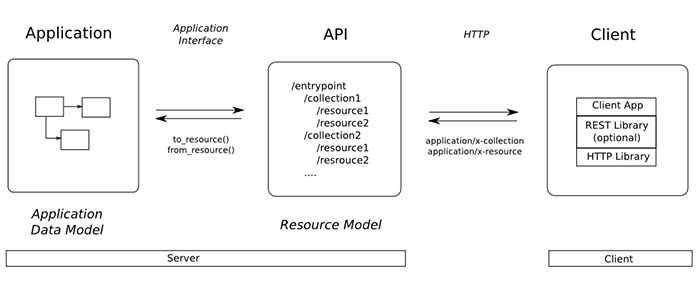
**API** (**A**pplication **P**rogramming **I**nterface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như [JSON](https://topdev.vn/blog/json-la-gi/) hay XML.

**REST** (**RE**presentational **S**tate **T**ransfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.

**RESTful API** là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.

Chức năng quan trọng nhất của **REST** là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một **RESTful API**.

**Cách thức hoạt động:**



Cách thức hoạt động RESTful API

REST hoạt động chủ yếu dựa vào giao thức HTTP. Các hoạt động cơ bản nêu trên sẽ sử dụng những phương thức HTTP riêng.

* GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
* POST (CREATE): Tạo mới một Resource.
* PUT (UPDATE): Cập nhật thông tin cho Resource.
* DELETE (DELETE): Xoá một Resource.

Những phương thức hay hoạt động này thường được gọi là CRUD tương ứng với Create, Read, Update, Delete – Tạo, Đọc, Sửa, Xóa.

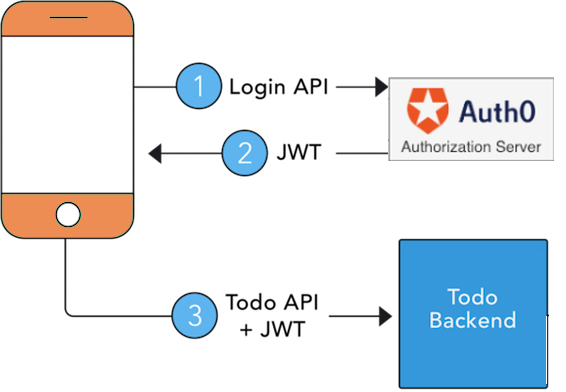
Hiện tại đa số lập trình viên viết RESTful API giờ đây đều chọn JSON là format chính thức nhưng cũng có nhiều người chọn XML làm format, nói chung dùng thế nào cũng được miễn tiện và nhanh.

* 1. **Sercurity:**

Authentication là phần không thể thiếu trong bất kỳ hệ thống nào. Phương phápauthentication đơn giản và hay được sử dụng trong các ứng dụng web đó là user gửi thông tin username và password lên server. Sau khi server chứng thực thành công sẽ tạo ra một chuỗi session\_id và lưu vào session hay database ở phía server. Sau đó, gửi session\_id này về client và được client lưu trên cookie. Nhưng với các ứng dụng trên mobile và các ứng dụng web SPA (Single Page Application) thì cần có cơ chế authentication tốt hơn khi mà chúng ta phải thiết kế các RESTful api (stateless) thì server không thể đảm nhiệm việc lưu trạng thái phiên làm việc của user. Một trong những phương pháp tốt để giải quyết vấn đề này là sử dụng JSON Web Token (JWT).

### 2.4.1 Giới thiệu Json Web Token (JWT)

Authentication là phần không thể thiếu trong bất kỳ hệ thống nào. Phương pháp authentication đơn giản và hay được sử dụng trong các ứng dụng web đó là user gửi thông tin username và password lên server. Sau khi server chứng thực thành công sẽ tạo ra một chuỗi session\_id và lưu vào session hay database ở phía server. Sau đó, gửi session\_id này về client và được client lưu trên cookie. Nhưng với các ứng dụng trên mobile và các ứng dụng web SPA (Single Page Application) thì cần có cơ chế authentication tốt hơn khi mà chúng ta phải thiết kế các RESTful api (stateless) thì server không thể đảm nhiệm việc lưu trạng thái phiên làm việc của user. Một trong những phương pháp tốt để giải quyết vấn đề này là sử dụng JSON Web Token (JWT).

[](https://gpcoder.com/wp-content/uploads/2019/06/jwt-access-token.png)

JWT architechture

JSON Web Token (JWT) là gì?

JSON Web Token (JWT) là 1 tiêu chuẩn mở ([RFC 7519](https://tools.ietf.org/html/rfc7519)), 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 lạibằng 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 vấn cơ sở dữ liệu nhiều lần).

# **XÂY DỰNG ỨNG DỤNG**

## **Tổng quan khảo sát các ứng dụng liên quan**

- Khi xây dựng Mobile Banking, ngân hàng cần quyết định đi theo một trong hai hướng. Thứ nhất là tự xây dựng toàn bộ hệ thống Mobile Banking, và thứ hai là kết hợp với một đối tác chuyên nghiệp đã có sẵn hạ tầng viễn thông.

Theo hướng thứ nhất - tự xây dựng: theo cách này, ngân hàng sẽ gặp phải khá nhiều khó khăn:

- Xây dựng hạ tầng viễn thông: việc kết nối với các mạng di động là công việc đòi hỏi phải có đội ngũ chuyên gia trong lĩnh vực viễn thông để hoàn thành tất cả các công đoạn: kết nối đường truyền, thử nghiệm dịch vụ, tính cước. Công việc này không chỉ đòi hỏi tính chính xác, chuyên nghiệp mà cũng là công việc rất mất thời gian.

- Chi phí thiết bị và phí duy trì hệ thống: (i) Phí thiết bị: Để xây dựng một hạ tầng viễn thông, ngân hàng sẽ phải bỏ tiền để mua tất cả những thiết bị đắt tiền như: máy chủ, thiết bị định tuyến, bộ chuyển đổi tín hiệu và những thiết bị khác. (ii) Phí duy trì hệ thống: ngoài những khoản phí cố định hàng tháng cho các công ty truyền dẫn, phí thuê đầu số cho các mạng di động, ngân hàng còn phải trả tiền để duy trì đội ngũ chuyên gia, nhân viên kỹ thuật quản lý hệ thống, bảo trì hệ thống, trang thiết bị hỗ trợ giúp hệ thống hoạt động thông suốt, chính xác 24/24. Đây cũng là khoản chi phí không nhỏ.

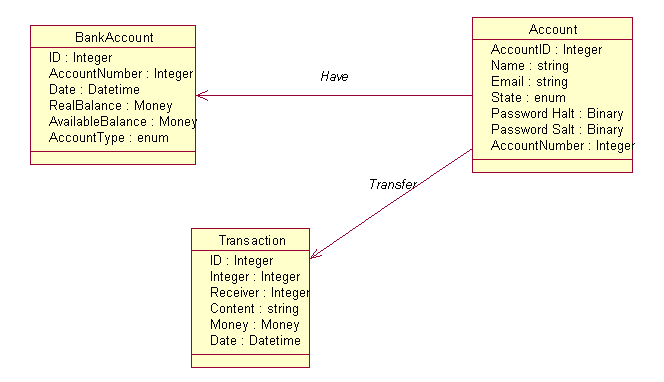
- Thực tế đã có một số ngân hàng tự triển khai Mobile Banking như Ngân hàng TMCP Hàng hải (MSB), VPBank. Trong quá trình triển khai đã gặp khá nhiều khó khăn vì thiếu các chuyên gia về lĩnh vực viễn thông, thời gian kết nối với các mạng di động kéo dài. Cuối cùng, do các dịch vụ Mobile Banking nghèo nàn trên nền công nghệ kém đã không thu hút được khách hàng, dẫn đến không bù đắp được khoản đầu tư vào Mobile Banking.

Theo hướng thứ hai - kết hợp với một đối tác chuyên nghiệp đã có sẵn hạ tầng viễn thông, ngân hàng sẽ được lợi nhiều mặt: Giải pháp Mobile Banking tốt hơn, các đối tác thứ ba hiểu rõ các nghiệp vụ ngân hàng, cam kết bảo mật tuyệt đối, hỗ trợ ngân hàng tốt nhất và xử lý sự cố nhanh nhất; Triển khai nhanh hơn: hạ tầng của các đối tác thứ ba đã sẵn sàng kết nối với ngân hàng. Chi phí thấp hơn: Mobile Banking triển khai trên hạ tầng ổn định sẵn có, các đối tác thứ ba sẽ lo toàn bộ chi phí quản trị và duy trì hệ thống, phí đường truyền kết nối tới các mạng di động.

## **3.2 Cách thiết kế database, cách cài đặt phần mềm**

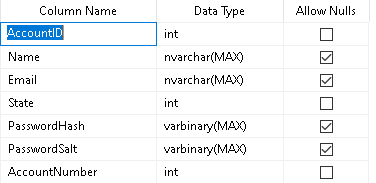
### 3.2.1 Database:

**Tổng quan:**

****

Bảng Class Diagram cơ sở dữ liệu

Bảng Account:



Chi tiết bảng Account

Trong đó:

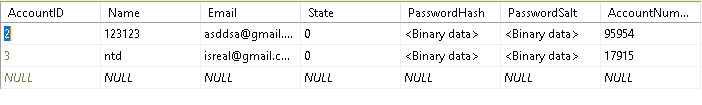
- AccountID: ID của account

- Email: email của account

- State: trạng thái của tài khoản gồm : Active, Unactive

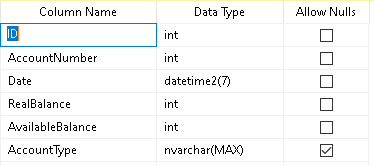
- PasswordHash, PasswordSalt: là mật khẩu của account được băm ra từ mật khẩu thực thành 2 mật khẩu trên

- AccountNumber: Số tài khoản ngân hàng



Bảng dữ liệu Account

**Bảng BankAccount**: Lưu thông tin ngân hàng của khách hàng



Thông tin chi tiết bảng BankAccount

Trong đó :

- ID: ID của tài khoản ngân hàng

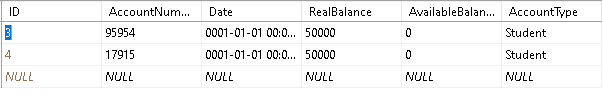
- AccountType là loại tài khoản gồm : Sinh viên, Cá nhân, Doanh Nghiệp

- Accountnumber: số tài khoản

- Date: ngày lập tài khoản

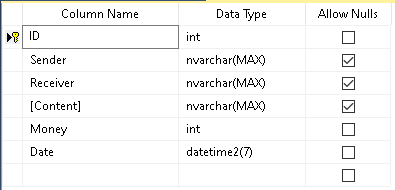
- Realbalance: tài khoản thực

- Availablebalance: tài khoản khả dụng



Bảng dữ liệu BankAccount

**Bảng Transaction**



Thông tin chi tiết bảng Transaction

Trong đó:

- ID: ID của giao dịch

- Sender: số tài khoản người gửi

- Receiver: số tài khoản người nhận

- Content: nội dung giao dịch

- Money: số tiền chuyển khoản

- Date: thời gian chuyển khoản

**Cách cài đặt phần mềm:**

### 3.2.2 FrontEnd:

- Cài đặt Node.js https://nodejs.org/

- Chạy câu lệnh sau npm install -g @angular/cli trong window command promd

- Mở cmd trong file front end

- Chạy lệnh npm start

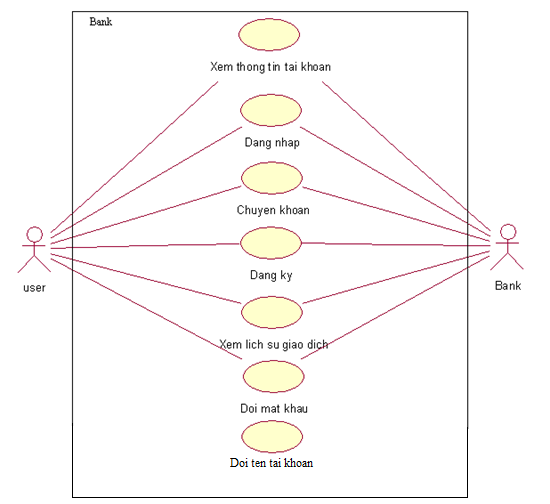
### 3.2.3 BackEnd:

- cài đặt .net core <https://dotnet.microsoft.com/download>

- chạy file project

- ctrl + f5 để chạy backend

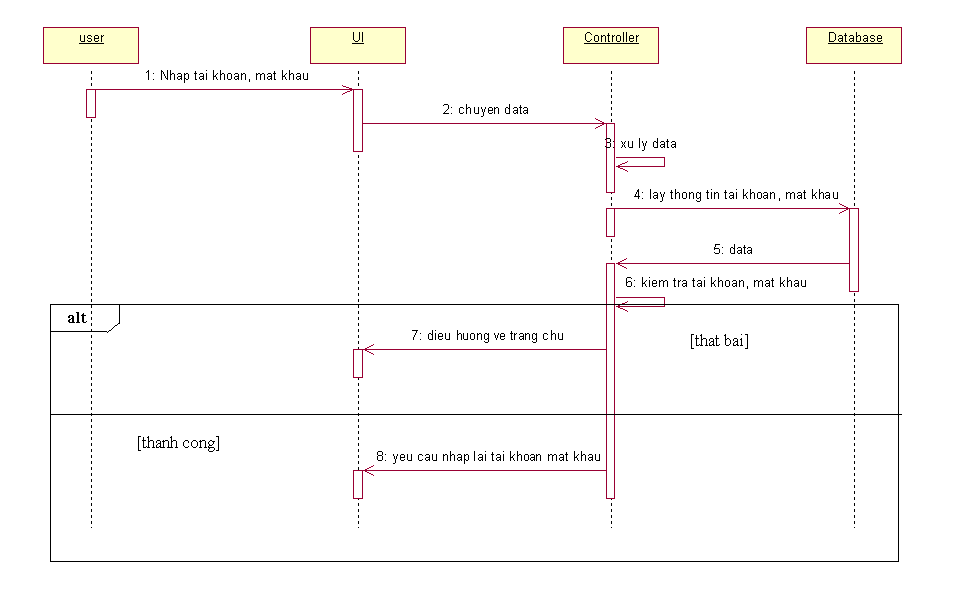
## **Usecase Diagram**

****

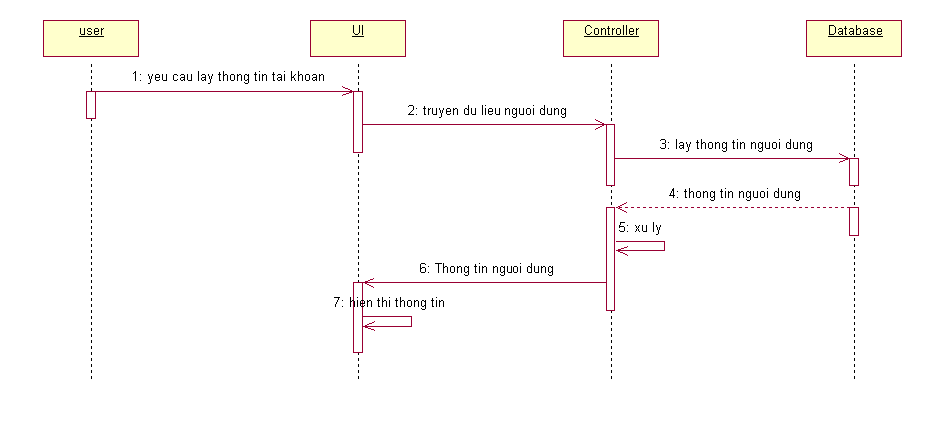
## **Sequence Diagram**



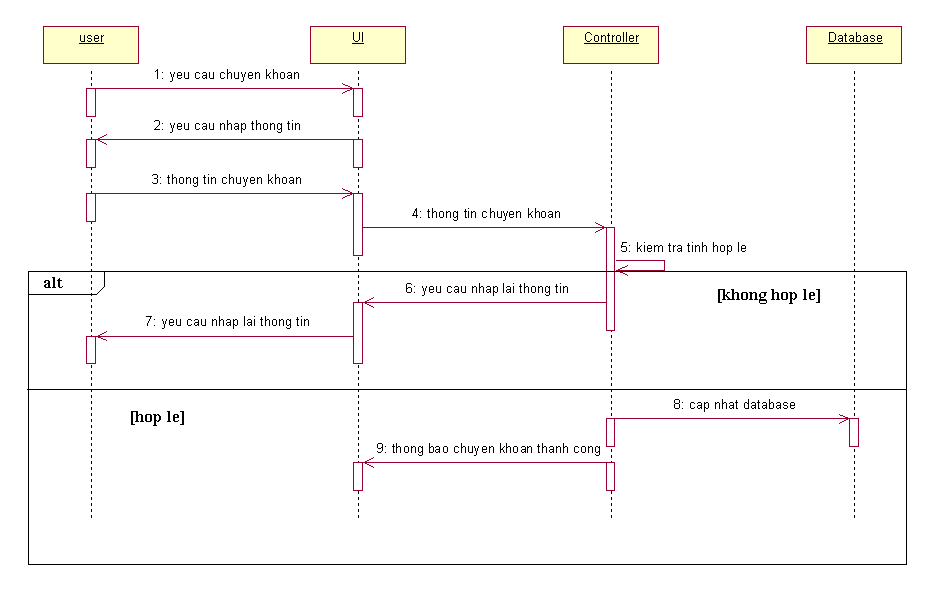
### Login:

****

### 3.4.2 Xem thông tin:

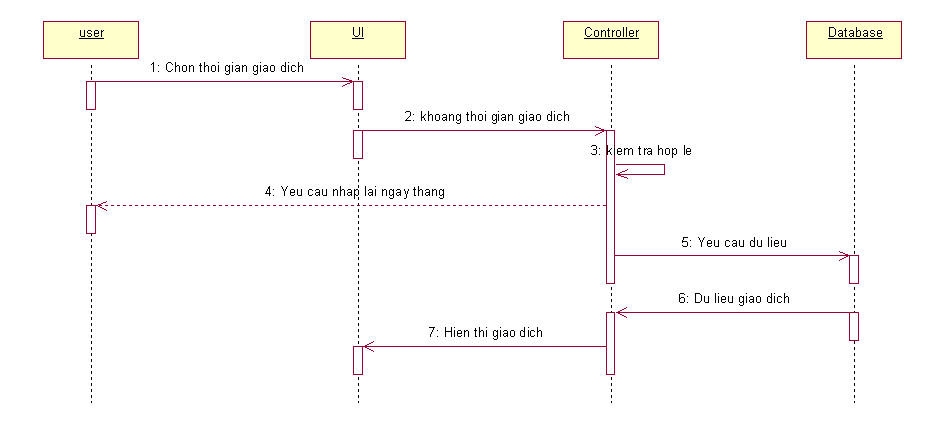
****

### 3.4.3 Chuyển khoản:

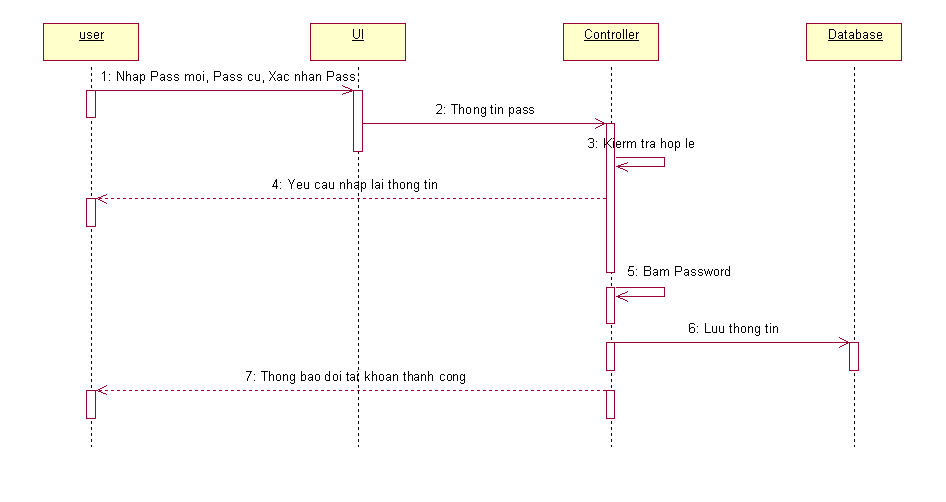
****

### 3.4.4 Đăngký:

### 3.4.5 Xem thông tin giao dịch:

****

### 3.4.6 Đổi mật khẩu, Đổi tên tài khoản:

****

## **Ứng dụng của công nghệ vào trong ứng dụng:**

**RESTful API** là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.

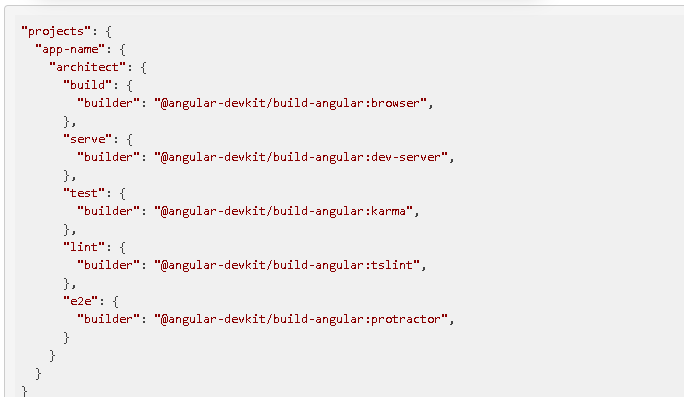
**Hỗ trợ SVG Template**

Hiện tại Angular 8 hỗ trợ template với đuôi mở rộng là SVG. Nghĩa là chúng ta có thể dùng file SVG để làm template như HTML mà không cần cấu hình gì khác cả. Nhưng câu hỏi là tại sao phải sử dụng .svg như là template thay vì sử dụng image trong HTML file. Lý do là khi sử dụng SVG template, chúng ta có thể sử dụng như là một directive và kết quả chúng ta có thể bind nó như HTML template. Với các tiếp cận này, chúng ta có thể tạo động các đồ hoạt tương tác trong ứng dụng.



Hỗ trợ thư viện mới New Builders/ Architect API

Phiên bản mới của Angular CLI cho phép chúng ta sử dụng phiên bản mới của Builders được biết đến là Architect API. Angular sử dụng Builders API để thực hiện các hoạt động như server, build, test, lint và e2e. Chúng ta có thể sử dụng builders trong file angular.json

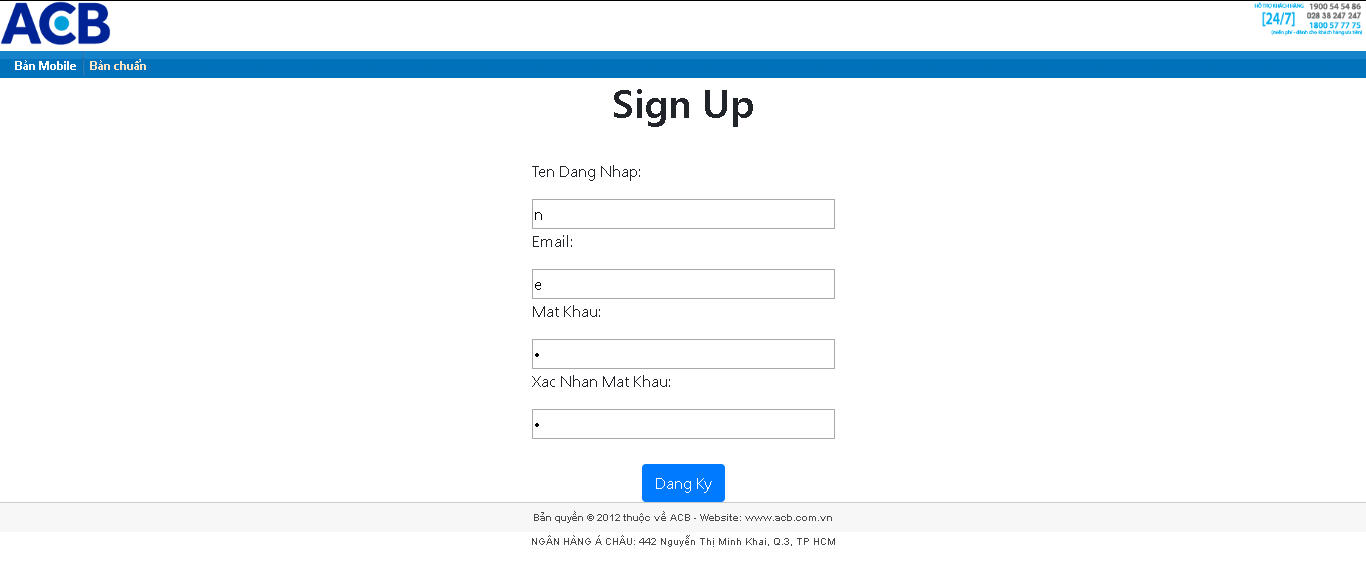


**Angular – Service và Dependency Injection**

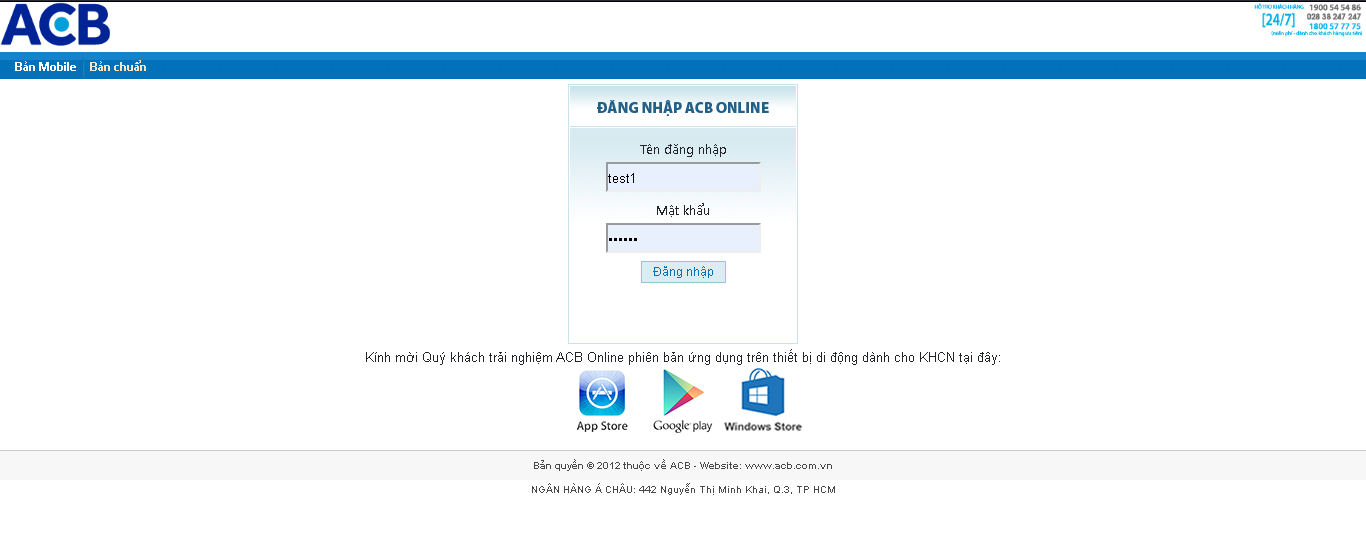
**4.7** / **5** ( **54** votes )

Service (dịch vụ) chẳng qua cũng là một cách giúp cho chúng ta tái sử dụng code mà thôi, chẳng hạn như bạn có một lớp Customer, thì thay vì mỗi lần cần lấy các đối tượng Customer đang có, chúng ta phải viết code để tạo đối tượng, truyền tham số…v.v ở nhiều nơi khác nhau, thì bây giờ chúng ta chỉ cần viết một lớp service làm điều đó luôn cho chúng ta, như vậy việc quản lý code sẽ dễ dàng hơn, chẳng hạn như mỗi lần thay đổi phương thức khởi tạo, thì chúng ta chỉ cần thay đổi code trong lớp service là được, thay vì phải đi sửa lại toàn bộ những dòng code khởi tạo đó.

Dependency Injection là chức năng cho phép chúng ta “nhúng” các lớp vào các lớp khác, giống như dùng một thư viện vậy, và chúng ta có thể dùng các lớp được nhúng vào đó giống như dùng một thuộc tính bình thường mà không cần phải thực hiện các công đoạn khai báo, khởi tạo…v.v

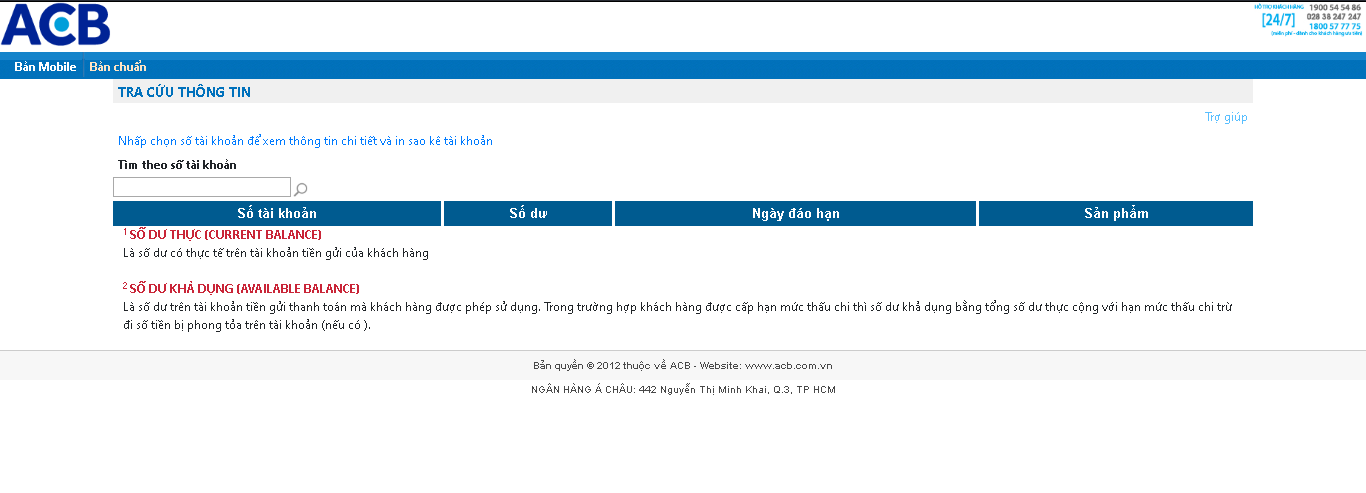
1. **THỬ NGHIỆM VÀ ĐÁNH GIÁ**
3. 1. **Môi trường thử nghiệm:**
   * **Node JS 6.9.0**
   * **Asp.net core 2.2**
   * **Window 10**
   * **SQL sever 2017**
   1. **Giao diện:**
      1. **Đăng ký:**

Màn hình đăng ký

* + 1. **Đăng nhập:**

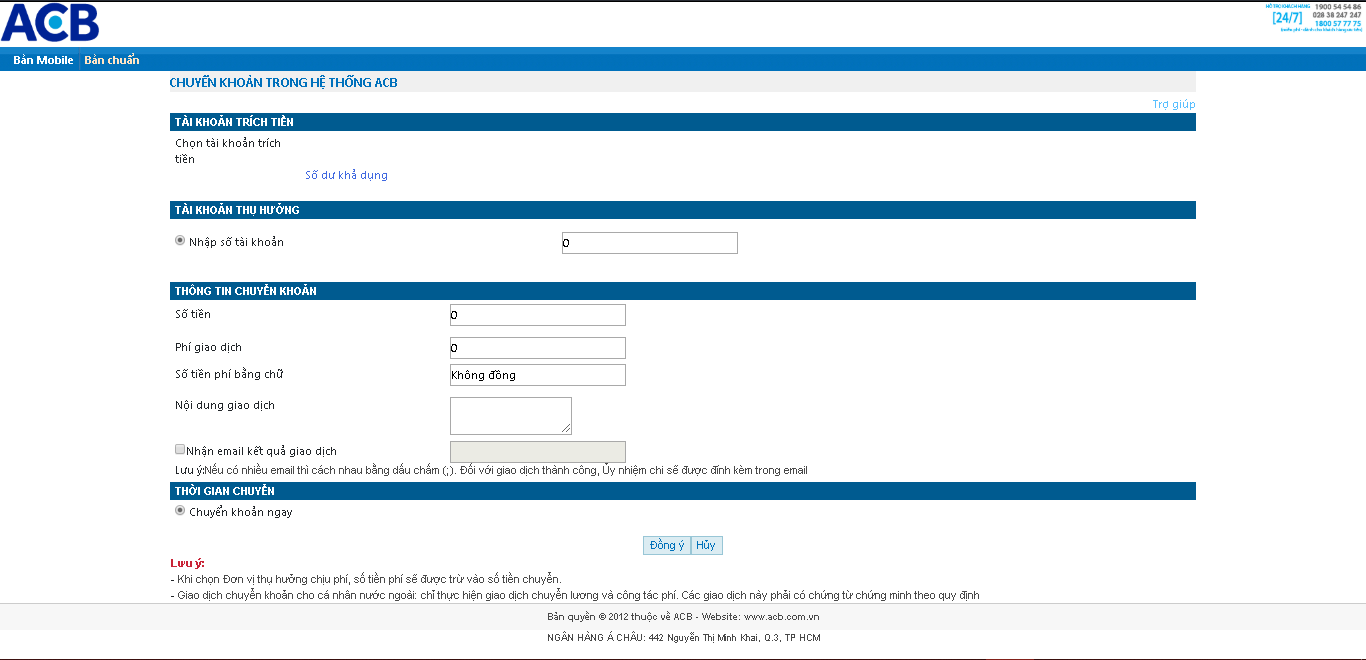
Màn hình đăng nhập

* + 1. **Thông tin tài khoản:**

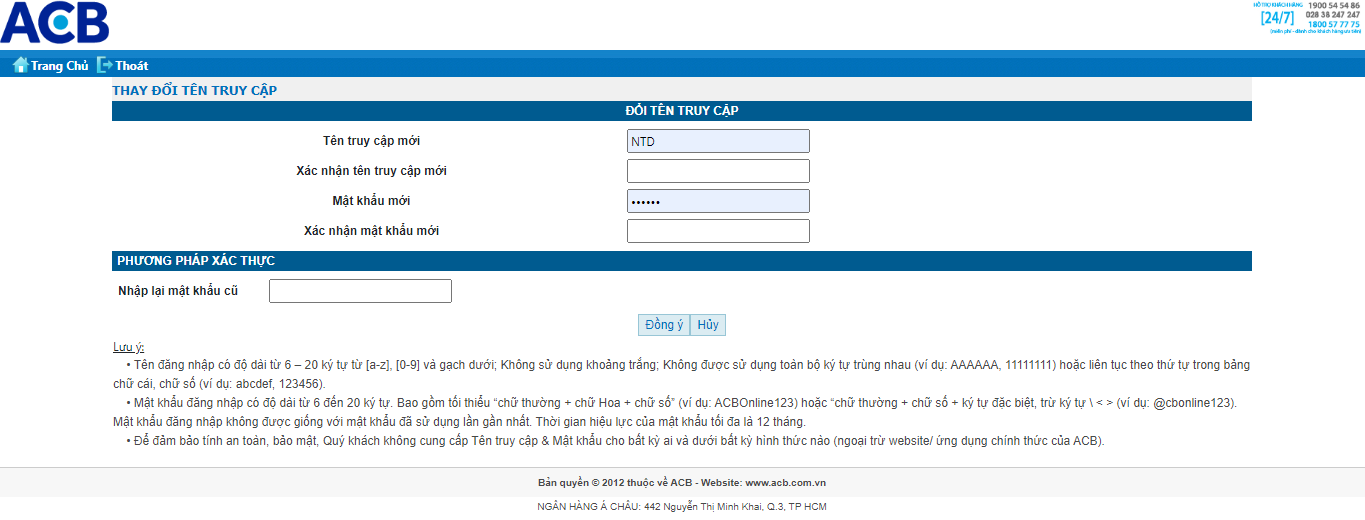
Màn hình thông tin tài khoản

* + 1. **Chuyển khoản:**

Màn hình chuyển khoản

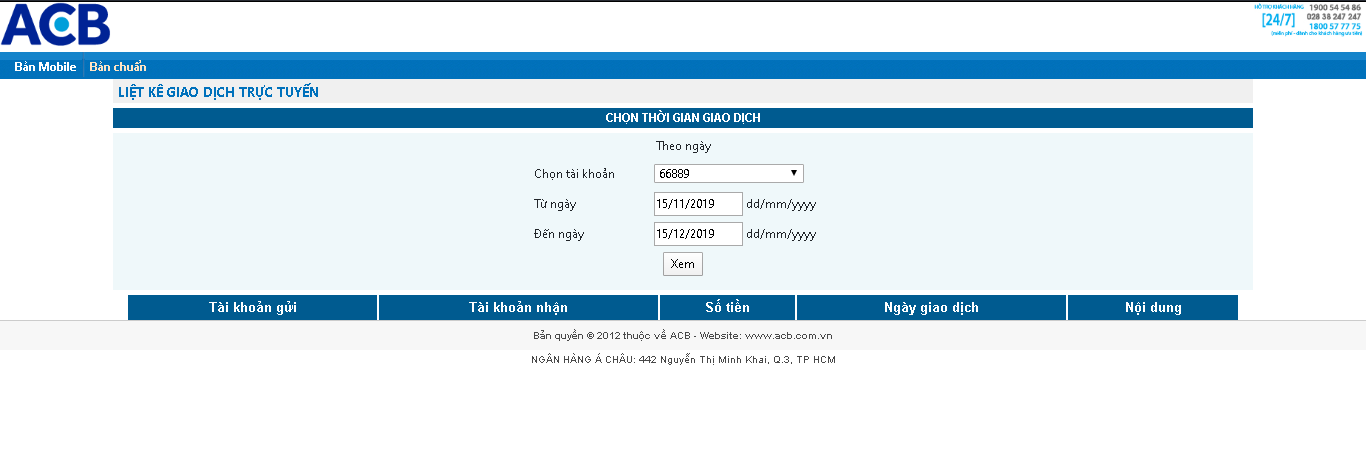


* + 1. **Đổi mật khẩu:**



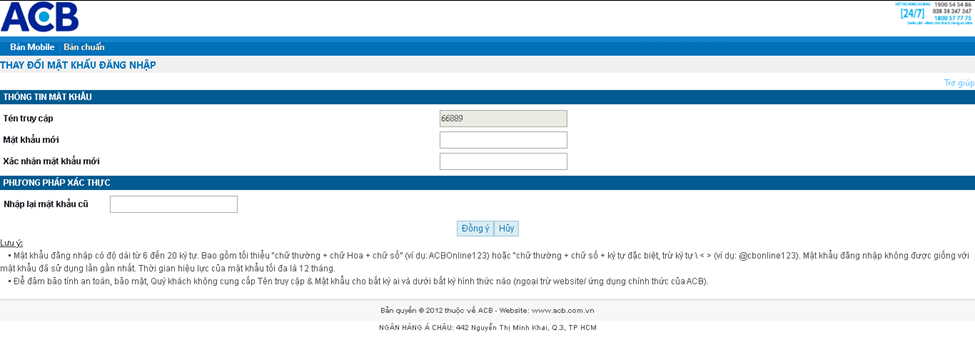
Màn hình đổi mật khẩu

* + 1. **Lịch sử giao dịch:**



Màn hình lịch sử giao dịch

* + 1. **Đổi tên đăng nhập:**



Màn hình đổi tên đăng nhập

* 1. **Kết quả đạt được:**

Sau khi hoàn thành đồ án với những chức năng như :

* + đăng nhập
  + xem thông tin tài khoản
  + tạo tài khoản
  + chuyển khoản nội bộ
  + xem lịch sử giao dịch
  + đổi mật khẩu

Sử dụng JWT authentication để bảo mật ứng dụng.

Database được thiết kế hợp lý, mật khẩu người dùng được băm ra để bảo vệ người dùng.

Sử dụng API để giao tiếp giữa client và sever

* 1. **Hạn chế và hướng phát triển:**

Ứng dụng chưa thể thực hiện các chức năng thực thụ như là một app E-banking như thanh toán dịch vụ, …. Do chưa có liên kết với các bên viễn thông và ngân hàng thật.

Hướng phát triển sẽ cố gắng phát triển ứng dụng có thể sử dụng mục đích thương mại được.

1. **TÀI LIỆU THAM KHẢO:**

**Tiếng việt:**

[1] Vương Đức Hoàng Quân, Nguyễn Thanh Quang, *VAI TRÒ INTERNET BANKING VÀ BÀI HỌC KINH NGHIỆM CHO NGÂN HÀNG THƯƠNG MẠI VIỆT NAM,* <https://www.vhu.edu.vn/Resources/Docs/SubDomain/qlkh/Vai%20tr%C3%B2%20c%E1%BB%A7a%20internet%20banking%20v%C3%A0%20b%C3%A0i%20h%E1%BB%8Dc%20kinh%20nghi%E1%BB%87m%20cho%20ng%C3%A2n%20h%C3%A0ng%20th%C6%B0%C6%A1ng%20m%E1%BA%A1i%20Vi%E1%BB%87t%20Nam.pdf>

[2] Topdev,*JSON Web Token (JWT) là gì ?,* <https://topdev.vn/blog/json-web-token-jwt-la-gi/>

[3] xluffy, *Băm và lưu password đúng cách,* <https://xluffy.github.io/post/how-to-hash-store-password/>

**Tiếng anh:**

[4] Unknown, *Angular documentation*, <https://angular.io/docs>

[5] Unknown, *ASP.NET Core 2.2 documentation,*<https://docs.microsoft.com/vi-vn/dotnet/core/>

[6] Unknown, *What is REST*, <https://restfulapi.net/>