LỜI CẢM ƠN

Lời đầu tiên em xin chân thành cảm ơn đến ban giám hiệu trường Đại học Bình Dương và đặc biệt là toàn thể quý thầy cô trong Khoa Tin học lời cảm ơn chân thành nhất. Thầy cô đã truyền đạt lại cho em những kiến thức căn bản nhất, từ đó giúp em có được nền tảng vững chắc để có thể tìm hiểu được những kiến thức mới, những phương thức mới trong học tập lẫn trong cuộc sống.

Em xin cảm ơn thầy TS. Tô Tuấn, người đã hướng dẫn em khi đề tài này còn là đồ án ngành. Thầy đã cho em những lời khuyên chân thành, những tầm nhìn xa trong việc phát triển phần mềm. Do sự lưỡng lự trong việc chọn đồ án tốt nghiệp của mình, em đã bỏ lỡ cơ hội được hoàn thành đồ án tốt nghiệp dưới sự hướng dẫn của thầy. Đó là một điều khiến em tiếc nuối.

Em xin cảm ơn Thầy ThS. Nguyễn Hoàng Phong, người hướng dẫn em hoàn thành đồ án tốt nghiệp. Thầy đã theo sát em từng ngày và chỉ bảo em hết sức tận tình, giúp em có được những ý tưởng, giải pháp rất hiệu quả. Có lẽ mọi việc sẽ trở nên rất khó khăn với em nếu như không được thầy hướng dẫn. Em cảm ơn thầy.

Em cũng xin gửi lời cảm ơn đến người thân, bạn bè đã giúp đỡ em hoàn thành đồ án này!

|  |
| --- |
| Bình Dương, tháng 4 năm 2016 |

|  |
| --- |
| Lê Khả Sỹ |

|  |
| --- |
|  |

NHẬN XÉT ĐÁNH GIÁ CỦA GIÁO VIÊN HƯỚNG DẪN

|  |
| --- |
| Bình Dương, ngày… tháng .... năm ....... |

NHẬN XÉT ĐÁNH GIÁ CỦA GIẢNG VIÊN PHẢN BIỆN

|  |
| --- |
| Bình Dương, ngày… tháng .... năm ....... |

# DANH SÁCH CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Viết tắt** | **Đầy đủ** | **Ý nghĩa** |
| RNS | Raw Notification System | Hệ thống thông báo (project hiện tại) |
| AES | Already Exists System | Hệ thống đã có sẵn mà cần cài đặt RNS lên. |
| WNS | Windows Notification Service | Dịch vụ thông báo của windows 10 |
| URI | Uniform Resource Identifier | Một chuỗi ký tự đại diện cho một ứng dụng trên một thiết bị. |
| UWP | Uiversal Windows Platform | Loại ứng dụng store chạy trên windows 10. |
| GUID | Global Unique Identifiy | Là một cấu trúc chứa 128bit có giá trị ngẫu nhiên, được biểu diễn dưới dạng 32 ký tự HEX. Thường được dùng làm Token hoặc ID. |
| CIL | Common Intermediate Language | Khi một ứng dụng .Net được biên dịch, nó là một file exe chứa mã CIL chứ không phải mã máy. Mã CIL là ngôn ngữ lập trình bậc thấp nhất mà con người có thể đọc được. CIL được định nghĩa bởi Common Language Infrastructure (CLI) và được sử dụng bởi .Net Framework và Mono. CIL còn được gọi là managed code. |
| CLR | Common Language Runtime | Khi một file exe của .Net được chạy, nó sẽ được thông dịch lại thành mã máy từ mã CIL. CLR chính là môi trường để chạy CIL của .Net. Có thể xem như đây là một máy ảo giúp biên dịch managed code thành mã máy sao cho phù hợp với môi trường mà nó đang chạy, tương tự như JVM của JAVA. |
| ORM | Object Relational Model | Là một kỹ thuật chuyển đổi dữ liệu không hướng đối tượng thành dữ liệu hướng đối tượng. Thường được dùng trong mapping data khi đọc từ DB lên. |

MỤC LỤC

[DANH SÁCH CHỮ VIẾT TẮT iv](#_Toc456867589)

[CHƯƠNG 2. MỞ ĐẦU 1](#_Toc456867590)

[2.1 LÝ DO HÌNH THÀNH ĐỀ TÀI 1](#_Toc456867591)

[2.2 MỤC TIÊU ĐỀ TÀI 2](#_Toc456867592)

[2.3 Ý NGHĨA THỰC TIỄN CỦA ĐỀ TÀI 2](#_Toc456867593)

[2.4 PHẠM VI ĐỀ TÀI 2](#_Toc456867594)

[CHƯƠNG 3. CƠ SỞ LÝ THUYẾT 3](#_Toc456867595)

[3.1 CÁC GIẢI PHÁP 3](#_Toc456867596)

[3.2 WINDOWS NOTIFICATION SERVICE (WNS) 3](#_Toc456867597)

[3.2.1 WNS là gì 3](#_Toc456867598)

[3.2.2 WNS làm việc như thế nào? 4](#_Toc456867599)

[3.2.3 Các vấn đề bảo mật của mô hình WNS. 5](#_Toc456867600)

[3.2.4 Làm việc với WNS 5](#_Toc456867601)

[3.3 CÁC GIẢI PHÁP, CÔNG NGHỆ, PATTERM KHÁC ĐƯỢC SỬ DỤNG 16](#_Toc456867602)

[3.3.1 Windows communication foundation (WCF) 16](#_Toc456867603)

[3.3.2 Dapper .Net 20](#_Toc456867604)

[3.3.3 Mô hình nhiều lớp 21](#_Toc456867605)

[CHƯƠNG 4. THIẾT KẾ TỔNG QUÁT RNS TRÊN NỀN TẢNG .NET FRAMEWORK 22](#_Toc456867606)

[4.1 MỤC TIÊU MÀ BẢN THIẾT KẾ HƯỚNG ĐẾN 22](#_Toc456867607)

[4.2 BẢN THIẾT KẾ TỔNG QUÁT CÁC THÀNH PHẦN CỦA HỆ THỐNG 22](#_Toc456867608)

[4.3 SƠ ĐỒ STATE CỦA NOTIFICATION DATA 24](#_Toc456867609)

[4.4 BẢN THIẾT KẾ MÔ HÌNH AUTHENTICATION VÀ AUTHORIZATION CHO RNS 25](#_Toc456867610)

[4.5 BẢN THIẾT KẾ CƠ SỞ DỮ LIỆU 27](#_Toc456867611)

[4.6 SƠ ĐỒ GIAO TIẾP 29](#_Toc456867612)

[CHƯƠNG 5. THIẾT KẾ CHI TIẾT RNS 31](#_Toc456867613)

[5.1 CÁC SƠ ĐỒ TRẠNG THÁI 31](#_Toc456867614)

[5.1.1 Notification 31](#_Toc456867615)

[5.1.2 Receiver 32](#_Toc456867616)

[5.1.3 Device 33](#_Toc456867617)

[5.2 CÁC SƠ ĐỒ HOẠT ĐỘNG 33](#_Toc456867618)

[5.2.1 Add Receiver 34](#_Toc456867619)

[5.2.2 Add Device 35](#_Toc456867620)

[5.2.3 Add Notification 36](#_Toc456867621)

[5.2.4 Send All Notification 37](#_Toc456867622)

[CHƯƠNG 6. CÔNG CỤ PHÁT TRIỂN, CÀI ĐẶT 38](#_Toc456867623)

[6.1 CÔNG CỤ PHÁT TRIỂN 38](#_Toc456867624)

[6.2 HƯỚNG DẪN CÀI ĐẶT RNS CHO NHÀ PHÁT TRIỂN AES 38](#_Toc456867625)

[6.2.1 Khởi tạo Database 38](#_Toc456867626)

[6.2.2 Config RNS 38](#_Toc456867627)

[6.2.3 Cài đặt module giao tiếp với Server service 39](#_Toc456867628)

[6.2.4 Xây dựng login service 41](#_Toc456867629)

[6.2.5 Xây dựng Mobile Application 41](#_Toc456867630)

[CHƯƠNG 7. KIỂM THỬ 44](#_Toc456867631)

[7.1 PHƯƠNG PHÁP KIỂM THỬ 44](#_Toc456867632)

[7.2 MỤC TIÊU KIỂM THỬ 44](#_Toc456867633)

[7.3 CÀI ĐẶT MÔI TRƯỜNG KIỂM THỬ 44](#_Toc456867634)

[7.4 TRIỂN KHAI RNS TRÊN MÔI TRƯỜNG KIỂM THỬ 44](#_Toc456867635)

[7.5 KẾT QUẢ KIỂM THỬ 47](#_Toc456867636)

[CHƯƠNG 8. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 48](#_Toc456867637)

[8.1 KẾT QUẢ ĐẠT ĐƯỢC 48](#_Toc456867638)

[8.2 ĐÁNH GIÁ KẾT QUẢ 48](#_Toc456867639)

[8.3 HƯỚNG PHÁT TRIỂN 49](#_Toc456867640)

[Tài liệu tham khảo 51](#_Toc456867641)

# MỞ ĐẦU

## LÝ DO HÌNH THÀNH ĐỀ TÀI

Ngày nay, có một sự thực dễ dàng nhận ra nhận ra: công nghệ đang thay đổi cuộc sống của chúng ta theo từng ngày. Có thể hôm nay bạn làm việc theo cách này, nhưng ngày mai, với những công nghệ mới, những công cụ mới, cách làm việc của bạn sẽ thay đổi theo chúng. Có nhu cầu thì sẽ có cung cấp, khi con người có sự bất tiện trong một hoạt động nào đó trong cuộc sống, những nhà phát triển phần mềm sẽ đưa ra những công cụ hữu ích nhằm đơn giản hóa, tự động hóa các hoạt động mà trước đây phải làm thủ công, giúp người dùng tiết kiệm được thời gian và công sức. Việc đi tìm câu trả lời cho câu hỏi “việc gì đang làm cho mọi người khó chịu?” sẽ đưa ra những ý tưởng tuyệt vời.

Cũng với câu hỏi đó, em đã quan sát và nhận ra việc khiến sinh viên trường Đại học Bình Dương nói riêng và một số trường đại học trong khu vực nói chung không thoải mái là vấn đề thông báo tin tức cho sinh viên. Việc tiếp cận thông tin đối với một sinh viên không khó, nhưng việc tiếp nhận thông tin kịp thời thì không phải lúc nào cũng dễ dàng. Ví dụ khi giảng viên có công việc đột xuất nên phải dời lịch học sang buổi khác, giảng viên thông báo việc này cho nhà trường biết và thông tin đó được đăng lên blog của lớp, nhưng thông tin đó quá gần giờ đi học nên có rất nhiều sinh viên đến trường mới biết. Chuyện đó không phải lỗi của giảng viên hay sinh viên, mà chỉ vì hệ thống thông tin của nhà trường chưa đủ khả năng đưa thông tin tới cho từng sinh viên một cách tức thời. Ngoài ra, một sinh viên phải lên trường để xem về rất nhiều thông tin như: điểm, lịch học, lịch thi…. những việc tuy đơn giản như vậy nhưng tốn rất nhiều thời gian của cả sinh viên lẫn các thầy cô.

Thêm vào đó, các kênh thông tin hiện tại chỉ có thể đưa thông tin được tới mức lớp, có nghĩa là một thông báo chỉ có thể gửi tới cho đơn vị nhỏ nhất là một lớp, không thể thông báo cho từng sinh viên cụ thể. Với vấn đề này, cách giải quyết là tìm một cách thích hợp để người dùng không phải đi tìm thông tin nữa, mà chính thông tin sẽ tìm đến người dùng!

Tuy nhiên, Trong thời gian đi vào phát triển thực tế, em nhận ra rằng: nhu cầu thêm chức năng thông báo cho một hệ thống có sẵn là rất lớn. Ví dụ: một công ty A có một hệ thống quản lí nhân sự, nhưng đôi khi cần thông báo qua điện thoại một tin tức nào đó của công ty. Hoặc trong một trường hợp khác: với một ứng dụng đọc sách, người dùng đăng nhập trên hai thiết bị khác nhau A và B. Họ mua một cuốn sách bằng ứng dụng đó trên thiết bị A. Vậy khi đó, ứng dụng thiết bị B phải được thông báo rằng bạn đã mua cuốn sách đó và ứng dụng sẽ thực hiện tải cuốn sách đó về, để khi người dùng mở ứng dụng trên thiết bị B ra thì cuốn sách đó đã sẵn sàng để được đọc.

Với cách nhìn đó, em hi vọng có thể tạo ra một hệ thống thông báo không chỉ cho một hệ thống xác định nào, mà nó có thể được sử dụng trong bất kì một hệ thống nào muốn có thêm chức năng thông báo như một hệ thống con, có thể gửi được những thông báo có nội dung lớn, code được chia sẻ hoàn toàn miễn phí. Điều này sẽ giúp các dự án phần mềm cần chức năng thông báo được hoàn thiện nhanh hơn rất nhiều.

## MỤC TIÊU ĐỀ TÀI

Mục tiêu của đề tài là: sẽ thiết kế và hiện thực một hệ thống gửi thông báo tự động dưới dạng Notification trong đó có hỗ trợ 1 số tính năng:

* Gửi thông báo tới các thiết bị của một hoặc nhiều người dùng.
* Lưu trữ thông báo.
* Giao tiếp được với tất cả các client được viết bằng các ngôn ngữ khác nhau.

Ngoài ra, đề tài sẽ có một ứng dụng demo, giúp chứng minh tính hữu dụng và ý nghĩa của đề tài.

Hệ thống được thiết kế để sẵn sàng gửi thông báo tới cho các hệ điều hành không phải là Windows như: IOS và Android. Tuy nhiên, do thời gian có hạn nên em không thể làm Demo trên hai hệ điều hành trên được mà sẽ tập trung vào nghiên cứu và thiết kế hệ thống hoạt động trên Windows 10.

## Ý NGHĨA THỰC TIỄN CỦA ĐỀ TÀI

Việc phát triển đề tài này sẽ giúp ích rất nhiều cho các tổ chức, cá nhân có nhu cầu gửi thông báo tới một nhóm người dùng. Giúp giảm thời gian, công sức lẫn chi phí trong quá trình phát triển phần mềm. Đối với người dùng, sẽ không còn việc đi kiểm tra thông tin mới hàng ngày nữa. Thông tin sẽ tự tìm đến với họ. Đó sẽ là một trải nghiệm tuyệt vời.

## PHẠM VI ĐỀ TÀI

Như đã đề cập ở phần trước, hệ thống con này không nhắm tới cụ thể một ứng dụng nào cả. Nó đóng vai trò là một hệ thống con của một hệ thống lớn khác.

Vì vậy, phạm vi của đề tài là:

* Chuyển một đoạn dữ liệu bất kỳ từ server tới một tập hợp các thiết bị.
* Đáp ứng việc ánh xạ 1-1 cơ sử dữ liệu người dùng đã có vào hệ thống con, dùng để phục vụ việc gửi thông báo.
* Tự động quản lý danh sách các thiết bị của người dùng.
* Lưu trữ và chuyển phát nội dung thông báo.

Phạm vi của đề tài không bao gồm:

* Cách đăng nhập người dùng.
* Thông tin của người dùng.
* Cơ sở hạ tầng được yêu cầu.
* Các ứng dụng cho người dùng đầu cuối.
* Mã hóa dữ liệu được gửi đi.

# CƠ SỞ LÝ THUYẾT

## CÁC GIẢI PHÁP

Khi đối mặt với việc phải gửi dữ liệu qua mạng tới một số thiết bị, không ít người sẽ nghĩ tới giải pháp lập trình toàn bộ bằng socket theo giao thức TCP hoặc UDP. Nếu ta sử dụng giao thức UDP, ta sẽ phải lắng nghe server liên tục, đồng thời server sẽ phải gửi liên tục … điều này sẽ hủy hoại tốc độ của hệ thống. Nếu gửi qua giao thức TCP, ta sẽ phải kết nối lại với server sau một khoảng thời gian nào đó để nhận dữ liệu mới. Điều này sẽ làm mất đi tính tức thời mà ứng dụng của chúng ta đa hướng tới. Cả hai giao thức này còn có hai nhược điểm nữa đó là rất khó khăn trong việc bắt lỗi tại Client dẫn đến code thiếu độ đảm bảo, thứ hai là phương pháp này sẽ rất khó khăn gửi thông báo tới cho ứng dụng khi nó không hoạt động, mà việc một ứng dụng không hoạt động là điều hết sức bình thường.

Thực ra thì trên Windows, khi ứng dụng không hoạt động thì vẫn có cách để gửi thông báo tới cho chúng thông qua một tùy chọn có tên là “Socket activity trigger”, nhưng do phải làm việc trên socket nên việc này rất vất vả và phải kiểm soát rất nhiều lỗi trong quá trình liên lạc, dẫn đến khó khăn trong việc nâng cấp lẫn bảo trì. Mặt khác, em không chắc chắn được có thể làm được trên các hệ điều hành khác như android hay iOS hay không.

Và đó cũng là lý do để Microsoft đưa ra một dịch vụ có tên Windows Notification Service (WNS) giúp các nhà phát triển đơn giản hóa việc gửi thông báo cho ứng dụng của họ. Em sẽ trình bày chi tiết về Windows Notification Service, các ưu điểm khi sử dụng giải pháp này trong phần tiếp theo.

## WINDOWS NOTIFICATION SERVICE (WNS)

### WNS là gì

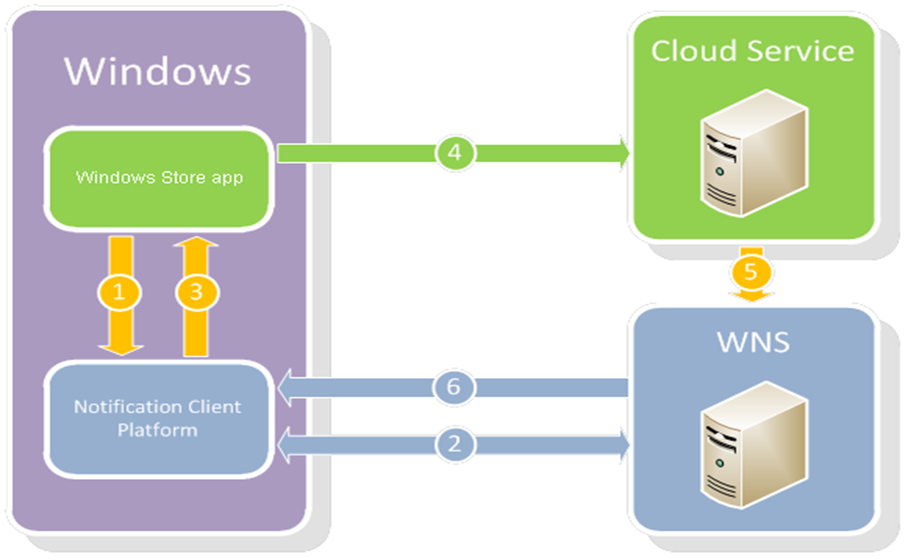
WNS là một dịch vụ đám mây của Microsoft, nó giúp các nhà phát triển phần mềm bên thứ 3 dễ dàng gửi dữ liệu tới ứng dụng của họ từ máy chủ đám mây của nhà phát triển. Tuy gọi là “Máy chủ đám mây của nhà phát triển”, nhưng nó có thể là bất cứ thiết bị nào: từ một main frame, PC hay thậm chí chỉ là một chiếc điện thoại, miễn là có thể giao tiếp với WNS bằng giao thức HTTP. Lý do ta gọi nó là máy chủ đám mây bởi vì với góc nhìn từ WNS, máy chủ của nhà phát triển chỉ là một thiết bị, WNS không biết nó ở đâu và nó là cái gì. Để phân biệt máy chủ đám mây của nhà phát triển với WNS (cũng là một đám mây), ta quy ước gọi máy chủ đám mây của nhà phát triển là Raw Notification Service (RNS) – cũng chính là hệ thống mà đề tài này đang xây dựng.

Từ Windows 8.1, WNS được sinh ra để thay thế cho Microsoft Push Notification Service (MPNS) – một dịch vụ khác có cùng chức năng, được sử dụng trước WNS trên các thiết bị chạy Windows Mobile, Windows Phone 7, Windows Phone 8. Sinh sau thì tất nhiên phải có ưu điểm, dưới đây là các điểm đáng chú ý của WNS so với MPNS:

* Thông báo được gửi trong vòng 5 giây tới thiết bị đang có kết nối internet.
* Có thể được sử dụng cho tất cả các ứng dụng Windows Store.
* Code có khả năng tái sử dụng và chia sẻ cao hơn, dẫn tới chí phí bảo trì thấp hơn.

### WNS làm việc như thế nào?

Sơ đồ dưới đây cho thấy tất cả các luồng dữ liệu liên quan đến việc gửi một Thông báo:



Biểu đồ trên gồm có 6 bước:

* Bước 1: Ứng dụng gửi yêu cầu tới Notification Client Platform (một phần của hệ điều hành windows), xin cấp một kênh thông báo.
* Bước 2: Notification Client Platform yêu cầu WNS tạo một kênh thông báo. Kênh thông báo này sẽ được gửi trả về cho Notification Client Platform trên thiết bị yêu cầu dưới dạng một Uniform Resource Identifier (URI). Một URI này thực ra là một chuỗi đại diện cho một ứng dụng trên một thiết bị.
* Bước 3: URI vừa nhận sẽ được chuyển cho ứng dụng.
* Bước 4: Ứng dụng phải gửi URI này cho RNS để nói cho RNS biết rằng: “khi có gì mới, hãy gửi chúng cho tôi bằng địa chỉ này”.
* Bước 5: Khi RNS có thông tin mới cần gửi, nó sẽ thông báo cho WNS gồm 2 thứ: nội dung thông báo và URI. Điều đó có nghĩa nó muốn nói rằng: “này WNS, tôi cần gửi thông báo này tới máy có URI này”. Và việc còn lại để WNS xử lý.

Với sơ đồ trên, ta có thể hình dung rằng WNS là một bưu điện, ứng dụng sẽ xin địa chỉ (URI) của mình từ bưu điện và gửi cho RNS. Khi có gì cần gửi cho ứng dụng, RNS sẽ gửi thư (Dữ liệu) có địa chỉ nhận mà đã nhận được từ ứng dụng trước đó. Đây là mô hình làm việc của WNS được cung cấp bởi Microsoft. Còn đối với các hệ điều hành khác như IOS và Android cũng có các dịch vụ tương tự như WNS, cách làm việc trên hệ thống gửi thông báo của họ cũng tương tự.

### Các vấn đề bảo mật của mô hình WNS.

Trong 6 bước nêu ở phần 2.2, chỉ có bước 4 là không có sự tham gia của Microsoft. Nhà phát triển phải tự đưa ra một giải pháp để gửi URI về cho RNS. Nếu như ở bước 4, nhà phát triển không sử dụng một kết nối án toàn như HTTPS hoặc mã hóa dữ liệu thì đó sẽ là một miếng mồi ngon cho các kẻ phá hoại. Nếu như bắt được các gói tin này, hacker có thể lấy được thông tin đăng nhập của người dùng, sau đó gửi thông tin URI giả tới cho server.  
Để hiểu rõ hơn, ta hãy xem một đoạn URI mẫu:

|  |
| --- |
| <https://hk2.notify.windows.com/?token=AwYAAAAdEB%2fhqur5%2f0FoscSaKtbes509KizjyqSTpQOa6fsMLbDpZOxO2W4ICrjFCZRAA0f%2f%2b94hAjb7fVywsKKn2c%2fMQberZ4hvBF9PuANK9J%2b1%2b0OmtJwopkFelbcWt%2f33DAM%3d> |

Đó là một địa chỉ có thể được sử dụng bởi phương thức HTTP POST. Vậy RNS sẽ gửi yêu cầu tới WNS theo giao thức HTTP bằng cách POST chúng lên cho server có địa chỉ là “notify.windows.com”. Nếu ta thay đổi URI thành một trang web khác thì các thông báo sẽ được gửi tới một máy chủ khác chứ không phải WNS. Điều này sẽ cực kỳ nguy hiểm nếu ta không kiểm tra trước địa chỉ mà ta cần gửi yêu cầu tới. Hacker có thể lợi dụng điều này nhằm chiếm lấy dữ liệu của người dùng được gửi đi. Vì vậy, mỗi khi nhận URI từ client, ta phải kiểm tra lại xem địa chỉ của URI có đúng là “notify.windows.com” hay không.

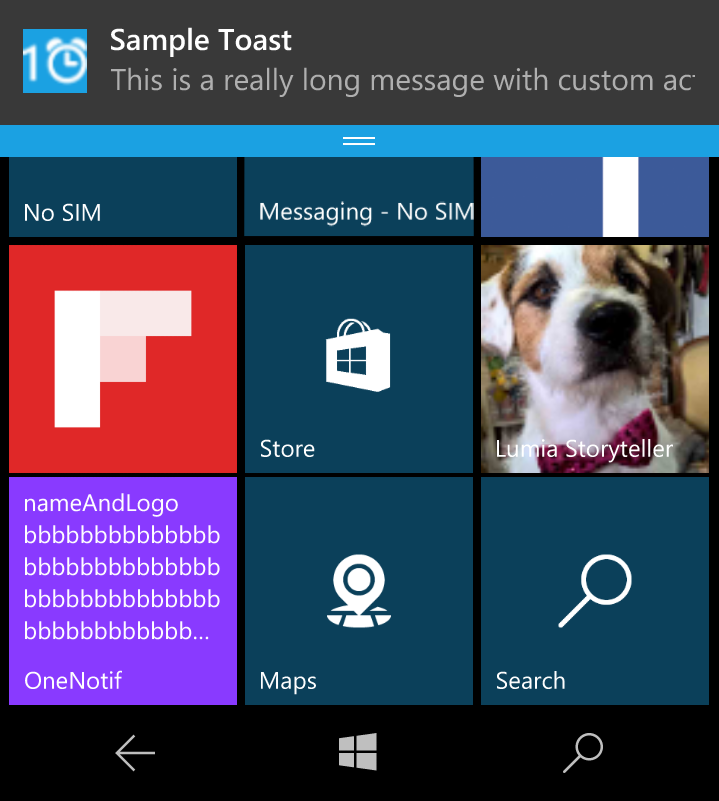
### Làm việc với WNS

Ở phần trước, chúng ta đã biết về 6 bước để WNS hoạt động. Trong phần này, ta sẽ tìm hiểu cách làm việc thực tế với WNS qua những bước đó. Tuy nhiên, một số phần trong 6 phần đó được Microsoft lo nên ta không cần quan tâm nhiều tới khi lập trình. Vì vậy, ta sẽ chỉ đề cập tới những bước mà developer cần làm và có thể chia nhỏ chúng ra để dễ theo dõi.

#### Các loại thông báo trên Windows 10

Trên Windows 10 có nhiều loại thông báo khác nhau như:

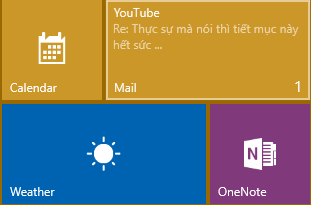
* Toast Notification: thông báo dạng popup lên màn hình.



* Title Notification: hiển thị thông tin lên live title của ứng dụng trong start screen.



* Badge Notification: hiển thị một con số lên live title của ứng dụng trong start screen.



* Raw Notification: đưa dữ liệu dạng thô tới cho ứng dụng, ứng dụng sẽ nhận được dữ liệu dạng chuỗi và phải tự tìm cách làm việc với chuỗi đó.

Trong bốn loại notification trên, nếu RNS muốn gửi một thông báo thuộc một trong 3 loại đầu thì dữ liệu gửi đi bắt bộc phải theo một định dạng nhất định, dữ liệu gửi đi bị hạn chế, và khi nhận được các thông báo đó, Windows sẽ làm việc với dữ liệu của thông báo đó luôn (hiện thông báo toast, cập nhật bage hoặc title). Có nghĩa là ứng dụng sẽ không có quyền gì nhiều trong việc xử lý dữ liệu nhận được từ thông báo. Nhưng với raw notification, bạn có thể gửi một đoạn chuỗi ký tự bất kỳ cho ứng dụng, miễn là nó không vượt quá 5KB. Và điều thú vị ở đây là ta có thể tạo ra toast, title và badge notification bằng code trong chính ứng dụng. Nếu ta có dữ liệu được gửi từ RNS bằng raw notification, ta hoàn toàn có thể tự tạo ra những notification khác theo ý thích. Và hơn thế nữa, ta có thể sử dụng dữ liệu nhận về từ raw notification để làm bất cứ việc gì tùy thích. Điều này khiến cho raw notification trở thành loại thông báo toàn diện nhất.

Như vậy, chúng ta sẽ chỉ tập trung vào raw notification để làm sao gửi được dữ liệu tới cho client là được. Do nó khá toàn diện nên cũng dể hiểu cho việc raw notification là loại notification khó đưa vào ứng dụng nhất, nhưng lợi ích nó đem lại là vượt trội hơn hẳn so với các loại thông báo khác.

#### Bước 1: Xin cấp một kênh thông báo

Ta có thể hình dung một kênh thông báo là một tài khoản của ứng dụng trên WNS. Tài khoản đó bao gồm:

* URI
* Ngày hết hạn của URI

Ứng dụng phải liên hệ với WNS để xin cấp một kênh thông báo. Trong .Net Core, một kênh thông báo được mô hình hóa thành một lớp tên: PushNotificationChanel. Để lấy được một đối tượng PushNotificationChanel, ta sử dụng phương thức tĩnh CreatePushNotificationChannelForApplicationAsync thuộc lớp PushNotificationChannelManager.  
Ta có thể dùng đoạn code đơn giản sau để lấy một kênh thông báo:

PushNotificationChannel channel = null;

try

{

channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();

}

catch (Exception ex)

{

// Không thể tạo một kênh thông báo.

}

Tuy nhiên, mỗi URI bắt buộc phải thuộc một ứng dụng windows store nào đó, nên channel trên tuy vẫn được trả về nhưng sẽ không được sử dụng. Muốn channel lấy về có thể sử dụng được, ta phải liên kết ứng dụng đó với một bản đăng ký trên windows store để nói cho WNS biết ứng dụng nào đang xin notification channel.  
Ta có thể làm theo các bước như sau: Click chuột phải vào tên project => Store => Associate app with store… và làm theo hướng dẫn.

#### Bước 2: Gửi kênh thông báo cho RNS

Một đối tượng PushNotificationChanel chứa một URI. Nên nếu tạo thành công, ứng dụng có thể gửi URI đó tới cho RNS để lưu vào cơ sở dữ liệu. Tuy nhiên, URI này cũng giống như một “Token”, nó có giới hạn về thời gian tồn tại. Theo các tài liệu của Microsoft nói thì URI sẽ tồn tại tối đa là 30 ngày. Tuy nhiên URI có thể hết hạn sớm hơn dựa vào rất nhiều yếu tố khác nhau do WNS quyết định. Bởi vậy, ta cần yêu cầu cấp một URI mới mỗi khi ứng dụng được mở và gửi chúng cho RNS. Ứng dụng cũng nên có một tác vụ nền có nhiệm vụ lấy URI sau một khoảng thời gian nhất định.

Ngoài ra, RNS phải được thiết kế sao cho khi ứng dụng gửi một URI mới lên thì RNS phải biết URI đó là của thiết bị nào để rồi Update lại URI chứ không phải tạo một bản ghi mới. Để làm được điều đó, ứng dụng phải gửi cả mã máy, hệ điều hành cho RNS. Trong bước này không có sự tham gia của Microsoft, cho nên ta có thể sử dụng bất kì phương thức gửi dữ liệu nào mà chúng ta muốn.

#### Bước 3: Xác thực danh tính với WNS

Để có thể nhờ WNS gửi thông báo tới một ứng dụng, trước hết, RNS phải xác thực với WNS rằng nó là máy chủ của ứng dụng đó. Dưới đây là mô hình xác thực danh tính:



Theo hình trên, ta có thể thấy cách làm việc của WNS:

* Gửi Package SID cà Secret Key cho WNS
* WNS trả về một chuỗi ký tự được gọi là Token
* RNS lưu Token lại
* Gửi nội dung thông báo có đính kèm Token cho WNS
* WNS gửi trả về kết quả của việc gửi đi.

Package SID và Secret Key là hai chuỗi được cấp khi đăng ký một app trên windows store dùng để xác thực rằng: RNS là chủ của ứng dụng đó. Chúng có thể được lấy về trên trang của ứng dụng đó tại <https://dev.windows.com/>. Nếu để lộ Secret ID với một ai khác, người đó có thể dùng nó để giả mạo RNS và gửi các thông báo rác tới ứng dụng. Tất cả sự giao tiếp với WNS đều được thực hiện trên giao thức HTTPS và chỉ chấp nhận phương thức PUSH. Vì vậy, để gửi Package SID và Secret ID, ta phải gửi chúng bằng một HTTP request có method là PUSH.   
Định dạng http request của gói tin yêu cầu token tới WNS như sau:

POST /accesstoken.srf HTTP/1.1

Content-Type: application/x-www-form-urlencoded  
Host: <https://login.live.com>  
Content-Length: <Length>  
grant\_type=client\_credentials&client\_id=<Package SID>&client\_secret=<Client Secret>&scope=notify.windows.com

Trong đó <Package SID> và <Client Secret> là hai tham số cần bổ sung, Length là độ dài của request.

Ta có thể dùng đoạn Code ngắn sau để lấy token từ WNS:

[DataContract]

public class OAuthToken

{

[DataMember(Name = "access\_token")]

public string AccessToken { get; set;}

[DataMember (Name = "token\_type")]

public string TokenType {get; set; }

}

private OAuthToken GetOAuthTokenFromJson (string jsonString)

{

using (var ms = new MemoryStream(Encoding.Unicode.GetBytes(jsonString))) {

var ser = new DataContractJsonSerializer(typeof(OAuthToken));

var oAuthToken = (OAuthToken)ser.ReadObject(ms);

return oAuthToken;

}

}

protected OAuthToken GetAccessToken (string secret, string sid)

{

var urlEncodedSecret = HttpUtility.UrlEncode(secret);

var urlEncodedSid = HttpUtility.UrlEncode(sid);

var body = String.Format("grant\_type=client\_credentials&client\_id={0}&client\_secret={1}&scope=notify.windows.com", urlEncodedSid, urlEncodedSecret);

string response;

using (var client = new WebClient())

{

client.Headers.Add("Content-Type",

"application/x-www-form-urlencoded");

response = client.UploadString("https://login.live.com/accesstoken.srf", body);

}

return GetOAuthTokenFromJson(response);

}

Sau khi có được token, RNS sẽ sẵn sàng để gửi thông báo tới cho WNS. Tuy nhiên, mỗi token đều có thời hạn sử dụng. Một khi hết hạn, RNS phải xin cấp một token mới thì mới gửi thông báo tiếp được.

#### Bước 4: Gửi thông báo cho WNS

Về cơ bản, URI nhận được từ ứng dụng chính là một đường link trên một web server. Vậy, để gửi thông báo tới cho URI đó, về cơ bản là RNS sẽ truy cập đường link đó bằng phương thức PUSH và gửi kèm theo token cùng dữ liệu của thông báo.

Như đã đề cập ở phần trước, ta chỉ quan tâm đến raw notification. Raw notification không quy định về cấu trúc cho phần nội dung thông báo, ta có thể gửi bất cứ thứ gì trong thông báo đó, sao cho tổng dung lượng không quá 5KB.  
Dưới đây định dạng một đoạn HTTP request dùng để gửi notification:

POST  
https://cloud.notify.windows.com/?token=AQE%bU%2fSjZOCvRjjpILow%3d%3d HTTP/1.1  
Content-Type: application/octet-stream  
X-WNS-Type: wns/raw  
X-WNS-Cache-Policy: cache  
X-WNS-RequestForStatus: true  
Authorization: Bearer <Token>  
Host: cloud.notify.windows.com  
Content-Length: <Length>

<body>  
</body>

Trong đó, <Token> là thẻ truy cập, <Length> là độ dài của gói tin.

Dưới đây là một số header quan trọng trong một gói tin gửi cho WNS:

|  |  |  |
| --- | --- | --- |
| **Header name** | **Required/Optional** | **Description** |
| Authorization | Required | Một header chuẩn của giao thức HTTP, dùng để xác thực danh tính của RNS. Bạn sẽ phải cung cấp thẻ truy cập trong header này. |
| Content-Type | Required | Một header chuẩn của giao thức HTTP, nếu sử dụng raw notification, header này phải có giá trị là: application/octet-stream |
| Content-Length | Required | Header này giúp xác định kích thước của gói tin. |
| X-WNS-Type | Required | Giúp xác định loại thông báo là title, toast, badge hay raw. nếu ta sử dụng raw notification, header này phải có giá trị là wns/raw |
| X-WNS-Cache-Policy | Optional | Nếu sử dụng header này, WNS sẽ lưu lại thông báo và gửi đi sau nếu như thiết bị offline. Giá trị nên đặt là “cache”. |
| X-WNS-RequestForStatus | Optional | Nếu sử dụng header này, response từ WNS sẽ chứa các thông tin liên quan đến trạng thái kết nối của thiết bị và tình trạng của thông báo. mặc định là false, ta nên đặt là true để nhận được các thông tin này. |

#### Bước 5: Nhận thông tin trả về từ WNS sau khi gửi thông báo

Nếu trong request gửi thông báo, header X-WNS-RequestForStatus được đặt giá trị là true thì sau khi gửi thông báo, WNS sẽ gửi trả lại cho RNS một số thông tin liên quan tới trạng thái hiện tại của thiết bị và của thông báo. Ta có thể sử dụng thông tin này để xem xét, liệu thiết bị có thể còn nhận được thông báo nữa hay không.

Dưới đây là một số header trong gói tin trả về từ WNS:

|  |  |  |
| --- | --- | --- |
| **Header name** | **Required/Optional** | **Description** |
| X-WNS-Debug-Trace | Optional | Một số thông tin dùng dể debug. |
| X-WNS-DeviceConnectionStatus | Optional | Chứa trạng thái kết nối của thiết bị. Header này chỉ có nếu gói tin gửi đi có dùng header X-WNS-RequestForStatus |
| X-WNS-Error-Description | Optional | Một chuỗi kí tự mô tả lỗi nếu có |
| X-WNS-Msg-ID | Optional | Chuỗi kí tự chứa ID của thông báo |
| X-WNS-Status | Optional | Chứa trạng thái của thông báo |

Chi tiết về một số header cần quan tâm trong số trên:

* X-WNS-DeviceConnectionStatus: header này có thể có 3 giá trị trong bảng sau:

|  |  |
| --- | --- |
| **Value** | **Description** |
| Connected | Thiết bị đang có kết nối tới WNS. Thông báo sẽ được chuyển tới thiết bị ngay lập tức. |
| Disconnected | Thiết bị không có kết nối tới WNS. |
| Tempconnected | Thiết bị đang tạm thời mất kết nối tới WNS, ví dụ như khi kết nối 3g tự động ngắt để tiết kiệm dung lượng hoặc kết nối Wireless bị rớt. Đây là sự gián đoạn mạng tạm thời chứ không phải là cố ý ngắt kết nối. |

* X-WNS-Status: header này có thể có 3 giá trị trong bảng sau:

|  |  |
| --- | --- |
| **Value** | **Description** |
| Received | Thông báo đã được nhận và sẽ tiếp tục được xử lí để đưa tới thiết bị. Tuy nhiên, điều này không đảm bảo rằng thiết bị đã nhận được thông báo. |
| Dropped | Thông báo đã bị hủy do lỗi hoặc máy client đã từ chối nhận những thông báo này. |
| Channelthrottled | Thông báo đã bị hủy do RNS vượt quá giới hạn tốc độ gửi thông báo cho phép cho kênh này. |

Ngoài các header, một gói tin trả về của HTTP luôn có status code. Dựa vào mã này, ta có thể có được những thông tin rất hữu ích:

|  |  |  |
| --- | --- | --- |
| **HTTP response code** | **Description** | **Recommended action** |
| 200 OK | Thông báo hợp lệ và đã được WNS chấp nhận. |  |
| 400 Bad Request | Lỗi do một vài header không đúng cú pháp hoặc khai báo các header lặp lại. | Xem lại các header và sửa lỗi. |
| 401 Unauthorized | Thẻ truy cập không chính xác hoặc đã hết hạn. | Xin cấp một thẻ truy cập mới rồi thử gửi lại thông báo với thẻ truy cập mới đó. |
| 403 Forbidden | RNS không được phép gửi thông báo tới cho URI này. | Lỗi này do thẻ truy cập được sử dụng không trùng với thẻ truy cập của URI tương ứng. Trường hợp này nên kiểm tra lại Application identity trong File Package.appxmanifest |
| 404 Not Found | URI không tồn tại. | URI này không đúng và hoặc không được công nhận bởi WNS. Hãy xóa thiết bị này khỏi cơ sở dữ liệu |
| 405 Method Not Allowed | Phương thức gửi HTTP được sử dụng không chính xác. | Chỉ có phương thức POST được chấp nhận. Hãy kiểm tra lại phương thức đang sử dụng để gửi gói tin HTTP. |
| 406 Not Acceptable | RNS gửi quá nhiều thông báo tới URI này trong một khoảng thời gian ngắn. | Tăng thời gian nghỉ giữa các lần gửi và gửi lại. |
| 410 Gone | URI hết hạn. | Do URI sẽ hết hạn sau 30 ngày. Nếu thiết bị không hoạt động trong 30 ngày, URI sẽ không được làm mới. Nên xóa thiết bị này khỏi cơ sở dữ liệu. |
| 413 Request Entity Too Large | Thông báo có kích thước quá lớn. | Mỗi thông báo raw chỉ được chứa tối đa 5000Byte dữ liệu. Hãy giảm kích thước của thông báo và gửi lại. |
| 500 Internal Server Error | Một lỗi gây ra bởi mạng nội bộ của WNS. | Đây là lỗi từ phía WNS, hãy gửi lỗi này lên MSDN để Microsoft khắc phục. |
| 503 Service Unavailable | Dịch vụ WNS không khả dụng. | Đây là lỗi từ phía WNS, hãy gửi lỗi này lên MSDN để Microsoft khắc phục. |

#### Bước 6: Nhận thông báo ở client

##### Mô hình chung cho việc nhận thông báo của ứng dụng

Sau khi thông báo được WNS nhận từ RNS, nó sẽ được chuyển tới cho ứng dụng. Ta có hai cách để nhận được thông báo này:

* Qua sự kiện nhận được thông báo (chỉ hoạt động khi ứng dụng đang chạy)
* Qua tác vụ nền (hoạt động ngay cả khi ứng dụng bị tắt)

Ứng dụng có thể sử dụng cả hai cách trên để nhận thông báo. Nếu ứng dụng đang được chạy thì thông báo sẽ được nhận ở sự kiện trước, sau đó được chuyển xuống tác vụ nền sau. Còn khi ứng dụng bị tắt, chỉ có tác vụ nền là nhận được thông báo. Ngoài ra, nếu trong sự kiện nhận thông báo, ta thay đổi giá trị Cancel của đối tượng EventArgs thành true, thông báo sẽ không được gửi xuống cho background task.

Ta có thể hình dung cách hoạt động của chúng như sơ đồ dưới đây:



##### Background task

Người lập trình Windows desktop chắc không còn lạ lẫm gì với các tiến trình chạy trên hệ thống mà không có giao diện người dùng, chúng ta thường gọi đó là các tác vụ chạy nền (Background task). Các tác vụ nền này vẫn được nhìn thấy trên cửa sổ Task manager, và nó luôn được khởi chạy tại một thời điểm xác định.

Trên UWP, các background task khác rất xa so với trên windows desktop. Ta có thể hình dung rằng: một background task trên UWP là một đoạn code. Ứng dụng phải đăng ký với hệ thống rằng đoạn code đó chính là một background task. Khi đăng ký một background task trên UWP, ứng dụng phải khai báo với hệ thống các thông tin và mục đích sử dụng background task đó. Đồng thời, ứng dụng cũng phải khai báo khi nào và hệ thống ở trạng thái nào thì background task mới được chạy.  
Ví dụ: một ứng dụng có thể muốn chạy một background task mỗi khi người dùng không sử dụng máy, đồng thời đang có internet.  
Một khi tất cả các điều kiện được thỏa mãn, hệ thống sẽ khởi động background task. Và background task này có 25 giây để hoàn thành công việc. Nếu quá 25 giây thì nó sẽ bị tắt. Do thời gia có hạn và việc trình bày làm cách nào để đăng ký một background task trên UWP có khá nhiều demo trên internet, em sẽ không đi chi tiết thên cho phần này nữa.

##### Lựa chọn sử dụng giữa hai phương thức nhận thông báo

Như đã nói ở phần trước, ứng dụng sẽ luôn nhận được thông báo ở background task, cho dù ứng dụng có đang chạy hay không. Vậy tại sao Microsoft lại đưa cho lập trình viên một cách nữa đó là nhận thông báo khi ứng dụng đang chạy?

Lý do là: nếu ta nhận thông báo ở background, ta khó có thể cập nhật giao diện được. Ví dụ người dùng đang sử dụng ứng dụng và có một thông báo tới. Lúc đó, ta cần cập nhật lại giao diện, ví dụ như hiện một con số hoặc cập nhật thêm thông báo vào danh sách thông báo đang hiển thị trên giao diện người dùng chẳng hạn. Việc đó sẽ rất dễ dàng nếu như được thực hiện trên tiến trình chứa giao diện, còn nếu thực hiện bằng background task thì rất khó khăn.

## CÁC GIẢI PHÁP, CÔNG NGHỆ, PATTERM KHÁC ĐƯỢC SỬ DỤNG

Phần này, em sẽ trình bày, giới thiệu một số giải pháp công nghệ và các patterm được sử dụng trong đề tài. Tất cả chúng đều rất nổi tiếng và qua thời gian, chúng đã được chứng minh là những công nghệ/ giải pháp khá toàn diện, giúp giải quyết các vấn đề kinh điển trong ngành công nghệ phần mềm.

### Windows communication foundation (WCF)

#### Khái niệm

WCF là một framework được dùng cho việc xây dựng các ứng dụng hướng dịch vụ (Service Oriented Architecture – SOA). Mục đích của WCF là tạo một công nghệ thống nhất cho các công nghệ giao tiếp trước đây như: Web Services, .NET Remoting, Microsoft Message Queuing và COM+.

Ba thành phần chính của một WCF service là:

* **Service class**: định nghĩa các contact (em sẽ trình bày ở bên dưới).
* **Hosting environment**:  WCF service có thể được host lên nhiều “môi trường” khác nhau, bao gồm IIS, Windows service, Selt-hosting.
* **End point**: Các “cổng” kết nối giữa client và service.

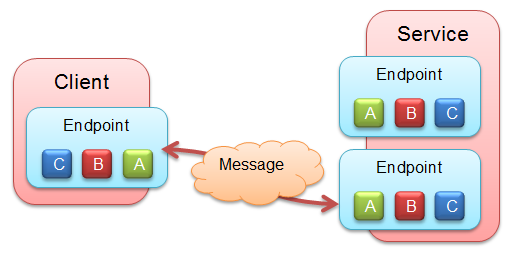
#### Mô hình kết nối ABC

Việc giao tiếp giữa client và WCF service được thực hiện thông qua các Endpoint. Như vậy một service có thể làm chấp nhận và xử lý nhiều yêu cầu khác nhau thông các endpoint riêng biệt. Mỗi endpoint được tạo thành từ ba thành phần với tên gọi tắt rất dễ nhớ là ABC. Trong đó:

* **A – Address (Where):** địa chỉ của service.
* **B – Binding (How):** Cách thức giao tiếp với service. Thành phần này xác định loại giao thức kết nối giữa client và service (như HTTP, TCP, MSMQ,…), kênh xử lý và kiểu mã hóa thông điệp. Danh sách các kiểu binding:

|  |  |
| --- | --- |
| Name | Purpose |
| BasicHttpBinding | Interoperability with Web services and clients supporting the WS-BasicProfile 1.1 and Basic Security Profile 1.0. |
| WSHttpBinding | Interoperability with Web services and clients that support the WS-\* protocols over HTTP. |
| WSDualHttpBinding | Duplex HTTP communication, by which the receiver of an initial message does not reply directly to the initial sender, but may transmit any number of responses over a period of time by using HTTP in conformity with WS-\* protocols. |
| WSFederationBinding | HTTP communication, in which access to the resources of a service can be controlled based on credentials issued by an explicitly-identified credential provider. |
| NetTcpBinding | Secure, reliable, high-performance communication between WCF software entities across a network. |
| NetNamedPipeBinding | Secure, reliable, high-performance communication between WCF software entities on the same machine. |
| NetMsmqBinding | Communication between WCF software entities by using MSMQ. |
| MsmqIntegrationBinding | Communication between a WCF software entity and another software entity by using MSMQ. |
| NetPeerTcpBinding | Communication between WCF software entities by using Windows Peer-to-Peer Networking. |

* **C – Contact (What)**: Bản mô tả các chức năng của service. Cụ thể, đây là các class (structure, enum,…) được định nghĩa bên service để với các phương thức mà client có thể yêu cầu service thực hiện.

[](https://yinyangit.wordpress.com/2012/03/08/wcf-mot-so-khai-niem-co-ban/wcf-endpoint-architecture-abc/)

#### Contract

Contract được coi là một bản “hợp đồng” quy định cách thức và phương tiện để hai bên có thể hợp tác và làm việc với nhau. Trong WCF, các contact được coi như những bản mô tả nhiệm vụ, kiểu dữ liệu của service. Client sẽ có được các contract sau khi tham chiếu đến service và thông qua đó để gửi và nhận các thông điệp. Như vậy, service và client không sử dụng cùng các kiểu dữ liệu như nhau, mà chỉ sử dụng chung các contact.

* **Service Contract**

Mô tả các công việc của service, bao gồm các kiểu dữ liệu truyền đi và kiểu dữ liệu trả về. Cụ thể, đây là các khai báo phương thức kèm với các tham số và kiểu trả về của service. Loại contract này có thể được chia ra làm hai phần là service contract (tương ứng với các interface) và operation contract (là các method).

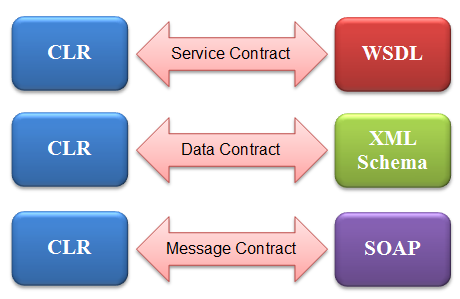
* **Data Contract**

“A data contract is a formal agreement between a service and a client that abstractly describes the data to be exchanged. That is, to communicate, the client and the service do not have to share the same types, only the same data contracts. A data contract precisely defines, for each parameter or return type, what data is serialized (turned into XML) to be exchanged.”  (MSDN)

Mô tả các kiểu dữ liệu tự tạo mà client có thể sử dụng. Các contract này dùng để chứa các dữ liệu cần thiết để truyền tải giữa client và service. Nếu service chỉ sử dụng các kiểu dữ liệu đơn giản (như int, string, DateTime,…) thì service đó không cần đến Data Contract.

* **Message Contract**

Tương tự như Data Contract nhưng kiểu contract này nhằm mục đích định nghĩa cấu trúc của các thông điệp được truyền tải dưới dạng SOAP (Simple Object Access Protocol). Một thông điệp SOAP được chia thành 2 phần là header và body.

[](https://yinyangit.wordpress.com/2012/03/08/wcf-mot-so-khai-niem-co-ban/relationship-of-contracts-to-the-common-language-runtime/)

* **Fault Contract**

Loại contact này giúp quản lý các ngoại lệ mà service ném ra khi thực hiện các yêu cầu của client. Khi cần thiết, service sẽ ném ra một FaultException nếu như muốn client bắt được ngoại lệ này.

### Dapper .Net

#### ORM

Như đã đề cập đến trong phần danh sách chữ viết tắt, ORM là một kỹ thuật chuyển đổi dữ liệu không hướng đối tượng thành dữ liệu hướng đối tượng. Thường được dùng trong mapping data khi đọc từ DB lên. Tuy nhiên, với sự xuất hiện của các ORM lớn như Linq to SQL và Entity Framework, Hibernate, … các ORM thường được nghĩ đến không chỉ có chức năng mapping data nữa, nó còn quản lý transaction, connection, ràng buộc dữ liệu, thông dịch từ code Linq sang sql querry… Những hỗ trợ đó giúp ích rất nhiều cho thời gian phát triển các ứng dụng. Đồng thời đem lại những đoạn code ngắn gọn và dễ bảo trì. Nếu là các ứng dụng nhỏ thì việc dùng các ORM là rất tốt.

Tuy nhiên, hiện đại thì luôn hại điện. Các ORM đó có hiệu suất phải nói là rất tồi tệ. Với những ứng dụng lớn và yêu cầu độ trễ thấp hoặc có tần suất làm việc nhiều thì việc sử dụng một ORM là một điều không thể chấp nhận được. Thay vào đó, giải pháp được sử dụng thay thế có tên Mini ORM xuất hiện.

#### Mini ORM

Thực ra, Mini ORM là một sự lược bỏ bớt của ORM. Nhiệm vụ của Mini ORM rất đơn giản: đọc dữ liệu từ DB và map kết quả vào các đối tượng trên code. Rõ ràng với sự đơn giản đó thì Mini ORM trở nên cực kỳ mạnh mẽ. Tuy nhiên, code chúng ta phải viết khi làm việc với Mini ORM cũng sẽ nhiều hơn ORM.

Thực chất, các tác vụ cơ bản như insert, update, delete, và connection, transaction cũng không có trên Mini ORM, ta phải tự làm thủ công. Ta có thể nhận ra một điều: code tự sinh ra càng nhiều thì hiệu suất càng giảm.

#### Dapper

Nổi lên giữa các Mini ORM, Dapper .net là một giải pháp có tính cân bằng giữa số code được viết sẵn và hiệu suất. Nhà phát triển không phải viết quá nhiều code (số lượng code nằm ở mức chấp nhận được) mà hiệu suất thì vẫn tuyệt vời. Lý do là Dapper .Net có một extension có tên Dapper Extension. Nó sẽ lo việc Insert, delete, update cho chúng ta.

Dưới đây là bảng so sánh hiệu suất giữa một số ORM và Mini ORM phổ biến:

Có thể thấy Dapper có tốc độ rất nhanh, chỉ đứng sau viết code thủ công (hand coded), có nghĩa là Dapper .Net là Mini ORM nhanh nhất trong các Mini ORM cũng như ORM.

Trong khuôn khổ giới thiệu sơ lược về Dapper .Net, em không muốn đi sâu hơn về chi tiết cách làm việc. Có thể vào trang của Dapper .Net trên GitHub để biết thêm chi tiết.  
Dapper .Net on GitHub:

<https://github.com/StackExchange/dapper-dot-net>

### Mô hình nhiều lớp

Để dễ quản lý các thành phần của hệ thống, cũng như không bị ảnh hưởng bởi các thay đổi, người ta hay nhóm các thành phần có cùng chức năng lại với nhau và phân chia trách nhiệm cho từng nhóm để công việc không bị chồng chéo và ảnh hưởng lẫn nhau.

Trước khi đến với các khái niệm về mô hình này, ta phải làm rõ thuật ngữ “Lớp”. lớp trong tiếng anh được có hai cách viết “layer” và “tier”. Và có rất nhiều người không phân biệt được hai từ này, dẫn đến việc dùng cả hai: lúc thì gọi là layer, lúc thì gọi là tier, trong khi chúng khá khác nhau.

Hầu hết các chuyên gia nhận ra một sự khác biệt giữa hai từ này:

* Tier dùng để chỉ các tầng vật lý trong cơ sở hạ tầng của một hệ thống.
* Layer dùng để chỉ các tầng ở mức khái niệm, giúp tạo nên giải pháp.

Ví dụ: một phần mềm được thiết kế theo mô hình ba lớp (layer) có thể được cài đặt dễ dàng trong một lớp (tiers) của doanh nghiệp, ví dụ như trên máy tính cá nhân chẳng hạn.

#### Mô hình 3 lớp

Có lẽ phần nhiều các kỹ sư phần mềm đều đã biết đến mô hình ba lớp. Mô hình ba lớp được cấu thành từ:

* Presentation Layers: Chứa giao diện người dùng.
* Business Logic Layers: Chứa các xử lý logic (Business Logic), được sử dụng bởi Presentation Layers.
* Data Access Layers: Lớp này chứa các nghiệp vụ liên quan đến lưu trữ và truy xuất dữ liệu của ứng dụng như đọc, lưu, cập nhật cơ sở dữ liệu. Chỉ lớp này mới biết đến việc dữ liệu ở đâu, làm sao thao tác với dữ liệu.

Theo cách phân chia trên, các thành phần của ứng dụng sẽ rất rõ ràng trong công việc của chúng. Nếu một lỗi phát sinh, ta có thể kiểm tra và tìm ra lỗi một cách dễ dàng.

Tuy nhiên, mô hình trên chỉ có thể áp dụng cho những ứng dụng nhỏ, nơi tất cả các thứ được cài đặt trên một máy tính chẳng hạn. Với những ứng dụng doanh nghiệp có nhiều người dùng, các thành phần được cài đặt trên nhiều máy khác nhau giúp tăng hiệu suất hoặc đơn giản chỉ vì nó có nhiều thành phần đặc biệt, ba lớp chắc chắn là không đủ.

#### Mô hình nhiều lớp

Mô hình nhiều lớp là một mô hình thiết kế phần mềm mà sử dụng nhiều lớp, mỗi lớp có một nhiệm vụ cụ thể. Cũng giống như mô hình ba lớp, một ứng dụng mô hình nhiều lớp cũng thường có những layer như: Presentation, Business Logic và Data Access. Tuy nhiên, một ưng dụng mô hình nhiều lớp thường có thêm một lớp nữa là Appplication Layer hay còn được gọi là Service Layer. Tầng này cung cấp các dịch vụ trung gian để các thành phần có thể giao tiếp với nhau thông qua mạng. Một ví dụ về việc sử dụng Service Layer là một hệ thống client – server. Client là ứng dụng desktop chỉ chứa UI, còn server sẽ chứa toàn bộ các business logic, data. Vậy cách để ứng dụng desktop liên lạc với server để thực thi logic là thông qua Service Layer. Có rất nhiều cách để cài đặt Service Layer: socket, web socket, webservice, COM+… nhưng có lẽ cách được sử dụng rộng rãi nhất hiện nay trong các ứng dụng doanhh nghiệp là sử dụng WCF.

Ngoài ra, mô hình nhiều lớp còn chứa một số lớp khác như: Business Infrastructure. Lớp này giống như một thư viện chứa các xử lí cho những công việc rất phổ biến và nó thường nhắm vào một lĩnh vực chứ không nhắm vào một ứng dụng cụ thể nào cả, do đó tính sử dụng lại cho nhiêu ứng dụng khác của tầng này là rất lớn, ví dụ như Business Infrastructure cho chuyển đổi tiền tệ chẳng hạn. Còn tầng Business Logic là tầng logic cho một ứng dụng cụ thể. Business Infrastructure nằm ở giữa Business và Data Access.

# THIẾT KẾ TỔNG QUÁT RNS TRÊN NỀN TẢNG .NET FRAMEWORK

## MỤC TIÊU MÀ BẢN THIẾT KẾ HƯỚNG ĐẾN

Mục tiêu mà bản thiết kế hệ thống này cần đạt đến là:

* Đơn giản, dễ tùy chỉnh.
* Có thể cài đặt nhanh chóng và tương thích với mọi hệ thống có sẵn.
* Độ tin cậy cao.
* Đảm bảo tính toàn vẹn dữ liệu.
* Tốc độ hồi đáp cao cho server, tiết kiệm năng lượng cho thiết bị đầu cuối.

## BẢN THIẾT KẾ TỔNG QUÁT CÁC THÀNH PHẦN CỦA HỆ THỐNG

Dưới đây là sơ đồ tổng quát các thành phần của hệ thống và cách chúng tương tác với nhau. Do đây chỉ mới là sơ đồ tổng quát nên sẽ có một vài chi tiết nhỏ như lưu xuống DB… đã bị lược bỏ bớt.



Như sơ đồ trên, ta có thể liệt kê các bước để gửi thông báo:

* Bước 1: AES gửi một yêu cầu gửi thông báo tới cho RNS.
* Bước 2: RNS liên hệ với dịch vụ gửi thông báo của nhà phát triển hệ điều hành tương ứng rồi nhờ dịch vụ đó gửi ID của thông báo.
* Bước 3: RN Adapter là một gói thư viện nhỏ được đi chung với Mobile applcication, giúp Mobile application nhận thông báo và quản lý thông báo dễ dàng hơn. RN Adapter sẽ nhận ID thông báo và preview data trực tiếp từ dịch vụ gửi thông báo của nhà phát triển hệ điều hành.
* Bước 4: RN Adapter thông báo tới Mobile application rằng có thông báo mới.
* Bước 5: Mobile application yêu cầu RN adapter cung cấp nội dung thực sự của thông báo.
* Bước 6: RN Adapter liên hệ trực tiếp với RNS để lấy thông báo dựa trên ID nhận được.
* Bước 7: RNS trả về nội dung của thông báo
* Bước 8: RN Adapter trả về thông báo cho Mobile application.

Lưu ý: Mobile Aplication, RN Adapter và AES có thể được viết bằng bất cứ ngôn ngữ nào và chạy trên bất kỳ nền tảng nào. Vì vậy RNS phải dùng một phương thức mà có thể giao tiếp được với chúng.

Giải pháp đơn giản nhất chính là sử dụng WCF service (đã trình bày ở chương 2). Với việc sử dụng WCF service, tất cả dữ liệu sẽ được chuyển xuống dạng văn bản xml. Khi client nhận được, đoạn xml đó sẽ được chuyển về CLR (Common Language Runtime).

## SƠ ĐỒ STATE CỦA NOTIFICATION DATA

Rõ ràng, khi thiết kế RNS, ta phải xác định rằng: nội dung của thông báo có thể là bất cư thứ gì có thể lưu trữ được. Và cách đơn giản nhất để lưu trữ, đọc và gửi đi “bất cứ thứ gì” là chuyển chúng về dạng mảng byte. Có nghĩa là RNS sẽ không quan tâm đến nội dung của thông báo là gì, nó là loại đối tượng gì, RNS chỉ biết đó là một mảng byte. Nó sẽ giao chính xác đoạn byte đó tới ứng dụng. Còn việc chế biến làm sao thành dạng mảng byte và làm sao chuyển từ mảng byte thành dữ liệu thì RNS không quan chịu trách nhiệm. Tuy nhiên, không chịu trách nhiệm không có nghĩa là không làm gì. RNS vẫn sẽ cung cấp cho người dùng một encoder dưới dạng một thư viện đi kèm RNS.

Trong thế giới lập trình, “bất cứ thứ gì” chính là **Object**. Vậy, câu hỏi được gói gọn lại là “làm thế nào để chuyển một object thành dạng mảng byte?”.  
Thực ra, câu trả lời đã có ngay trong một công nghệ mà ta đang sử dụng: WCF.

WCF cũng làm được việc chuyển Object (CLR type) về dạng mảng byte để gửi đi qua môi trường mạng. Và cách nó làm như sau:

1. Chuyển CLR type về dạng chuỗi XML.
2. Chuyển chuỗi XML về dạng mảng byte.
3. Gửi mảng byte từ server.
4. Nhận về mảng byte ở client.
5. Đưa mảng byte về dạng chuỗi XML.
6. Chuyển chuỗi XML về CLR type.

Tuy nhiên, việc dùng XML không tốt bằng JSON, do dữ liệu được biểu diễn dưới dạng XML có kích thước lớn hơn JSON cũng khư khó đọc hiểu trực tiếp hơn. Do đó, ta sẽ chuyển CLR type về dạng chuỗi JSON thay vì XML. Các bước còn lại giống như WCF làm.



Lý do RNS không ép buộc AES phải encode theo cách này là nó không nhanh, có nhiều cách có hiệu suất cao hơn, nhưng các cách đó không toàn diện. Do sự thiếu hoàn hảo nên em quyết định để cho nhà phát triển lựa chọn cách encode, nếu họ không biết cách làm hoặc muốn sử dụng cách của em thì có thể dùng cách này.

## BẢN THIẾT KẾ MÔ HÌNH AUTHENTICATION VÀ AUTHORIZATION CHO RNS

Dù việc thiết kế mô hình gửi thông báo cho Mobile App đã xong, tuy nhiên RNS chưa định nghĩa việc những ai được phép đọc thông báo đó. Nếu một người lạ chỉ có ID của thông báo mà liên hệ với RNS để lấy nội dung của thông báo thì sao? Lỡ có ai đó dò thông báo dựa trên ID thì sao?

Dưới đây là mô hình authentication của RNS cho việc gửi URI tới RNS:



Như ta đã biết, có hàng chục kiểu đăng nhập khác nhau. Từ phương thức đăng nhập bằng tài khoản - mật khẩu là hai chuỗi ký tự, ta có mã PIN, vân tay, nhận dạng khuôn mặt (như hello windows trên windows 10), nhận dạng bằng mắt … và có lẽ sẽ còn nhiều hình thức đăng nhập khác đang và sẽ được sử dụng nữa. Vì vậy, RNS không thể ép buộc AES phải theo một loại đăng nhập nào đó, mà AES phải tự đưa ra định nghĩa cho việc đăng nhập. Để làm điều đó, cách đơn giản nhất là sử dụng token. Token này cũng tương tự như token mà ta nhận được từ WNS trong quá trình authentication với nó.  
Ý tưởng là Mobile Application sẽ liên hệ với một service login. Service này có nhiệm vụ xác thực danh tính của user (đây không phải một phần của đề tài, hệ thống sử dụng RNS phải tự làm phần này).

Dưới đây là các bước của quá trình đăng nhập:

1. Mobile Client gửi yêu cầu đăng nhập tới cho Login service.
2. Login service xác thực việc đăng nhập, sau đó gửi ID của user qua cho RNS.
3. RNS nhận được ID của user từ Login Service (dạng chuỗi ký tự). Nó sẽ tạo ra một chuỗi ký tự ngẫu nhiên bằng cấu trúc GUID (gọi là receiver token), sau đó lưu receiver token cùng với ID của user xuống DB. Mỗi receiver mới trong DB sẽ có một ID mới bằng kiểu số nguyên để việc tìm kiếm nhanh hơn (gọi là receiver new ID).
4. RNS trả về cho Login service hai thứ: receiver token và receiver new ID của receiver vừa được tạo.
5. Login service gửi trả về cho Mobile Application hai thứ: receiver token và receiver new ID.
6. Mobile Application đưa receiver new ID và receiver token cho RN Adapter. Từ bước này, tất cả các việc giao tiếp trực tiếp với RNS đều được thực hiện nhờ RN Adpater. Thông tin của receiver được đại diện bằng receiver new ID và receiver token.
7. RN Adapter gửi request thêm thiết bị cho receiver, với receiver trên RNS có ID là receiver new ID và có token là receiver token. Request có kèm theo thông tin về thiết bị như: hệ điều hành, URI, IMEI.
8. RNS kiểm tra thông tin receiver và thêm thiết bị vào cơ sở dữ liệu rồi trả về kết quả cho RN Adapter.
9. RN Adapter trả về kết quả cho Mobile Application.

Còn về mô hình Authorization, RNS phải cấp quyền cho những user nhận được thông báo mới có quyền xem nội dung thông báo. Ý tưởng cho việc đó cũng khá đơn giản, ta cũng sẽ dùng token: mỗi thông báo sẽ có một token, gọi là notification token. Token đó sẽ được gửi nhờ WNS tới RN Adapter cùng với preview data và notification ID. Khi RN Adapter muốn lấy nội dung của một thông báo, nó phải gửi ID của thông báo đó cùng tới notification token để xác nhận rằng nó có quyền đọc thông báo đó.

## BẢN THIẾT KẾ CƠ SỞ DỮ LIỆU

Để có thể đảm đương được các công việc được vạch ra ở “BẢN THIẾT KẾ TỔNG QUÁT CÁC THÀNH PHẦN CỦA HỆ THỐNG” và “BẢN THIẾT KẾ MÔ HÌNH AUTHENTICATION VÀ AUTHORIZATION”, cơ sở dữ liệu của RNS phải làm được các việc sau:

* Lưu thông báo: bao gồm preview data và notification data.
* Lưu người nhận (receiver).
* Thể hiện được một receiver có nhiều notification.
* Thể hiện được một receiver có nhiều thiết bị (device).
* Mỗi device phải thuộc về 1 hệ điều hành (OS).
* Mỗi notification và receiver đều phải có token.
* Mỗi token phải có ngày hết hạn.
* Ánh xạ 1-1 từ user trên AES thành receiver trên RNS.



Mô tả:

Cơ sở dữ liệu gồm có 5 bảng:

* Receiver (**Id**, OldId, DeviceCount, AccessToken, TokenExpiredTime)

Một receiver có một số nguyên Id tự tăng duy nhất; một chuỗi ký tự OldId chứa Id của receiver trên AES; một số nguyên DeviceCount chứa số lượng thiết bị của receiver đó; một chuỗi AccessToken chứa token của receiver đó; TokenExpiredTime chứa thời gian hết hạn của AccessToken.

* Notification (**Id**, NotificationContent, NotificationAccessKey, NotificationPreviewContent)

Một notification có một số nguyên Id tự tăng duy nhất, dữ liệu của thông báo dưới dạng mảng byte (NotificationContent), token cho notification (NotificationAccessKey), dữ liệu dùng để xem trước trên ứng dụng (NotificationPreviewContent).

* Device (**Id**, IMEI, ReceiverNewId, URI, OSId)

Một Device có một số nguyên Id tự tăng duy nhất, một IMEI duy nhất của máy, một Id của receiver xác định chủ sở hữu của thiết bị, một thiết bị chỉ được sở hữu bởi một receiver duy nhất, một URI duy nhất của ứng dụng trên máy đó, một OSId chứa ID của hệ điều hành mà thiết bị đang sử dụng.

* ReceiverNotification (**Id**, ReceiverNewId, NotificationId, IsRead)

Một ReceiverNotification có một số nguyên Id tự tăng duy nhất, Id của receiver trên RNS (ReceiverNewId), Id của notification (NotificationId), notification có thể được đánh dấu là đã đọc hay chưa (IsRead).

* DeviceNotification (**Id**, ReceiverNotificationId, DeviceId)

Một DeviceNotification có một số nguyên Id tự tăng duy nhất. Một thông báo được chia cho từng người nhận (ReceiverId), một ReceiverNotification được chia cho từng Device của người đó (DeviceId).

## SƠ ĐỒ GIAO TIẾP

Dưới đây là mô hình giao tiếp giữa RNS với các tác nhân khác của hệ thống và giữa các tác nhân của chính RNS:



Như trên sơ đồ ta có thể thấy:

* RNS được chia làm hai service con: một service chuyên làm việc với AES (Server Service), nó nhận các yêu cầu như gửi thông báo, thêm thông báo, thêm receiver… Và một service chuyên làm việc trực tiếp với các RN Adapter của Mobile Application (Client Service); nó nhận các yêu cầu như lấy nội dung thông báo, đánh dấu thông báo là đã đọc…. Hai service này dùng chung một DB.
* Trong hai service kể trên thì chỉ có Server service là giao tiếp với các dịch vụ gửi thông báo từ nhà phát triển hệ điều hành, có nghĩa là nó sẽ giữ nhiệm vụ gửi thông báo.
* Ngoài ra, ta có một login service, service này chỉ có nhiệm vụ giúp đăng nhập. Nó giao tiếp với Mobile Application và server service.

Lý do phải tách RNS thành hai service bởi chúng ta có thể đưa chúng chạy trên hai máy khác nhau, do đó hiệu suất sẽ tốt hơn. Ngoài ra, việc đó sẽ giúp độc lập chúng với nhau, có nghĩa là cái này nếu bị sự cố thì cái kia vẫn làm việc được, đồng thời nó sẽ giúp việc bảo mật tốt hơn bằng cách chỉ cho phép các máy trong mạng nội bộ mới truy cập vào server service được chẳng hạn.

# THIẾT KẾ CHI TIẾT RNS

## CÁC SƠ ĐỒ TRẠNG THÁI

Dưới đây là vòng đời trạng thái của ba đối tượng chính trong RNS:

### Notification



### Receiver



### Device



## CÁC SƠ ĐỒ HOẠT ĐỘNG

Dưới đây là các hoạt động chính cần chú ý trong RNS:

### Add Receiver



### Add Device



### Add Notification



### Send All Notification



# CÔNG CỤ PHÁT TRIỂN, CÀI ĐẶT

## CÔNG CỤ PHÁT TRIỂN

Do được phát triển trên C# nên IDE tốt nhất là Visual Studio. Bên cạnh đó, do lập trình cho windows 10 nên một điều bắt buộc là phải lập trình trên Visual Studio 2015 và hệ điều hành phải sử dụng là windows 10. Do vậy, các lập trình viên muốn mở file Solution bắt buộc phải sử dụng Visual Studio 2015 và sử dụng windows 10.

Framework: .Net Framework 4.6, .Net Core 5.1.0

Hệ quản trị cơ sở dữ liệu: Microsoft SQL Server 2014.

Quản ký mã nguồn: GitHub.

Công cụ vẽ các mô hình, sơ đồ: Microsoft Visio 2016.

## HƯỚNG DẪN CÀI ĐẶT RNS CHO NHÀ PHÁT TRIỂN AES

Nếu đã có sẵn một hệ thống (AES), ta có thể làm theo. những bước sau để đưa RNS vào hoạt động.

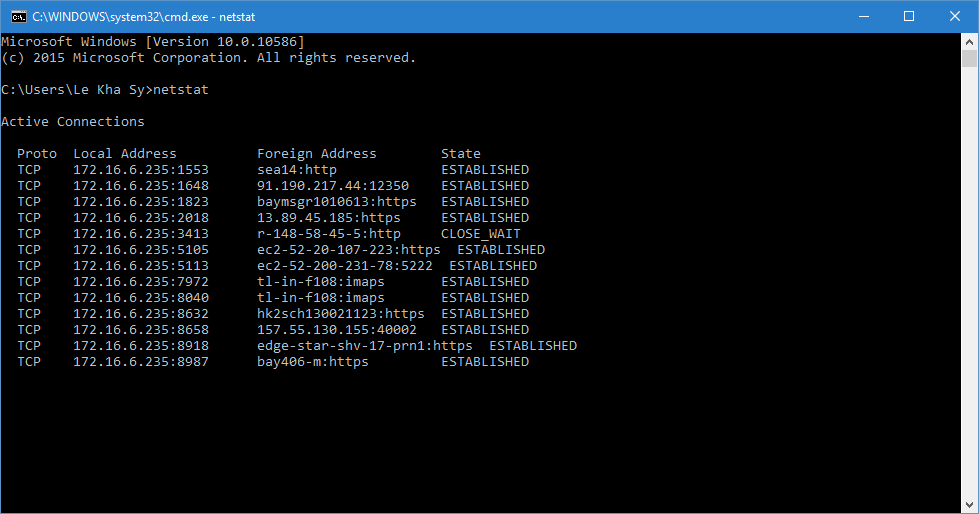
### Khởi tạo Database

Trước hết, ta cần phải khởi tạo DataBase cho RNS. Mở thư mục sau đây trong folder của đề tài ra: Implementation\Design\DBScript. Trong thư mục đó có một file script SQL. Mở file đó bằng MSSQL rồi chạy đoạn script đó. Sau khi chạy, xuất hiện một DB mới có tên Notification\_DB, có nghĩa là đã khởi tạo Db thành công.

### Config RNS

RNS gồm có 2 service là Server Service và Client Service, mỗi service sẽ sử dụng một port trên hệ thống. Trước khi chạy hai service này lên, ta cần phải cấu hình lại file config để đảm bảo service không dùng port mà các ứng dụng khác đang dùng.

Để biết các port nào đang được sử dụng trên máy chủ, ta mở command prompt rồi gõ lệnh **netstat.**



Sau khi đã thấy danh sách port đầu đủ, ta chọn 1 port cho mỗi service. Vào thư mục Implementation\RN\_Enhance\RawNotification\RN\_Build\ServerCommunicator, mở file config ra, tìm đến key :

* **PortNumber**: port muốn dùng.
* **HostNameOrIP**: IP hiện tại của máy để các máy khác có thể truy cập vào được.
* **DBSerNameOrIP**: đổi giá trị của nó thành IP của Database server.
* **DBUserId:** user name để đăng nhập vào database.
* **DBUserPassword:** password của user trên.
* **WNSSecretKey**: secretkey của ứng dụng windows store.
* **WNSPackageSID**: là Package SID của ứng dụng windows store.
* **TokenPeriodInHours**: số giờ có hiệu lực của một token.
* **TokenPeriodInMinutes**: Số phút có hiệu lực của một token.
* **TokenPeriodInSeconds**: Số giây có hiệu lực của một token.

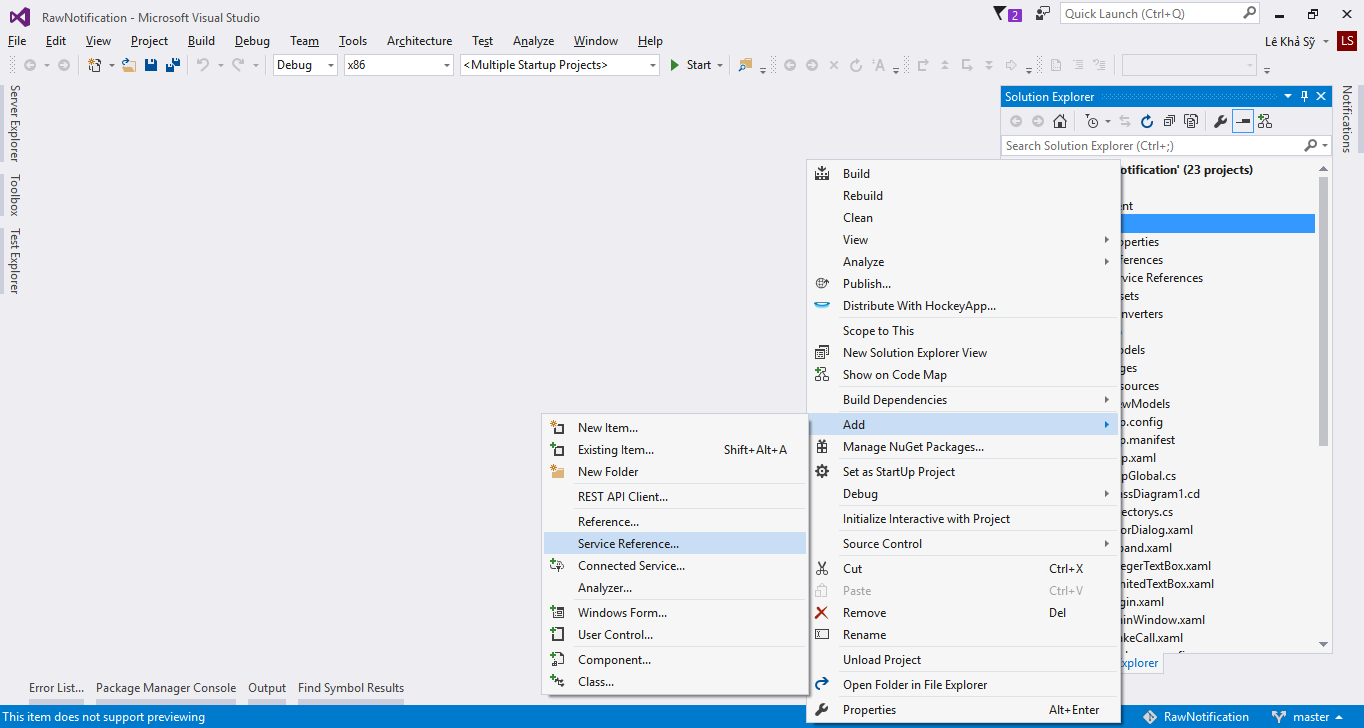
**\* Note**: Tổng thời gian có hiệu lực của một token bằng tổng của cả ba giá trị giờ, phút, giây.

Với Client Service cũng làm tương tự. Tới đây là đã xong việc cài đặt các Module nằm trong phạm vi của đề tài.

### Cài đặt module giao tiếp với Server service

Để có thể yêu cầu RNS gửi thông báo, nhà phát triển AES cần phải cài đặt thêm những đoạn code dùng đẻ tương tác với Server service (là một WCF service). Nếu AES là một ứng dụng được viết trên .Net Framework và có sử dụng Visual Studio để phát triển thi việc tương tác với RNS sẽ hết sức đơn giản. Chỉ cần làm theo những bước sau:

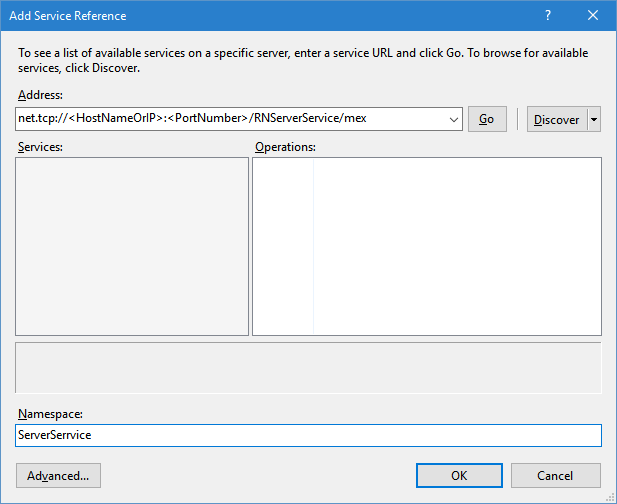
1. Vào thư mục chứa Server Service và khởi chạy service với quyền Admin.
2. Mở solution chứa project cần tương tác với RNS bằng Visual Studio.
3. Click phải vào project cần tương tác với RNS và chọn Add



1. Điền địa chỉ của Server Service vào ô địa chỉ theo định dạng sau:

net.tcp://<HostNameOrIP>:<PortNumber>/RNServerService/mex

1. Đổi tên NameSpace ở ô Namespace.



1. Ấn OK.

Như vậy là AES đã có thể tương tác với RNS.

### Xây dựng login service

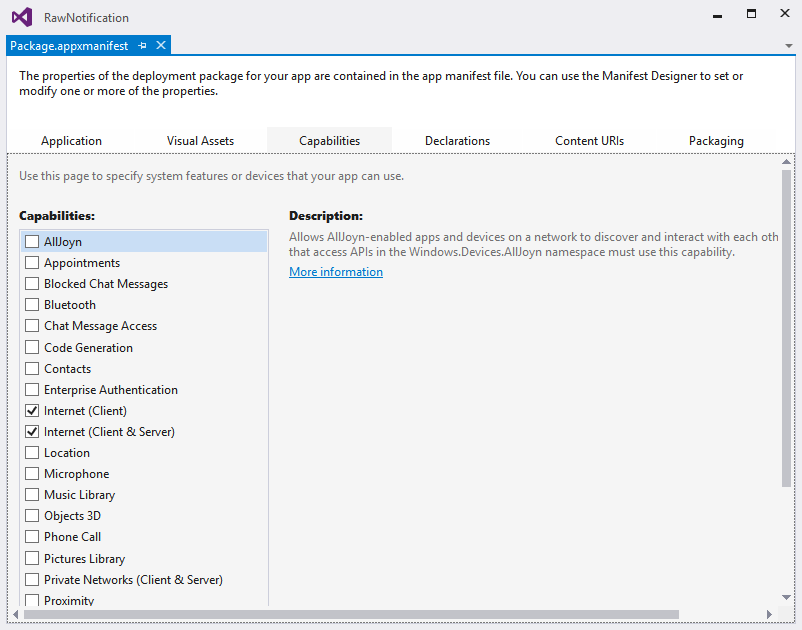
Như đã nói trước, việc login sẽ do nhà phát triển AES đảm nhiệm. Việc xây dựng service này cũng khá đơn giản, ta có thể sử dụng một WCF service rồi add reference tới Server Service. Hiện tại trong solution của đề tài đã làm demo sẵn một login service bằng WCF, nhà phát triển AES có thể tham khảo. Tuy nhiên, nhà phát triển AES hoàn toàn có thể sử dụng một công nghệ khác như Web API, Web Service, hay thậm chí là làm việc dưới tầng TCP-IP hoặc Web Socket đều được.

### Xây dựng Mobile Application

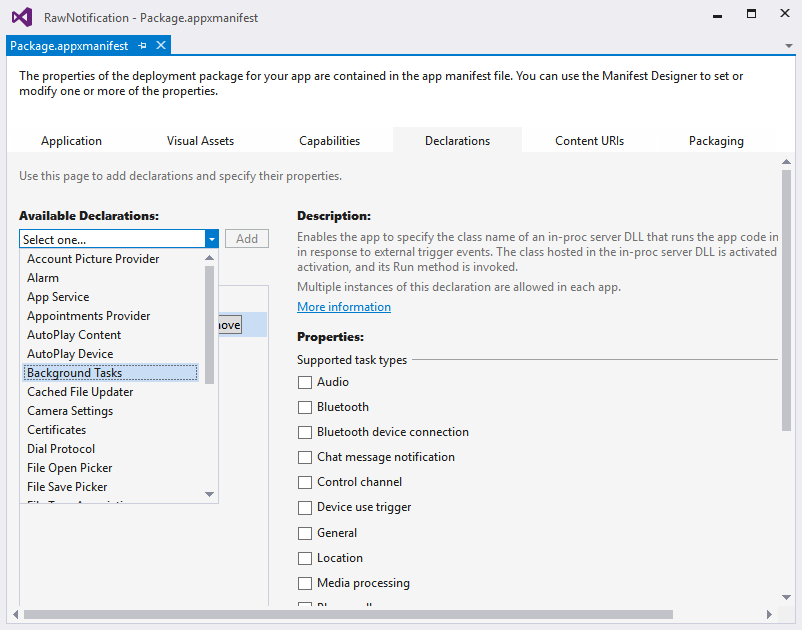
Với RN Adapter, việc xây dựng mobile application sao cho có thể giao tiếp với RNS, đồng thời quản lý các thông báo đã được lấy về đã được đơn giản hóa rất nhiều.

Việc xây dựng Mobile Application có thể thực hiện theo các bước sau:

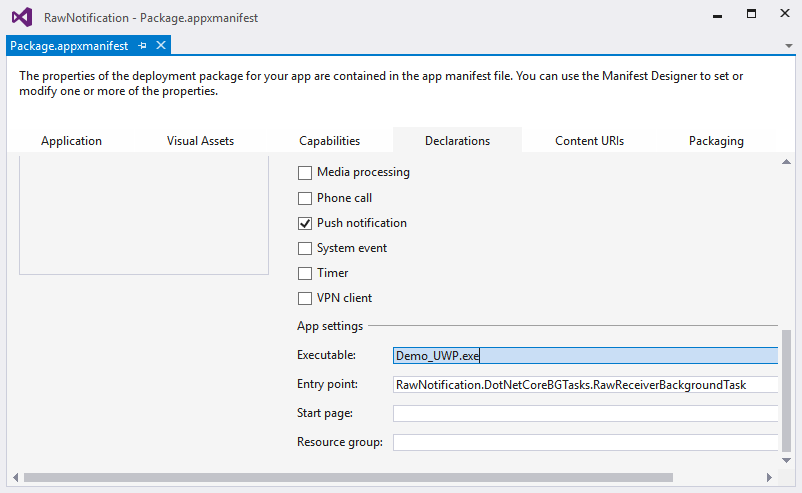
* Tạo một ứng dụng UWP trên visual studio 2015
* Add RNS vào solution của ứng dụng
* Add refference từ mobile application tới các project:
  + RawNotification.DotNetCoreBL
  + RawNotification.DotNetCoreLibs
  + RawNotification.DotNetCoreModels
  + RawNotification.DotNetCoreUserCodes
* Mở file Package.appxmanifest trong project ra. Đây là file chứa tất cả các khai báo về khả năng của ứng dụng (nó cần nhữn gì, nó sẽ làm những gì).
  + Vào tab capabilities
  + Chọn internet (Client)



* + Chuyển qua tab Declaration và add 2 background task:



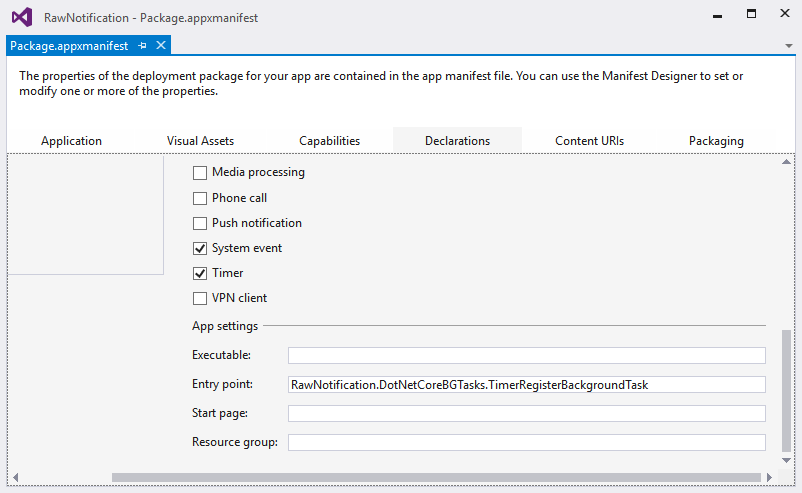
* + Với Background task thứ nhất, ta làm như hình:



Trong đó, Executable là assembly name của project. Để xem assembly name, click phải vào project-> property->application.

Entrypoint đặt là:  
RawNotification.DotNetCoreBGTasks.RawReceiverBackgroundTask

* + Với background thứ 2, ta cấu hình như sau:



Riêng mục này, ta để Executable trống.

* Cài đặt phần login với login service
* Sau khi login thành công thì gọi RNAdapter.Initialize()
* Tiếp theo, đăng ký sự kiện nhận được notification bằng cách gọi hàm RegisterNotificationReceivedAsyncEvent trong RNAdapter.
* Cuối cùng, ta vào project RawNotification.DotNetCoreUserCodes, class UserCodes có một hàm static: OnNotificationReceivedInBGTask. Khi notification được nhận ở Background Task thì hàm đó sẽ được gọi. Ta chỉ cần viết code vào đó để xử lý sự kiện nhận thông báo ở background task là được.

Trong project của đề tài hiện đã có một ứng dụng UWP demo, có thể vào đó để tham khảo.

# KIỂM THỬ

Một đề tài cho dù có áp dụng bao nhiêu công nghệ, bao nhiêu pattern, nhưng nếu không sử dụng được thì coi như vô nghĩa. Việc kiểm thử phần mềm là bắt buộc nếu muốn biết được sự đúng đắn, tính tương thích… của phần mềm.

## PHƯƠNG PHÁP KIỂM THỬ

Do RNS là một module của một hệ thống khác nên việc kiểm thử nó đòi hỏi phải có một phần mềm khác có sẵn (AES), sau đó cho AES tương tác với RNS. Nếu hai hệ thống đó làm việc hiệu quả với nhau thì việc kiểm thử sự tương thích của RNS đã thành công.

## MỤC TIÊU KIỂM THỬ

Nếu kết quả của kiểm thử đạt được các tiêu chí sau thì có thể xem việc kiểm thử đã thành công:

* AES có thể gửi yêu cầu thêm thông báo, gửi thông báo tới RNS.
* Mobile Application có thể nhận được thông báo trên cả UI lẫn dưới background task.
* Hệ thống không gặp lỗi khi gửi nhiều thông báo cùng một lúc.
* Tốc độ gửi nhanh (khoảng 1000 thông báo trong vòng 5 giây).
* Dữ liệu đảm bảo được tính toàn vẹn.

## CÀI ĐẶT MÔI TRƯỜNG KIỂM THỬ

Môi trường cần có để thực hiện kiểm thử là:

* Một phần mềm quản lý có sẵn (AES).
* Các phần mềm cần thiết khác: SQL Server 2014, Windows 10, Visual studio 2015.
* Nếu muốn chạy RNS trên một hoặc hai máy độc lập, ta có thể dùng thêm hai máy nữa chạy Window Vista, 7, 8, 10 (có thể dùng máy ảo để kiểm thử). (do Windows XP không thể cài .Net 4.0 hoặc cao hơn).
* Mạng Internet có độ trễ thấp (dưới 10 mili seconds được đo dựa theo trang speedtest.net).
* Bởi vì RNS giao tiếp với Mobile Application qua Intrenet, nó cần một IP tĩnh hoặc một tên miền. Do việc mua IP tĩnh không rẻ nên em đã sử dụng dynamic DNS bằng cách đăng ký một tên miền trên trang noip.com.
* Một điện thoại Windows chạy hệ điều hành Windows 10 Mobile.

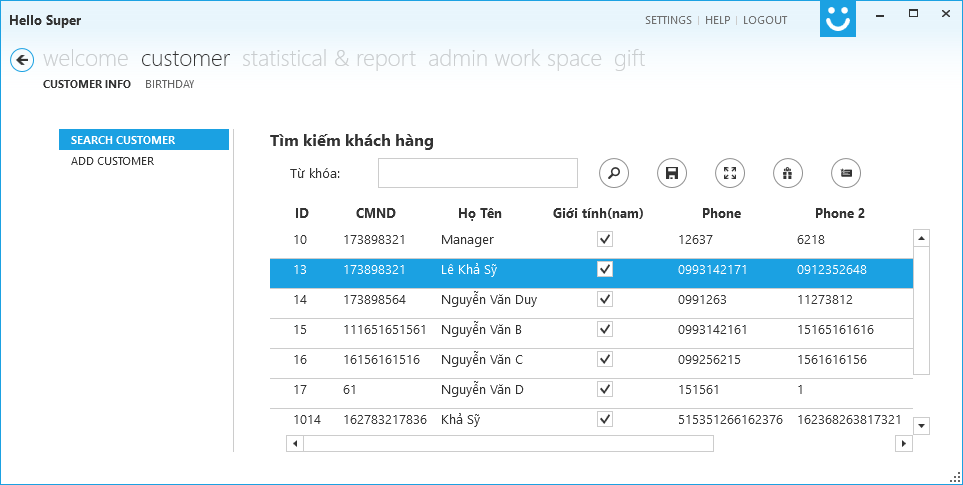
Ta tiến hành cài đặt các phần mềm trên lên một máy, SQL server có thể cài trên một máy khác.

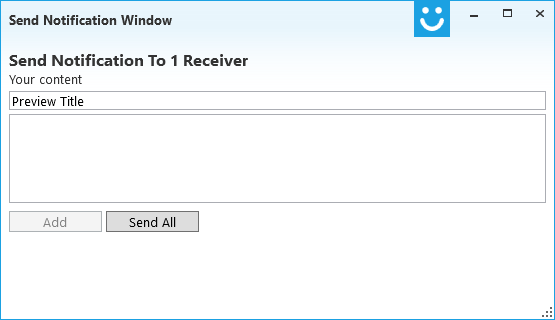
## TRIỂN KHAI RNS TRÊN MÔI TRƯỜNG KIỂM THỬ

Sau khi đã có môi trường kiểm thử, ta tiến hành cài đặt RNS lên môi trường theo phần “5.2 HƯỚNG DẪN CÀI ĐẶT RNS CHO NHÀ PHÁT TRIỂN AES”. Tuy nhiên, ở phần này, ta sẽ nói rõ ra ta sử dụng cái gì làm AES, công nghệ nào để làm Login Service và Mobile Application có chủ đề là gì.

AES trong lần kiểm thử này chính là phần mềm quản lý quan hệ khách hàng, cũng là đề tài môn lập trình .net nâng cao của em. Nó được viết bằng C#, IDE là visual studio, công nghệ WPF (Windows Presentation Foundation).

Dưới đây là một số hình ảnh của AES:





Login Service là một service WCF khác, nó dùng chung cơ sở dữ liệu với AES để xác thực user.

Tuy AES là một phần mềm quản lý quan hệ khách hàng, nhưng để tính sử dụng lại của code cao, Mobile Application sẽ không hướng đến một ứng dụng nào cả. Có nghĩa là nó không được thiết kế giao diện cho phù hợp với một nhóm người dùng nào cả. Mà đơn giản là nó chỉ có nhiệm vụ nhận thông báo dạng chuỗi và hiển thị nó lên cho người dùng xem. Một ứng dụng đơn giản sẽ giúp người cần tham khảo dễ dàng hơn khi tham khảo. Ngoài ra, nó còn có thể gửi thông báo tới các người dùng khác.

Dưới đây là một số hình ảnh của Mobile Application:

|  |  |
| --- | --- |
| **Main_04** | **login** |
|  |  |
|  |  |

## KẾT QUẢ KIỂM THỬ

Dưới đây là một vài kết quả kiểm thử đáng chú ý:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Mô tả | Các bước | Kết quả mong đợi | Kết quả nhận về | Thành công |
| TC\_SyLK\_01 | Kiểm thử việc đăng nhập trong môi trường lý tưởng. | * Mở ứng dụng * Nhập tài khoản, mật khẩu rồi ấn nút đăng nhập | Đăng nhập thành công, cơ sở dữ liệu có thêm một người dùng và một thiết bị | Như mong đợi. | Yes |
| TC\_SyLK\_02 | Thêm thông báo trong môi trường lý tưởng. | * Mở AES. * Mở giao diện gửi thông báo. * Chọn một hoặc nhiều receiver. * Nhập notification content và preview data và ấn add notification | * Một notification mới được thêm vào DB. | Như mong đợi. | Yes |
| TC\_SyLK\_03 | Gửi thông báo trong môi trường lý tưởng | * Mở AES * Đảm bảo trong DB đã add sẵn một số notification. * Ấn nút gửi thông báo | Thông báo được gửi tới thiết bị của người dùng | Như mong đợi | Yes |
| TC\_SyLK\_04 | Gửi thông báo khi RNS không có internet | * Mở AES * Thêm một hoặc nhiều notification * Ngắt kết nốii internet của RNS * Bấm nút gửi thông báo. | RNS không gửi được thông báo, tuy nhiên, không phát sinh lỗi. Các thông báo không được gửi sẽ được giữ lại để gửi trong lần sau | Như mong đợi | Yes |
| TC\_SyLK\_05 | Nhận notification trên RN Adapter khi internet không tốt. | * Gửi một notification tới thiết bị trong điều khiện thiết bị không có kết nối tốt. (môi trường 2G) | RN Adapter vẫn nhận được thông báo ngay lập tức. | RN Adapter vẫn nhận được thông báo, nhưng bị trễ. | No |

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## KẾT QUẢ ĐẠT ĐƯỢC

Đồ án tìm hiểu và xây dựng module gửi thông báo tới thiết bị di động đã hoàn thành được những chức năng sau:

* Hỗ trợ các AES được viết trên mọi nền tảng
* Hỗ trợ các Mobile Application được viết trên nền tảng .Net Core
* Đã cài đặt và gửi thông báo tới Mobile Application thành công.

## ĐÁNH GIÁ KẾT QUẢ

Phần đánh của RNS thực ra có hai mục:

* Kiểm tra độ tin cậy (code logic).
* Kiểm tra tính dễ dùng, dễ tùy chỉnh, tương thích cao khi áp dụng vào một AES.

Nếu ta cho độ thành công là 100% thì độ tin cậy sẽ là 40% và tính dễ dùng, dễ tùy chỉnh và tương thích cao là 60%. Lý do bở: nếu logic bị sai, ta có thể chỉnh sửa một cách dễ dàng. Nhưng nếu tính dễ dùng, dễ tùy chỉnh và tương thích cao không đạt được, có nghĩa là architecture của RNS gặp phải vấn đề. Mà nếu sai khi thiết kế thì sẽ đem đến vô số hệ lụy sau này. Tuy nhiên, thế nào là “dễ” thì khó mà xác định được. Vì vậy, ta sẽ quy ước: dễ là khi không có thao tác thừa. Có nghĩa là tất cả các việc làm tự động được thì RNS sẽ phải làm, những đoạn code hay phần cấu hình nào có thể làm tự động được mà chưa có thời gian làm thì ta sẽ tính là chưa tốt. Các mục của phần này sẽ dựa theo chương 5: hướng dẫn cài đặt RNS cho nhà phát triển AES. Thứ hai là những thứ chưa hỗ trợ ở thời điểm hiện tại được cũng sẽ được coi là khó dùng. Tuy nhiên, cho dù đã có tiêu chí đánh giá, nhưng kết quả có lẽ không tránh khỏi đôi chút chủ quan, vậy nên phần đánh giá này chỉ mang tính chất tham khảo. Nếu muốn một đánh giá khách quan, ta cần nhiều hơn một người cho ý kiến.

Về độ tin cậy, ta có thể thấy chất lượng khá tốt, bắt hầu hết các lỗi có thể xảy ra trong ứng dụng, ví dụ như internet lỗi. Nhưng về việc nhận thông báo khi kết nối internet không tốt có vẻ như chưa đảm bảo được tính tức thời. Bên cạnh đó là một chi tiết về cách lưu dữ liệu trên RN Adapter. Hiện tại tầng Data Access của RN Adapter đang lưu các thông báo nhận được vào file có định dạng text. Cách này khá thô sơ và dễ khiến ứng dụng tràn bộ nhớ nếu thông báo được gửi về có dung lượng lớn. Bên cạnh đó, cách này sẽ khiến hiệu suất bị giảm đi rất nhiều. Đây là một điểm trừ cho hiệu suất của RN Adapter. Trong tương lai, vấn đề này sẽ được giải quyết bằng cách sử dụng SQLite để thao tác với dữ liệu. Theo quan điểm cá nhân, mức thành công của độ tin cậy là 35%/40%.

Về tính dễ dùng, do code được viết theo mô hình nhiều lớp kết hợp với dependency injection pattern nên nếu hiểu về mô hình này thì sẽ rất dễ hiểu cấu trúc của RNS. Bên cạnh đó, tất cả các công nghệ được sử dụng trên RNS đều là những công nghệ khá nổi tiếng, nên việc nắm bắt chúng không hề khó khăn. Tất cả các điều trên cho thấy việc tùy chỉnh trên RNS là khá dễ dàng. Còn về việc tương thích cao, RNS đã chứng minh được tính tương thích cao của nó khi được áp dụng vào một phần mềm quản lý quan hệ khách hàng. Không những vậy, nếu xét trên lý thuyết, RNS được thiết kế theo. mô hình hướng dịch vụ (SOA), nên việc không tương thích với một hệ thống nào đó dẫn tới không thể áp dụng RNS vào là điều khó xảy ra. Bên cạnh đó, đi kèm với RNS là một dự án Demo, nên nếu ai đó muốn áp dụng RNS vào hệ thống của họ thì chỉ cần xem qua Demo và đọc qua tài liệu là sẽ hiểu được.

Tuy nhiên, như ta thấy trong phần 5.2, việc cài đặt RNS vào AES trải qua khá nhiều bước. Nhưng thực ra rất nhiều bước trong số đó có thể được thực hiện bằng một công cụ tự động. Ví dụ như bước cấu hình file config, ta hoàn toàn có thể viết một tool nhỏ, giúp người dùng cấu hình file đó trên giao diện người dùng chứ không phải làm việc trên văn bản XML.

Một điểm nữa là do thiếu thời gian nên RNS mới chỉ hỗ trợ Windows mà chưa hỗ trợ Android và IOS nên đây có thể được xem là một sự thiếu hoàn hảo ở thời điểm hiện tại. Tuy nhiên, đây không phải là một lỗi thiết kế hệ thống, chỉ là do thiếu thời gian để cài đặt mà thôi. Ta có thể hoàn thành điểm này trong tương lai.

Vì vậy, em nghĩ mức thành công của tính dễ dùng, dễ tùy chỉnh, tương thích cao đạt khoảng 45%/60%.

## HƯỚNG PHÁT TRIỂN

Trong thời gian sắp tới, em sẽ tập trung vào những việc sau:

* Dùng SQLite cho tầng Data Access trên RN Adapter.
* Phát triển RN Adapter cho các nền tảng khác như Android và IOS
* Hiện tại dự án RNS đã được em đưa lên GitHub, trong thời gian tới, em sẽ tìm thêm những cộng tác viên để cùng xây dựng RNS.

# Tài liệu tham khảo

1. Richard Blum. (2003). *C# Network Programming.* Sybex.
2. Trần Đình Quế - Nguyễn Mạnh Sơn. (2007). *Phân tích & Thiết kế hệ thống thông tin.* Học Viện Công Nghệ Bưu Chính Viễn Thông.
3. John Sharp. (2007) *Microsoft Windows Communication Foundation Step by Step*. Custom Editorial Production, Inc.
4. Bob Tabor. (2015) *Windows 10 Development for Absolute Beginners*.
5. Phần lớn kiến thứ em tổng hợp được là từ các trang web: <http://msdn.com>, <http://chanel9.com>, <http://stackoverflow.com>, <http://codeproject.com> ...