

**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA TIN HỌC**

[LOGO TRƯỜNG]

**ĐỒ ÁN NGÀNH**

**TÌM HIỂU VÀ XÂY DỰNG MODULE  
GỬI THÔNG BÁO TỚI THIẾT BỊ DI ĐỘNG**

**LÊ KHẢ SỸ**

**Bình Dương, 01/2016**

**TRƯỜNG ĐẠI HỌC BÌNH DƯƠNG**

**KHOA TIN HỌC**



[LOGO TRƯỜNG]

**ĐỒ ÁN NGÀNH**

**TÌM HIỂU VÀ XÂY DỰNG MODULE  
GỬI THÔNG BÁO TỚI THIẾT BỊ DI ĐỘNG**

**SVTH: Lê Khả Sỹ - 12050003**

**GVHD: TS. Tô Tuấn**

**Bình Dương, 01/2010**

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.

Đặc biệt, em xin cảm ơn thầy TS. Tô Tuấn, người trực tiếp hướng dẫn em để hoàn thành đồ án này. Thầy đã chỉ bảo em hết sức tận tình và giúp em có được những ý tưởng tuyệt vời. Không những thế, em xin cảm ơn thầy vì đã cho em thấy được sự cố chấp, cái tôi lớn từ em, giúp em có thể nhẹ nhàng lắng nghe những góp ý chân thành đến từ mọi người.

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

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

......................................................................................................................................

Bình Dương, ngày … tháng 1 năm 2016

Giảng viên hướng dẫn

TS. Tô Tuấn

MỤC LỤC

[CHƯƠNG 1: MỞ ĐẦU 7](#_Toc439962002)

[1.1 LÝ DO HÌNH THÀNH ĐỀ TÀI 7](#_Toc439962003)

[1.2 MỤC TIÊU ĐỀ TÀI 8](#_Toc439962004)

[1.3 Ý NGHĨA THỰC TIỄN CỦA ĐỀ TÀI 9](#_Toc439962005)

[1.4 PHẠM VI ĐỀ TÀI 9](#_Toc439962006)

[1.5 CÁC TỪ VIẾT TẮT 10](#_Toc439962007)

[CHƯƠNG 2: TÌM KIẾM GIẢI PHÁP 10](#_Toc439962008)

[2.1 CÁC GIẢI PHÁP 10](#_Toc439962009)

[2.2 TỔNG QUAN VỀ WINDOWS PUSH NOTIFICATION SERVICE (WNS) 11](#_Toc439962010)

[2.1.1 WNS là gì? 11](#_Toc439962011)

[2.1.2 Tại sao nên sử dụng WNS? 11](#_Toc439962012)

[2.1.3 WNS làm việc như thế nào? 12](#_Toc439962013)

[2.1.4 Các vấn đề bảo mật với mô hình của WNS 13](#_Toc439962014)

[2.2 LÀM VIỆC VỚI WNS 14](#_Toc439962015)

[2.2.1 Các loại thông báo trên Windows 10 14](#_Toc439962016)

[2.2.2 Bước 1: Xin cấp một kênh thông báo 16](#_Toc439962017)

[2.2.3 Bước 2: Gửi kênh thông báo cho RNS 16](#_Toc439962018)

[2.2.4 Bước 3: Gửi thông báo cho WNS 17](#_Toc439962019)

[2.2.5 Bước 5 Nhận thông tin trả về từ WSN sau khi gửi thông báo 21](#_Toc439962020)

[2.2.6 Bước 6 Nhận thông báo ở Client 24](#_Toc439962021)

[2.3 CÁC XỬ LÍ PHỤ 27](#_Toc439962022)

[2.3.1 Gửi cho thiết bị hay gửi cho người dùng 27](#_Toc439962023)

[2.3.2 Xác nhận đã nhận được thông báo 28](#_Toc439962024)

[CHƯƠNG 3: THIẾT KẾ MODULE THÔNG BÁO TRÊN NỀN TẢNG .NET FRAMEWORK 28](#_Toc439962025)

[3.1 MỤC TIÊU MÀ BẢN THIẾT KẾ HƯỚNG ĐẾN 28](#_Toc439962026)

[3.2 SƠ ĐỒ Ý TƯỞNG CÁC THÀNH PHẦN HỆ THỐNG 28](#_Toc439962027)

[3.3 THIẾT KẾ CƠ SỞ DỮ LIỆU 30](#_Toc439962028)

[3.4 SƠ ĐỒ TỔNG QUÁT HỆ THỐNG HOÀN THÀNH 32](#_Toc439962029)

[CHƯƠNG 4: THIẾT KẾ CHI TIẾT CÁC THÀNH PHẦN CỦA HỆ THỐNG 34](#_Toc439962030)

[4.1 THIẾT KẾ CHO RNS 34](#_Toc439962031)

[4.2 THIẾT KẾ CHO AES COMMUNICATOR VÀ SERVER INTERFACE 36](#_Toc439962032)

[4.3 THIẾT KẾ CHO MC COMMUNICATOR VÀ MCAI 38](#_Toc439962033)

[CHƯƠNG 5: CÔNG CỤ PHÁT TRIỂN, CÀI ĐẶT VÀ KIỂM THỬ 39](#_Toc439962034)

[5.1 CÔNG CỤ PHÁT TRIỂN 39](#_Toc439962035)

[5.2 HƯỚNG DẪN CÀI ĐẶT CHO NHÀ PHÁT TRIỂN AES 39](#_Toc439962036)

[5.3 KIỂM THỬ 40](#_Toc439962037)

[5.3.1 Mục tiêu kiểm thử 40](#_Toc439962038)

[5.3.2 Tiến hành kiểm thử 40](#_Toc439962039)

[5.3.3 Kết quả kiểm thử 40](#_Toc439962040)

[CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 41](#_Toc439962041)

[6.1 KẾT QUẢ ĐẠT ĐƯỢC 41](#_Toc439962042)

[6.2 NHỮNG HẠN CHẾ 41](#_Toc439962043)

[6.3 HƯỚNG PHÁT TRIỂN 42](#_Toc439962044)

[Tài Liệu Tham Khảo 43](#_Toc439962045)

# CHƯƠNG 1: 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. Thông tin được đưa đến chỉ được phân kênh tới mức lớp, có nghĩa là một thông tin tối đa chỉ thông báo 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 đưa thông tin tới cho từng sinh viên, đồng thời sinh viên phải tiếp cận được thông tin đó một cách dễ dàng.

Với thực tế hiện nay, em quyết định chọn thiết bị di động sử dụng một trong ba hệ điều hành: Windows, IOS hoặc Android làm kênh chuyển tin tức bởi vì đây là ba hệ điều hành phổ biến nhất hiện nay và các thiết bị di động đến thời điểm hiện tại đã hết sức phổ biến, giá thành cho mỗi thiết bị cũng không quá cao.

Tuy nhiên, do thời gian có hạn nên em chỉ làm demo cho một hệ điều hành đó là windows. Lý do em chọn Windows bởi vì:

* Có thể được phát triển bằng ngôn ngữ C# - một ngôn ngữ được đào tạo khá kỹ ở trường Đại học Bình Dương.
* Thời điểm làm đề tài trùng với lúc Windows 10 được phát hành, phiên bản windows này có một vài điểm đặc biệt so với các phiên bản trước của nó:
* Các ứng dụng viết cho windows 10 mà có mục tiêu hướng tới Universal Windows Platform đều có thể chạy được trên tất cả các thiết bị windows: Từ PC, tablet, phone, Xbox hay thậm chí là các thiết bị IOT.
* Windows 10 sẽ là phiên bản cuối cùng của windows. Vì vậy tính tương thích trong thời gian dài có thể sẽ tốt hơn.

- Tuy nhiên, module có thể dễ dàng được nâng cấp lên và có khả năng hỗ trợ cho các hệ diều hành khác như IOS và Android.

## 1.2 MỤC TIÊU ĐỀ TÀI

Nếu nhìn một cách tổng quát hơn, ta có thể dễ dàng nhận ra nhu cầu thêm chức năng thông báo cho một hệ thống có sẵn là rất lớn. Ví dụ như một công ty có một hệ thống quản lí nhân sự, nhưng đôi khi cần thông báo qua điện thoại một tin tức nào đó không quá quan trọng của công ty, họ sẽ làm như thế nào? Đăng lên web để nhân viên phải tự mò lên xem hay tốn tiền với việc điện thoại cho từng người một hoặc nhắn tin hàng loạt?  
 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.

Vậy khi người ta cần một chức năng như vậy cho hệ thống, họ sẽ phải tự viết code từ đầu đến cuối? Điều đó thật quá lãng phí thời gian, nhưng thực tế thì trên internet chưa có một module nào có thể đáp ứng một cách đầy đủ cho việc đó cả. Chỉ có những đoạn code rời rạc, không bắt hết các lỗi có thể xảy ra, không thể áp dụng vào bất kì hệ thống nào và độ tin cậy gần như bằng không. Còn về các hệ thống có tính phí thì thực ra là đã có một dịch vụ do Microsoft xây dựng có tên Notification Hub. Tuy nhiên, dịch vụ này có những điểm yếu như sau:

* Phải trả phí mới có thể sử dụng
* Chỉ gửi được những thông báo có dạng xml và không lo luôn việc gửi những thông báo có nội dung lớn.

Với cách nhìn đó, em hi vọng có thể tạo ra một module 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, 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.

## 1.3 Ý NGHĨA THỰC TIỄN CỦA ĐỀ TÀI

Việc phát triển module này sẽ giúp ích rất nhiều cho các nhà phát triển cũng như giúp cho người dùng đầu cuối tiếp cận thông tin dễ dàng hơn, kịp thời hơn, thụ động hơn.

## 1.4 PHẠM VI ĐỀ TÀI

Chức năng này sẽ giúp mọi người dễ dàng tiếp cận thông tin qua các thiết bị chạy Windows 10. Đồng thời những lập trình viên khác cũng có thể phát triển thêm cho các nên tảng khác như Android hay IOS mà không cần thay đổi cấu trúc cơ sở dữ liệu lẫn thiết kế module, cũng như không thay đổi quá nhiều về xử lý và giao diện phần mềm quản lý.  
 Bên cạnh đó, phải thiết kế hệ thống sao cho phần thông báo tin tức chỉ là một module và móc nối không quá sâu vào hệ thống quản lý. Điều này sẽ giúp cho việc áp dụng vào một hệ thống khác trở nên đơn giản hơn, đồng thời việc tháo gỡ module này ra cũng đơn giản và hoàn toàn không có tác động xấu đối với hệ thống chính và phải đảm bảo được các yêu cầu sau:

* Đơn giản đối với các nhà phát triển muốn áp dụng module vào hệ thống có sẵn của họ
* Phù hợp với mọi loại hệ thống có sẵn nào, với bất cứ nền tảng nào
* Độ tin cậy cao
* Tốc độ hồi đáp cao cho server, tiết kiệm năng lượng cho thiết bị đầu cuối

## 1.5 CÁC TỪ VIẾT TẮT

|  |  |
| --- | --- |
| **Viết tắt** | **Đầy đủ** |
| WNS | Windows Notification Service |
| RNS | Raw Notification Server |
| AES | Already Exist System |
| IFMCAI | Interface For Mobile Client Application |
| URI | Uniform Resource Identifier |
| IFAES | Interface For Already Exist System |

# CHƯƠNG 2: TÌM KIẾM GIẢI PHÁP

## 2.1 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 khi ứng dụng không hoạt động thì vẫn có cách để gửi thông báo tới cho chúng thông qua Socket activity trigger, nhưng do phải làm việc trên socket nên việc này rất vất vả và chúng ta phải kiểm soát rất nhiều lỗi trong quá trình liên lạc. 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 giúp các nhà phát triển đơn giản hóa việc gửi thông báo cho ứng dụng của họ.  
Ta sẽ tìm hiểu chi tiết về Windows Notification Service, các ưu điểm khi sử dụng giải pháp này trong phần tiếp theo.

## 2.2 TỔNG QUAN VỀ WINDOWS PUSH NOTIFICATION SERVICE (WNS)

### 2.1.1 WNS là gì?

WNS là một dịch vụ 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. Máy chủ đám mây này có thể là bất cứ thiết bị nào: từ một main frame hay thậm chí chỉ là một chiếc điện thoại. 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 dãy IP nên nó có thể là bất kỳ thiết bị nào ở bất kỳ đâu và có thể chạy bất kỳ loại hệ điều hành nào. Với WNS, máy chủ của chúng ta là một đám mây, nhưng để đơn giản hóa việc gọi tên, ta quy ước gọi máy chủ đám mây của nhà phát triển là Raw Notification Server (RNS).

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.  
WNS có nhiều ưu điểm đáng chú ý hơn MPNS như:

* 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.
* Khả năng gửi thông báo tới cho ứng dụng ngay cả khi ứng dụng đã bị tắt.

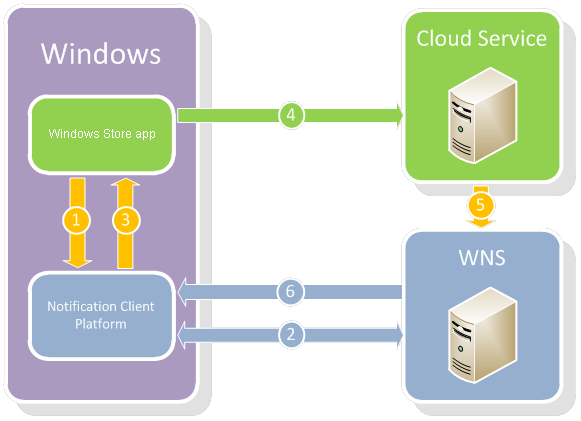
### 2.1.2 Tại sao nên sử dụng WNS?

Như đã đề cập ở phần trước, các giải pháp thủ công có rất nhiều hạn chế trong hiệu suất, độ tin tưởng của code, thông báo khi ứng dụng không hoạt động…

Với WNS, tất cả các vấn đề trên sẽ được giải quyết triệt để. Người phát triển phần mềm sẽ không còn phải lo bắt các lỗi nhỏ trên client và nếu có thay đổi gì trong các phiên bản sau của hệ điều hành, Microsoft vẫn sẽ đảm bảo tính tương thích của ứng dụng sử dụng WNS.

Vậy WNS đã làm việc như thế nào để đảm bảo được các điều trên?

### 2.1.3 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.
* Bước 6: WNS nhận được yêu cầu cần gửi thông báo, nó sẽ gửi thông báo tới cho thiết bị sớm nhất có thể.

Với sơ đồ trên, ta có thể tưởng tượng rằng WNS là một bưu điện, ứng dụng sẽ xin địa chỉ của mình từ bưu điện và gửi cho RNS. Khi có gì cần gửi cho ứng dụng, RNS sẽ gửi thư có địa chỉ nhận mà đã nhận được từ ứng dụng trước đó. Như vậy thì thật đơn giản đối với nhà phát triển.

Đây là mô hình làm việc của WNS được cung cấp bởi Microsoft. Còn đối với các hệ điều hành khác, cách làm việc trên hệ thống gửi thông báo của họ cũng tương tự.

### 2.1.4 Các vấn đề bảo mật với mô hình của WNS

Tuy nhiên có một vấn đề nho nhỏ trong việc bảo mật: trong 6 bước trên, chỉ có bước 4 là không có sự tham gia của Microsoft. Vậy nếu ở bước này, bất kỳ một thiết bị nào cũng có thể làm giả một gói tin để gửi tới cho RNS nhằm đánh cắp thông tin của người dùng. Để hiểu thêm về vấn đề này, chúng ta hãy xem qua một URI nhận được từ client:

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

Rất dễ dàng để nhận ra, đây 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. 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.

Thêm vào đó là vấn đề an toàn đối với bước 5: ta hãy nhìn lại bước 4, giả sử hacker bằng cách nào đó lấy về được URI của một ứng dụng bằng cách bắt các gói tin được truyền trên mạng ở bước 4 chẳng hạn. Vậy thì ở bước 5, hacker có thể gửi yêu cầu nhờ WNS chuyển một thông báo rác, hay một thông tin sai sự thật tới cho người dùng không khi mà hacker đã có URI của thiết bị?  
Với Microsoft Push Notification Service, hacker hoàn toàn có thể làm điều đó. Nhưng với WNS, việc đó đã được giải quyết bằng phương pháp xác thực danh tính của RNS. Có nghĩa là chỉ có các RNS của nhà phát triển mới được gửi thông báo tới cho ứng dụng của họ.

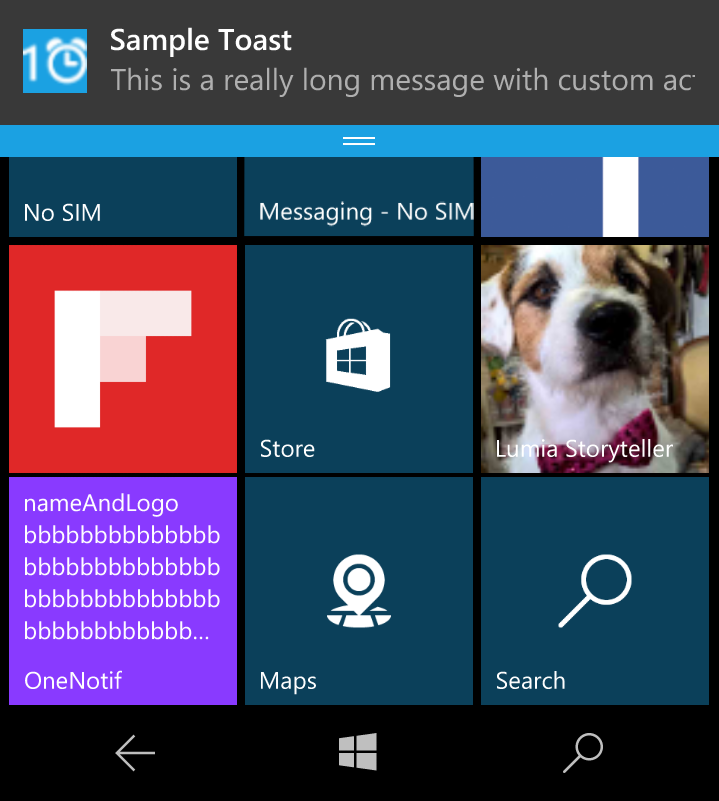
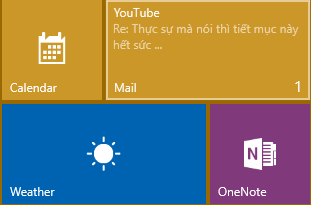
Vậy làm sao biết máy chủ nào là của nhà phát triển?  
Giải pháp rất đơn giản: khi gửi thông báo cho WNS, ta cần phải thêm mật khẩu gọi là “thẻ truy cập” mà WNS cấp cho chúng ta. Mật khẩu này chỉ có hiệu lực trong một khoảng thời gian nhất định. Khi hết hiệu lực, ta phải xin WNS cấp cho một mật khẩu mới. Chi tiết của việc xác thực danh tính với WNS sẽ được đề cập ở mục 2.2.4.

## 2.2 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 làm tự động bởi Microsoft 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.

### 2.2.1 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 và không được gửi gì khác ngoài những thứ được quy định. 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.

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.

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

Trong .Net, 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.  } |

Phương thức CreatePushNotificationChannelForApplicationAsync sẽ ném ra một ngoại lệ nếu kết nối internet bị trục trặc.   
Tuy nhiên, do mỗi URI phải thuộc một ứng dụng windows store nào đó, AES sẽ không được phép gửi thông báo tới cho URI lấy được ở trên bởi vì URI này không phải là URI của một ứng dụng windows store nào cả. Tất nhiên là vậy rồi, từ đầu tới giờ ta đâu đã cho hệ thống biết tên ứng dụng của chúng ta trên windows store, vậy thì làm sao WNS có thể cung cấp cho chúng ta một URI chính xác được?  
Để cung cấp định danh của ứng dụng rất đơn giản, ta chỉ cần click chuột phải vào tên project => Store => Associate app with store… và làm theo hướng dẫn.

### 2.2.3 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. Nhưng không phải mỗi lần gọi thi ta sẽ nhận được một đối tượng PushNotificationChanel chứa một URI khác với lần gọi trước. Một URI sẽ có thời gian tồn tại là 30 ngày. Vì vậy, khi ta gọi yêu cầu cấp một URI lần đầu tiên thì 30 ngày sau đó có thể URI sẽ không thay đổi, mà khi URI không đổi thì ta không cần thiết phải gửi chúng cho RNS. Tuy nhiên, con số 30 ngày đó chỉ là thời gian tối đa cho một URI tồn tại, một URI có thể hết hạn bởi một số nguyên nhân khác do WNS quyết định.  
Vì vậy, ta cần yêu cầu cấp một URI mới mỗi khi ứng dụng được mở và lưu URI đó ngay tại thiết bị. Lần sau khi nhận được một URI khác, ta sẽ kiểm tra xem chúng có giống nhau không, nếu khác thì mới gửi 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, hệ thống phải được thiết kế sao cho khi ứng dụng gửi một URI mới lên RNS thì Server phải biết URI đó là của thiết bị nào để rồi Update lại URI chứ không phải tạo một bản ghi mới. Có nghĩa là khi gửi URI lên cho RNS, ta phải gửi kèm theo thông tin của thiết bị như mã máy, đồng thời, ta phải gửi cả thông tin về hệ điều hành lên cho RNS vì với mỗi hệ điều hành khác nhau thì có mỗi máy chủ thông báo khác nhau, ngay cả cùng một hệ điều hành nhưng khác phiên bản cũng có vài điều khác nhau. Vì vậy, để đảm bảo tính tương thích tối đa, ta cần gửi đầy đủ các thông tin cần thiết về phần cứng lẫn phần mềm mà máy đang chạy 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. Tuy nhiên, phương thức gì thì cũng phải đảm bảo được độ bảo mật nhất định đối với dữ liệu. Để an toàn, nên mã hóa dữ liệu trước khi chuyển đi.

Tuy nhiên, có một điều khiến không ít người thắc mắc rằng: liệu trong khi một URI hết hạn mà ứng dụng không được mở lên để có thể lấy một URI mới mà RNS lại có một thông báo muốn gửi đến thì sao? Liệu thông báo đó có bị thất lạc vĩnh viễn?  
 Nếu người dùng không mở ứng dụng lên ít nhất 1 lần trong 30 ngày, họ sẽ không nhận được bất kỳ thông báo nào bởi vì URI đã hết hạn. Để hạn chế trường hợp này, ta phải làm một tác vụ chạy nền để yêu cầu một URI sau một khoảng thời gian nhất định.

### 2.2.4 Bước 3: Gửi thông báo cho WNS

Như đã đề cập ở phần Các vấn đề bảo mật với mô hình của WNS, bất kỳ một máy chủ nào muốn gửi một thông báo tới cho WNS đều phải xác thực danh tính trước với một mật khẩu được gọi là “thẻ truy cập” hay còn gọi là Token. Trong phần này, chúng ta sẽ tìm hiểu cách để lấy được token giúp xác nhận danh tính với WNS sau đó gửi các thông báo tới cho WNS.

1. **Lấy token từ WNS** Để có được thẻ truy cập từ WNS, ta phải gửi mã của ứng dụng trên windows store (Package SID) và mật khẩu của nó (Client secret) cho WNS, gói tin trả về sẽ chứa mật khẩu dùng để gửi kèm theo mỗi thông báo tới WNS. Package SID và Client secret có thể được lấy từ trang của ứng dụng trên <https://dev.windows.com/>.  
   Bước này phải có một tài khoản developer, sau đó thêm một ứng dụng vào danh sách các ứng dụng (chưa cần tải ứng dụng lên). Một trang của ứng dụng sẽ được tạo và ta có thể vào đó để lấy thông tin cần thiết cho việc xác thực danh tính với WNS.  
   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.

Lưu ý: Mỗi ứng dụng có một Package SID và Client Secret riêng cho nên chúng sẽ có một mã truy cập khác nhau. Để xác định ứng dụng là một phiên bản của một ứng dụng nào đó trên store, ứng dụng phải có một giá trị tĩnh được gán vào nó trong file AppManifest.xml gọi là Application identity, giá trị này có thể lấy được trên trang của ứng dụng tại <http://dev.windows.com>. Nếu một máy chủ nào muốn gửi thông báo cho ứng dụng A mà dùng thẻ truy cập của một ứng dụng B thì sẽ không được chấp nhận.

Định dạng http request của gói tin yêu cầu thẻ truy cập 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 thông báo.

Ta có thể dùng đoạn Code ngắn sau để lấy thẻ truy cập 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 thẻ truy cập, RNS sẽ sẵn sàng để gửi thông báo tới cho WNS. Tuy nhiên thẻ truy cập sẽ hết hạn và khi đó, RNS phải dùng phương thức trên để lấy thẻ truy cập mới. Để biết khi nào thẻ truy cập hết hạn, ta phải xem header của mỗi gói tin trả về khi gửi một thông báo tới WNS. Vấn đề này sẽ được làm rõ hơn ở phần tiếp theo: Gửi thông báo cho WNS.

1. **Gửi thông báo cho WNS**

Để gửi một thông báo cho WNS, RNS sẽ tạo một yêu cầu HTTP POST tới URI nhận được từ ứng dụng. Yêu cầu này phải chứa các header và nội dung cần thiết cho thông báo. Header authorization phải chứa thẻ truy cập để dùng cho việc xác thực danh tính với WNS.

Với các loại thông báo khác như title, toast và badge thì phần nội dung của chúng phải tuân theo một cấu trúc nhất định. Tuy nhiên như đã đề cập, chúng ta chỉ quan tâm tới raw notification bởi vì có thể tự tạo các notification loại khác bằng dữ liệu có được trong raw notification. Raw notification không có quy định về cấu trúc cho phần nội dung, toàn bộ phần nội dung sẽ được chuyển cho ứng dụng và ứng dụng sẽ quyết định xử lí dữ liệu đó như thế nào. Dưới đây là một đoạn http dùng để gửi một raw notification

|  |
| --- |
| POST https://cloud.notify.windows.com/?token=AQE%bU%2fSjZOCvRjjpILow%3d%3d HTTP/1.1Content-Type: application/octet-streamX-WNS-Type: wns/rawX-WNS-Cache-Policy: cacheX-WNS-RequestForStatus: trueAuthorization: Bearer <Token>Host: cloud.notify.windows.comContent-Length: <Length><body> </body> |

Trong đó, <Token> là thẻ truy cập, <Length> là độ dài của thông tin phía dưới. Thông tin về các header quan trọng trong một gói tin gửi cho WNS được liệt kê dưới đây:

|  |  |  |
| --- | --- | --- |
| **Header name** | **Required/Optional** | **Description** |
| Authorization | Required | Là một header HTTP chuẩn 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 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 thông báo. |
| 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 cần 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. |

### 2.2.5 Bước 5 Nhận thông tin trả về từ WSN sau khi gửi thông báo

Việc nhận thông tin trả về là việc cần làm ngay sau khi gửi thông báo. Tuy nhiên, việc này có một số điểm khá phức tạp nên việc tách ra khỏi bước 3 là cần thiết.

Sau khi gửi thông báo, WNS sẽ gửi trả lại cho RNS một sô thông tin liên quan tới trạng thái hiện tại của thiết bị và của thông báo. Ta có thể sử dụng những thông tin này để xác định thiết bị đó có còn có thể nhận thông báo được nữa không hay phải gửi lại thông báo này…

Dưới đây là các 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. |

Việc xác định đây là tạm thời ngắt kết nối hay bị mất kết nối thực sự được quyết định bởi Notification Client Platform. Việc của developer chỉ là đọc thông tin này và quyết định các ứng xử tiếp theo với thiết bị đó (có nên gửi tiếp thông báo cho thiết bị đó hay không).

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. |

Mã trả về: Mỗi gói tin HTTP sẽ chứa một mã trả về. 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. |

### 2.2.6 Bước 6 Nhận thông báo ở Client

Sau khi thông báo được WNS nhận từ RNS, nó sẽ được chuyển tới cho ứng dụng. Tới bước này, ta phải có một thành phần nào đó giúp ứng dụng nhận được thông báo cũng như liên lạc lại với RNS để lấy nội dung thông báo thực sự và tất cả các công việc đối với Notification. Ta gọi thành phần đó là Interface For Mobile Client Application (IFMCA) và từ đây trở đi, ta có thể xem IFMCA chính là ứng dụng mobile.

Đây là lúc cần thao tác để đón lấy thông báo tới. Có hai cách để IFMCA nhận được raw notification:

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

IFMCA có thể sử dụng cả hai cách trên để nhận thông báo. Trong trường hợp đó và nếu IFMCA đang chạy thì thông báo sẽ được đưa tới cho sự kiện nhận thông báo trước và sẽ được chuyển xuống cho tác vụ nền sau. Tuy nhiên tại sự kiện nhận thông báo, ta có thể quyết định rằng tác vụ nền có được chạy hay không. Dưới đây là vài điểm thiết yếu mà ta cần nắm được ở mỗi cách.

1. **Sự kiện thông báo**

Trong ứng dụng, ta có thể dễ dàng dùng một phương thức để xử lí sự kiện này bằng cách đăng ký sự kiện PushNotificationReceived của đối tượng PushNotificationChannel mà ta dùng để nhận thông báo. Trong phương thức xử lí sự kiện, ta có thể nhận được tất cả các thông tin cần thiết về thông báo trong tham số PushNotificationReceivedEventArgs.  
Tuy nhiên, sự kiện này chỉ có thể được kích hoạt khi ứng dụng đang chạy. Nếu ứng dụng không chạy thì chỉ có tác vụ nền được gọi.

1. **Tác vụ nền**

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.  
Tuy nhiên trên ứng dụng UWP, ta có thể hiểu rằng tác vụ chạy nền là một loại tác vụ chạy ngầm dưới hệ thống, bạn không thể tắt nó bằng task manager và không thể nhìn thấy việc nó hoạt động cũng như người dùng không thể biết được tên của nó. Để đơn giản, ta có thể xem tác vụ nền là một khối các câu lệnh, khối lệnh đó được ứng dụng đăng ký với hệ thống rằng nó sẽ chạy mỗi khi có một sự kiện nào đó trên hệ thống xảy ra, ví dụ như nhận một thông báo mới, đến một thời điểm nào đó hay internet khả dụng. Khi sự kiện đó xảy ra, đoạn code đó sẽ được chạy và các thông tin cần thiết về sự kiện cũng được hệ thống cung cấp cho đoạn code đó.  
Các tác vụ nền được quản lí chặt chẽ hơn và bị giới hạn rất nhiều nhằm giúp việc kiểm soát hệ thống tốt hơn. Cụ thể những giới hạn đó như sau:

* + Một tác vụ nền phải thuộc một ứng dụng nào đó, có nghĩa là ứng dụng phải là đối tượng đứng ra đăng ký cho tác vụ nền đó. Thực chất, tác vụ nền không còn là một tiến trình độc lập như trên Windows desktop nữa mà đã trở thành một chức năng của ứng dụng và chức năng đó được kiểm soát bởi hệ điều hành.
  + Khi đăng ký một tác vụ chạy nền, ứng dụng phải thông báo trước với hệ thống rằng chúng chạy nền để làm việc gì. Hệ điều hành sẽ đưa ra một số mục đích cho việc chạy nền và ứng dụng sẽ phải chọn một trong số chúng, các mục đích được chọn cũng chính là điều kiện để các tác vụ chạy nền được chạy.
  + Một tác vụ chạy nền được chạy bắt đầu từ một hàm có tên Run.  
    hàm này được chứa trong một lớp mà thực thi interface IBackgroundTask. Để có thể thực thi được interface này, class đó phải được chứa trong một project loại Windows Runtime Component. Và đương nhiên, project đăng ký phải Reference tới project chứa tác vụ nền.
  + Muốn chạy tác vụ nền, ứng dụng phải xin user quyền được chạy tác vụ nền.

Trên windows 8 hoặc 8.1, ứng dụng chỉ có quyền đó khi trở thành một ứng dụng màn hình khóa. Tuy nhiên ở Windows 10, ta chỉ cần gọi một phương thức để xin user quyền được chạy nền:  
BackgroundExecutionManager.RequestAccessAsync();

Phương thức trên sẽ trả về một enum BackgroundAccessStatus. Enum này có bốn giá trị:

|  |  |  |
| --- | --- | --- |
| Giá trị | Mô tả | Nguyên nhân |
| AllowedMayUseActiveRealTimeConnectivity | Cho phép nhưng có hạn chế khi đang sử dụng chế độ chờ kết nối | Người dùng chọn “Yes” nhưng hệ thống hạn chế quyền hạn trong chế độ chờ kết nối vì ứng dụng không đăng ký realtime connection broker (RTC). |
| AllowedWithAlwaysOnRealTimeConnectivity | Cho phép | Người dùng chọn “Yes” |
| Unspecified | Không xác định | Người dùng không chọn và tắt thông báo đi. |
| Denied | Không cho phép | Người dùng chọn “No”. |

Khi gọi phương thức này trên Windows 8 hoặc 8.1, một thông báo xin cấp quyền sẽ được hiển thị. Còn ở Windows 10, sẽ không còn thông nào hiện ra nữa.  
Thay vào đó, ứng dụng sẽ luôn nhận được quyền AllowedMayUseActiveRealTimeConnectivity.

Chú thích: có thể hiểu chế độ chờ kết nối là một loại khác của chế độ sleep hay hibernate. Khác biệt ở chỗ là chế độ này vẫn duy trì tất cả các tiến trình nhưng ở một mức độ tiêu thụ năng lượng cực kì thấp. Chế độ này giúp kéo dài thời lượng sử dụng pin hơn và thời gian để đánh thức thiết bị cũng nhanh hơn rất nhiều so với sleep hay hibernate truyền thống. Thời gian đánh thức thiết bị từ chế độ Connected Standby khoảng 500 mili giây. Tuy nhiên, chế độ này chỉ khả dụng trên các thiết bị đời mới cao cấp có hệ thống được thiết kế tiết kiệm năng lượng. PC lúc đó sẽ hơi lai giữa tablet và Laptop (Dòng Surface pro hoặc Surface book là một ví dụ).

1. **Lựa chọn sử dụng giữa hai phương thức**

Bây giờ ta có một cách mà có thể nhận được thông báo mọi lúc như background task thì tại sao ta lại cần cách còn lại trong khi nó chỉ nhận được thông báo khi ứng dụng đang chạy? Thực ra, cách thứ hai là thực sự cần thiết trong một số trường hợp.  
Ví dụ: Ta có một ứng dụng, mỗi khi nhận thông báo tới thì sẽ hiển thị một thông báo toast lên màn hình. Tuy nhiên, khi người dùng đang làm việc với ứng dụng thì việc hiển thị một thông báo toast trên chính ứng dụng đó trở nên khá vô duyên. Thay vào đó, ta có thể hiển thị một thông báo nhỏ khác bên trong giao diện của ứng dụng thì sẽ tốt hơn nhiều. Trong trường hợp này, ta phải nhận thông báo qua cả hai cách: sự kiện và tác vụ nền. Khi sự kiện nhận được thông báo thì nó sẽ hiển thị thông báo trong giao diện của ứng dụng. Sau khi hoàn thành công việc, nó sẽ hủy sự kiện để không cho tác vụ nền có thể nhận được thông báo nữa bằng câu lệnh đơn giản sau:  
e.Cancel = true  
Trong đó, e là đối tượng PushNotificationReceivedEventArgs được cung cấp trong sự kiện.

## 2.3 CÁC XỬ LÍ PHỤ

Trong phần 2.2: làm việc với WNS, Chúng ta gần như đã có thể gửi trọn vẹn một thông báo tới cho WNS và nhận thông báo đó tại Client.  
Tuy nhiên, còn vài vấn đề nhỏ nhưng không thể không lưu ý tới.  
Phần này sẽ đưa ra các lưu ý nhỏ, giúp lập trình cho hệ thống hoạt động hiệu quả và đúng đắn hơn.

### 2.3.1 Gửi cho thiết bị hay gửi cho người dùng

Ví dụ trong một hệ thống quản lí sinh viên, hệ thống có tác vụ báo điểm cho sinh viên qua thông báo. Khi thực hiện tác vụ đó, căn bản thì máy chủ sẽ gửi thông báo tới một danh sách các thiết bị. Nhưng mỗi sinh viên có thể có nhiều hơn 1 thiết bị nên việc đó thực chất là gửi cho một danh sách các sinh viên. Vậy, nếu server gửi một thông báo dạng tin nhắn hội thoại hoặc tương tự như vậy thì ở phía client, khi nhận được thông báo nó phải kiểm tra được tin nhắn đó đã được đọc bởi một thiết bị nào khác chưa. Nếu đã đọc rồi thì chỉ cập nhật dữ liệu nhưng không hiển thị thông báo, tránh làm phiền người dùng.  
Điều này tuy nhỏ nhặt nhưng rất nhiều ứng dụng lớn mắc phải và khiến người dùng rất khó chịu.  
 Ví dụ như ứng dụng Facebook trên Windows 10. Khi có một tin nhắn mới, cho dù người dùng đã đọc được chúng trên web hoặc trên một thiết bị khác rồi, nhưng lát sau lại thấy một thông báo về tin nhắn đó trên điện thoại, thật hết sức ngớ ngẩn.

Vậy làm sao để biết tin nhắn đó đã được đọc hay chưa?  
Giải pháp rất đơn giản: khi nhận được thông báo đó, IFMCA phải liên hệ lại với server để xác nhận xem thông báo đó đã có ai đọc chưa. Để làm được như vậy thì trước hết, mỗi khi IFMCA xác nhận rằng người dùng đã đọc qua thông báo đó, nó sẽ báo cáo lại với RNS rằng người dùng đã đọc thông báo đó rồi.

### 2.3.2 Xác nhận đã nhận được thông báo

Theo mô hình gửi thông báo của WNS, RNS không thể biết được liệu MCAI đã nhận được thông báo mình gửi đi hay chưa.  
Giả sử RNS gửi đi một thông báo, WNS trả về cho RNS một gói tin trong đó cho thấy rằng thiết bị đang ở trạng thái connected và sẵn sàng nhận thông báo. Nhưng chúng ta không thể chắc chắn rằng MCAI đã nhận được thông báo, bởi vì có thể ngay khi WNS xác nhận với RNS rằng thiết bị đang connected thì thiết bị mất kết nối ngay sau đó, WNS không thể gửi thông báo tới cho MCAI. Một trường hợp khác rõ ràng hơn hết là trong gói tin trả về cho thấy thiết bị đang trong trạng thái tempconneted.  
 Để biết được thiết bị đã nhận được thông báo hay chưa, chỉ có một cách đó là khi nhận được thông báo, MCAI phải liên hệ trực tiếp lại với RNS để nhắn với RNS rằng thông báo đó đã được chuyển tới nơi. Tuy nhiên, Microsoft có khuyến cáo rằng : các thông tin nhạy cảm cao không nên chuyển bằng WNS bởi vì nó không an toàn bằng một kết nối trực tiếp tới RNS để làm việc.  
Vậy với những thông báo quan trọng và có tín nhạy cảm cao mà nhà phát triển nghĩ người dùng phải được thông báo tới thì không nên sử dụng hệ thống này để truyền dữ liệu. Vì vậy hệ thống này không cần thiết phải được thiết kế để đảm bảo được việc RNS có thể biết được thiết bị đã nhận được thông báo hay chưa.

# CHƯƠNG 3: THIẾT KẾ MODULE THÔNG BÁO TRÊN NỀN TẢNG .NET FRAMEWORK

## 3.1 MỤC TIÊU MÀ BẢN THIẾT KẾ HƯỚNG ĐẾN

Như đã đề cập ở phần mục tiêu đề tài, mục tiêu mà bản thiết kế hệ thống này cần đạt đến là:

* Đơn giản đối với các nhà phát triển muốn áp dụng module vào hệ thống có sẵn của họ
* Phù hợp với mọi loại hệ thống có sẵn nào, với bất cứ nền tảng nào
* Độ tin cậy cao
* Tốc độ hồi đáp cao cho server, tiết kiệm năng lượng cho thiết bị đầu cuối.

## 3.2 SƠ ĐỒ Ý TƯỞNG CÁC THÀNH PHẦN HỆ THỐNG

Để hiểu được một hệ thống, trước tiên ta cần tưởng tượng xem hệ thống đó như thế nào, gồm những thành phần gì đã. Ta sẽ xem xét các thành phần của một hệ thống có sẵn đã được thêm module thông báo tin tức. Để đơn giản, ta chỉ nhìn hệ thống theo một cách đơn giản nhất có thể.



Ta có ba gói thư viện dùng ở ba nơi khác nhau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tên gói | Chức năng | Nền tảng | Giao thức sử dụng | Định dạng mô tả dữ liệu |
| RNS | - Nhận yêu cầu gửi thông báo từ hệ thống, lưu trữ và gửi chúng tới IFMCA.  - Nhận yêu cầu từ MC. | .Net | TCP/IP , HTTPS (giao tiếp với WNS) | JSON,  XML |
| IFAES | - Giúp hệ thống hiện thời giao tiếp với RNS | .Net hoặc các nền tảng khác | TCP/IP | JSON |
| MCAI | Giúp store app nhận notification. | .Net và UWP Core hoặc các nền tảng khác. | TCP/IP | JSON |

Sơ đồ trên mô tả hệ thống một cách rất đơn giản:

* Phần mềm có sẵn sẽ dùng một gói thư viện được cung cấp để có thể liên lạc với RNS, qua đó có thể yêu cầu server này gửi thông báo tới IFMCA trên các thiết bị.
* Sau khi nhận được yêu cầu, RNS sẽ lưu thông báo xuống data base rồi nhờ dịch vụ chuyển thông báo của nhà phát triển hệ điều hành chuyển ID của thông báo tới các ứng dụng.
* IFMCA sẽ sử dụng một gói thư viện khác để nhận về được các thông báo này.
* Sau khi lấy được ID của thông báo, IFMCA liên hệ trực tiếp với RNS để yêu cầu lấy nội dung của thông báo.

Như ta đã thấy, có 2 module nằm không cùng máy với gói RNS. Nếu hệ thống có sẵn được viết bằng Java và họ muốn gửi thông báo tới thiết bị android thì hai gói đó phải được viết bằng Java.  
Vậy thì liệu chúng có thể giao tiếp được với gói RNS được viết trên .Net, chúng có hiểu được dữ liệu mà RNS gửi cho cũng như dữ liệu mà chúng gửi cho RNS? Đó cũng là lý do phải có một chuẩn mô tả dữ liệu chung giữa chúng – JSON.

JSON là một chuẩn dùng để mô tả dữ liệu dưới dạng văn bản tương tự như XML. Hiện tại, hầu hết các ngôn ngữ lập trình bậc cao đều có hỗ trợ các thư viện để làm việc với chuẩn này. Vì vậy, nếu ta dùng giao thức TCP/IP và chuẩn JSON thì sẽ đảm bảo được client viết trên bất kì nền tảng nào cũng có thể giao tiếp được với server.  
Thực ra, việc sử dụng giao thức TCP/IP và định dạng mô tả dữ liệu JSON cũng tương tự như viết thủ công một web service hay một ứng dụng WCF vậy. Tuy nhiên, Web service và WCF cần được một web server host nên khá cồng kềnh và tốn nhiều thời gian để áp dụng. Thay vào đó, việc sử dụng trực tiếp giao thức TCP/IP sẽ giúp module độc lập hơn với AES và quá trình cài đặt cũng đơn giản hơn nhiều. Cách tiếp cận này có nhược điểm là code phải viết nhiều hơn, phải cẩn thận trong từng dòng code vì lỗi xảy ra khi truyền dữ liệu qua mạng có rất nhiều rủi do.

## 3.3 THIẾT KẾ CƠ SỞ DỮ LIỆU

Sơ đồ lớp:



Giải thích sơ đồ lớp:

Có tổng cộng 4 lớp:

|  |  |  |  |
| --- | --- | --- | --- |
| Tên lớp | Mô tả | Chức năng | Ghi chú |
| Notification | Thể hiện cho một thông báo | Lưu trữ thông báo dưới dạng mảng byte | Kiểu dữ liệu là image bởi vì nó cũng là mảng byte |
| Receiver | Thể hiện cho một người nhận | Ánh xạ một người nhận thông báo từ hệ thống đã tồn tại | Người nhận có trường OldID có kiểu string được hệ thống đã tồn tại cung cấp, NewID là ID được cơ sở dữ liệu cấp. Cả hai đều là duy nhất nhưng để tốc độ tìm kiếm theo ID nhanh hơn, ta cần trường NewID là giá trị được cấp tự động với kiểu số nguyên. |
| Device | Thể hiện cho một thiết bị của người nhận. | Lưu trữ thông tin thiết bị bao gồm URI và Device ID. | Một người nhận có thể có nhiều thiết bị thuộc nhiều hệ điều hành khác nhau. |
| OperatingSystem | Thể hiện một hệ điều hành | Lưu trữ thông tin về hệ điều hành của thiết bị |  |

Sơ đồ dữ liệu quan hệ:



## 3.4 SƠ ĐỒ TỔNG QUÁT HỆ THỐNG HOÀN THÀNH

Qua hai bước tưởng tượng ra các thành phần của hệ thống và thiết kế cơ sở dữ liệu cho nó, ta đã có sơ đồ tổng quát cuối cùng của hệ thống trong thực tế.  
Đây là mô hình phức tạp nhất có thể xảy ra khi áp dụng module này vào một hệ thống có sẵn.



Ta có thể thấy, phần mềm trên hệ thống cũ có thể được viết và chạy ở bất cứ nền tảng nào. Vì vậy, đối với mỗi nền tảng khác nhau, ta cần viết một thư viện cho chúng để có thể giao tiếp được với server. Một khi AES Communicator nhận được yêu cầu, nó sẽ làm trực tiếp luôn, hoặc nếu yêu cầu có liên quan tới bộ phận khác thì nó sẽ gửi yêu cầu đó lên cấp cao hơn (Notification Server) để xử lí, rồi AES Communicator sẽ trả kết quả lại cho Client.

Về phía ứng dụng cài trên thiết bị của người dùng thì cũng tương tự như phần mềm trên hệ thống cũ. Với mỗi nền tảng, ta cần có một bộ thư viện để giúp chúng nhận thông báo và liên hệ lại với RNS để lấy nội dung thông báo. Bộ phận MC Communicator sẽ chịu trách nhiệm nhận các yêu cầu này từ phía ứng dụng, xử lí và trả kết quả lại cho Mobile App.

Và một điều quan trọng là tất cả các hoạt động mất nhiều thời gian đều phải được thực hiện bất đồng bộ nếu không muốn hệ thống gửi thông báo quá chậm hoặc khi người dùng ấn nút gửi thông báo thì phải đợi cho đến khi gửi thông báo xong mới làm việc tiếp được.

# CHƯƠNG 4: THIẾT KẾ CHI TIẾT CÁC THÀNH PHẦN CỦA HỆ THỐNG

## 4.1 THIẾT KẾ CHO RNS

RNS là trái tim của Module. Nó chính là nơi để gửi thông báo đi đến các dịch vụ chuyển thông báo. Hiệu suất của toàn bộ hệ thống nằm ở đây. Vì vậy, việc gửi thông báo bắt buộc phải sử dụng đa luồng với threadpool.



Thiết kế cho thấy, một Notification Server gồm 3 thành phần chính:

* Bộ phận giao tiếp với hệ thống cũ (AES Communicator)
* Bộ phận giao tiếp với IFMCA app (MC Communicator)
* Bộ phận gửi thông báo (Raw Notification Sender)

Trong đó, hai thành phần giao tiếp với client sử dụng ba thành phần:

* TCP Server: Mô tả một server liên lạc trên giao thức TCP/IP
* Serializer: thành phần này giúp đưa các đối tượng cần chuyển về dạng mảng byte. Từ mảng byte này ta mới đưa xuống giao thức TCP để gửi đi. Thành phần này cũng giúp ta chuyển từ mảng byte quay về đối tượng để khi nhận được dữ liệu từ giao thức TCP, ta có thể chuyển chúng quay lại các đối tượng được dùng trên hệ thống.
* DBInteractor: Thành phần này có nhiệm vụ tương tác với cơ sở dữ liệu. Để đơn giản và tăng độ tin cậy cho chương trình, em sử dụng LINQ to SQL để tương tác với cơ sở dữ liệu.

Thành phần cuối cùng Raw notification sender sử dụng một tập hợp các thư viện khác dùng để gửi thông báo tới các hệ điều hành. Khi cần gửi thông báo tới một URI, nó sẽ xem URI đó là của hệ điều hành nào rồi lấy gói dành cho hệ điều hành đó để gửi thông báo tới cho thiết bị. Thêm vào đó, việc gửi thông báo đi tiềm ẩn rất nhiều rủi do (không có internet, PackageSID hoặc SecretKey không chính xác, gửi quá nhiều thông báo cho một thiết bị…) nên các gói thư viện đó cũng phải có trách nhiệm thông báo lại kết quả xảy ra cho đối tượng gọi nó. Việc thông báo lại kết quả này không phải là return một giá trị nào đó, bởi vì nếu thiết kế như vậy, đoạn code gọi gửi thông báo đi sẽ rất lộn xộn và khó quản lí:

|  |
| --- |
| switch (OperatingSystem)  {  case Windows: {var result = WindowsSender.Send();  // xử lí cho kết quả nhận về của Windows  ...  }  case IOS: {var result = IosSender.Send();  // xử lí cho kết quả nhận về của IOS  ...  }  case Android: {var result = AndroidSender.Send();  // xử lí cho kết quả nhận về của android  ...  }  } |

Đoạn xử lí cho kết quả thực tế sẽ rất dài và chỉ một chi tiết sai sót nhỏ thôi thì cũng có thể ảnh hướng tới các đoạn code lân cận. Vả lại, nếu sau này có một lập trình viên nào đó muốn sử dụng lại chỉ mình gói thư viện dùng để gửi thông báo tới cho windows thì sao? Nếu anh ta không xử lí kết quả trả về thì sao?

Vì vậy, cách để thông báo kết quả tốt nhất là sử dụng delegate. Người gọi muốn tạo một đối tượng như WindowsSender thì phải cung cấp trước cho nó tất cả các phương thức dùng để xử lí cho các loại kết quả trả về (khoảng 5-6 loại tương ứng với 5-6 phương thức), và mỗi hệ điều hành sẽ có 1 file riêng để viết các phương thức đó ra. Như vậy, người dùng đối tượng đó bắt buộc phải xử lí các lỗi xảy ra, giúp code sạch đẹp hơn, có độ tin tưởng cao hơn.  
Dưới đây là một phần của phương thức khởi tạo của lớp dùng để gửi thông báo tới hệ điều hành windows:

|  |
| --- |
| internal Windows10RawNotificationSender (string packageSID, string secretKey,  EventHandler<Exception> internetErrorcOccurred,  EventHandler<DelegatesAndEnums.BadURIEventArgs> badURIErrorOccurred,  EventHandler<Exception> notAccecptableErrorOccurred,  EventHandler<Exception> unknowErrorOccurred,  EventHandler<DeviceNotification> sendSuccessed,  EventHandler<Exception> wrongPackageSIDOrSecretKey)  {  PackageSID = packageSID;  SecretKey = secretKey;  InternetOrFirewallErrorOccurred = internetErrorcOccurred;  BadURIErrorOccurred = badURIErrorOccurred;  NotAccecptableErrorOccurred = notAccecptableErrorOccurred;  UnknowErrorOccurred = unknowErrorOccurred;  SendSuccessed = sendSuccessed;  WrongPackageSIDOrSecretKey = wrongPackageSIDOrSecretKey;  } |

Tuy nhiên, do thời gian có hạn nên em chỉ viết được thư viện gửi thông báo tới cho Windows. Còn các hệ điều hành khác nếu có thời gian, em sẽ bổ sung sau.

## 4.2 THIẾT KẾ CHO AES COMMUNICATOR VÀ SERVER INTERFACE

AES Communicator và Server Interface là hai thành phần giúp cho hệ thống cũ có thể liên lạc với RNS. Khi muốn gửi một thông báo tới một người hay một nhóm người trong hệ thống cũ thì hệ thống cũ phải đưa ra hai thứ: Gửi cho những ai và gửi cái gì.

Dưới đây là bản thiết kế cho AES Communicator và Server Interface:



Ta thấy rằng: AES Communicator có hai TCP server:

* TCP Server: dùng để nhận yêu cầu, xử lí và trả về kết quả cho client.
* Notify TCP Server: dùng để gửi thông tin về server mà client cần cập nhật, ví dụ như một lỗi xảy ra ở server.

Tương ứng với hai TCP Server, phía client cũng có hai TCP client dùng để giao tiếp với server. Cả hai bên đều dùng một Serializer và cũng dùng chung các gói tin để giao tiếp với nhau.

Về phía hệ thống cũ, Server Interface cung cấp cho hệ thống cũ một Interface có tên IReceiver. Interface này chỉ chứa duy nhất một phương thức: GetReceiverID().  
Interface này giúp module lấy được ID của người nhận trên hệ thống cũ. Qua đó có thể ánh xạ được user của hệ thống cũ qua cơ sở dữ liệu của module này. Ví dụ khi hệ thống cũ muốn gửi thông báo cho một người a thuộc lớp A thì lớp A bắt buộc phải thực thi interface IReceiver. Và bên trong hàm GetReceiverID (), phải trả về được một ID duy nhất cho đối tượng đó. Nếu ID trả về không duy nhất thì việc gửi thông báo có thể hoạt động không đúng.

Ngoài ra, Server Interface còn cung cấp các lời gọi hàm để hệ thống cũ có thể giao tiếp được với server như:

* Thêm thông báo
* Gửi thông báo

. Cùng với đó là các sự kiện được bắn ra như:

* Trạng thái server thay đổi.
* Mất kết nối tới server.
* Nhận được gói tin mới từ server

## 4.3 THIẾT KẾ CHO MC COMMUNICATOR VÀ MCAI

Đây là hai phần giúp mobile app có thể liên hệ lại với server để lấy dữ liệu thực sự của thông báo. MC Communicator sẽ là server còn MCAI là client.  
Do tính đặc thù là làm việc dưới nền và không đặt cao yêu cầu về độ trễ nên việc thiết kế bất đồng bộ cho MCAI là hoàn toàn không cần thiết. Bên cạnh đó, việc thiết kế bất đồng bộ sẽ làm tốn năng lượng của thiết bị hơn là đơn luồng. Nếu người dùng thấy ứng dụng của ta sử dụng quá nhiều năng lượng, nguy cơ bị gỡ bỏ là rất cao. Dưới đây là bản thiết kế các thành phần cho hai module nói trên, tuy nhiên bên Client ở mỗi hệ điều hành khác nhau sẽ có những cách thức làm việc ở background khác nhau, dẫn đến bản thiết kế của hệ điều hành này chưa hẳn đã làm việc được trên các nền tảng khác. Vì vậy, dưới đây chỉ là bản thiết kế cho client là hệ điều hành windows 10:



Trong sơ đồ, hai bên server – client giao tiếp với nhau qua giao thức TCP/IP bằng hai thành phần hỗ trợ cho giao thức này (TCP server cho server và UWP TCP Client cho client chạy trên nền tảng UWP Core (Windows 10)). Ngoài ra, còn có 2 tác vụ nền khác:

* Raw Receiver có nhiệm vụ chạy nền để nhận các thông báo từ phía máy chủ thông báo của hệ điều hành, khi nào có thông báo mới thì gửi chúng cho MC communicator để nó lấy nội dung thông báo trực tiếp với server.
* Refresh URI có nhiệm vụ chạy nền để làm mới URI của máy, nếu URI thay đổi, nó sẽ đưa URI mới đó cho MC communicator để nó cập nhật URI lên server.

Tất cả các gói trên đều là ẩn đối với mobile app và nó không cần quan tâm tới. Mobile app chỉ cần làm việc với một thành phần duy nhất là Mobile Interface. Mobile app cần phải cung cấp các thông tin cần thiết khác cho Mobile Interface như:

* ID của người dùng trên hệ thống đã tồn tại
* IP hoặc tên miền của máy chủ
* Port của máy chủ.

Chỉ cần cung cấp nhiêu đó, tất cả các thứ còn lại, Mobile Interface sẽ lo. Cho dù đây là bản thiết kế cho client là windows 10 nhưng nếu muốn phát triển gói dành cho các nền tảng khác cũng có thể dựa trên mẫu này để đi tới chi tiết hóa bởi vì chúng luôn giống nhau về mặt tư tưởng.

# CHƯƠNG 5: CÔNG CỤ PHÁT TRIỂN, CÀI ĐẶT VÀ KIỂM THỬ

## 5.1 CÔNG CỤ PHÁT TRIỂN

Do được phát triển trên C# nên công cụ phát triển 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 pro (vì từ bản pro trở lên mới hỗ trợ công nghệ ảo hóa Hyper -V).  
Framework được sử dụng là .Net framework mới nhất: .Net 4.6.  
Hệ quản trị cơ sở dữ liệu được sử dụng là Microsoft SQL Server 2014.

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

Do đây không phải là một phần mềm mà là một module của phần mềm cho nên việc cài đặt nó không đơn giản.  
Như ta đã biết, module này gồm có 3 thành phần chính:

* RNS
* IFAES
* IFMCA

Ta phải cài đặt 3 thành phần này vào 3 nơi khác nhau:

Bước 1: Tạo một project có khả năng chạy được (đầu ra là file exe) và thêm RNS vào project đó.

Bước 2: Sử dụng class RawNotificationServer của RNS để tạo một TCP server.

Bước 3: Mở solution của dự án AES và thêm project AES Interface vào

Bước 4: Dùng class Server Interface của project AES Interface để gửi yêu cầu tới RNS.

Bước 5: Tạo một project ứng dụng cho windows 10 (UWP project) rồi thêm project IFMCA vào project mới đó.

Bước 6: Sử dụng MCAI Interface để nhận thông báo trả về từ server.

## 5.3 KIỂM THỬ

### 5.3.1 Mục tiêu kiểm thử

* + Kiểm tra tính đúng đắn
  + Kiểm tra thời gian hồi đáp có chấp nhận được không

### 5.3.2 Tiến hành kiểm thử

Tiến hành kiểm thử với việc gửi 1000 thông báo tới cho 2 thiết bị.  
2 thiết bị đó của cùng một người dùng.

AES và RNS được cài đặt trên cùng 1 máy tính. MCAI được cài đặt trên một thiết bị chạy windows 10 mobile.

### 5.3.3 Kết quả kiểm thử

|  |  |  |
| --- | --- | --- |
| STT | Cách kiểm thử | Kết quả |
| 1 | Tất cả có kết nối internet, sẵn sàng thực thi yêu cầu. | Thành công, 1000 thông báo được chuyển đi trong khoảng 3.5 giây. |
| 2 | MCAI không có kết nối internet | Thành công, 1000 thông báo được chuyển đi trong khoảng 3.5 giây và thông báo được nhận sau khi MCAI có kết nối internet trở lại. |
| 3 | RNS không có kết nối internet | Đúng như mong đợi, 1000 thông báo vẫn được lưu lại dưới cơ sở dữ liệu và chờ để đợt sau gửi lại. Không có lỗi xảy ra cho RNS. AES nhận được thông báo về lỗi từ RNS và xuất thông báo lên cho người dùng xem. |
| 4 | RNS không hoạt động | Đúng như mong đợi, AES không thể gửi yêu cầu tới cho RNS. Không có lỗi phát sinh trên AES. Thông báo kết nối thất bại hiển thị |
| 5 | Database ngừng hoạt động | Đúng như mong đợi, RNS không thể lưu thông báo xuống Database và lỗi được thông báo về lại cho AES. Thông báo dược hiển thị lên màn hình người dùng. |
| 6 | Gửi thông báo là một đoạn kí tự đặc biệt không có trong mã UTF-8 | Lỗi, Hệ thống vẫn gửi thông báo bình thường, nhưng sau khi MCAI liên hệ lại vời RNS và đã lấy về được nội dung thông báo dưới dạng byte [], MCAI chuyển từ các byte [] đó sang dạng object thì bị lỗi. Nguyên nhân là do API của .Net không hỗ trợ cho các đoạn ký tự không phải của UTF-8. Lỗi này sẽ sớm được khắc phục bằng cách sử dụng một API chuyển đổi object và byte[] khác. |

# CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 6.1 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 nền tảng .net
* Hỗ trợ các Mobile App được viết trên nền tảng .net hoặc UWP Core
* Đã cài đặt và gửi thông báo tới MCAI thành công.
* Toàn bộ các hoạt động có liên kết tới giao diện đều được thực hiện bất đồng bộ, không gây nên việc đứng giao diện. Đồng thời, do thực hiện bất đồng bộ nên việc gửi thông báo với số lượng lớn cũng trở nên nhanh chóng hơn nhiều.

## 6.2 NHỮNG HẠN CHẾ

Đề tài còn có những hạn chế sau:

* Không thể gửi được các đoạn văn bản có ký tự không thuộc bảng mã UTF-8
* Do thời gian có hạn nên em chỉ mới phát triển được Interface cho các hệ thống được phát triển trên .Net framework.

## 6.3 HƯỚNG PHÁT TRIỂN

Sắp tới, em sẽ tập trung vào các việc sau:

* Sửa lỗi kí tự không thuộc bảng mã UTF-8.
* Phát triển Interface cho các nền tảng khác, đặc biệt là Java.
* Tối ưu hiệu suất của Server

Sau khi hoàn tất các bước trên, em sẽ thử nghiệm trên một VPS như một dịch vụ hoàn toàn miễn phí nhưng có chèn quảng cáo.

# 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. Phần lớn các kiến thức em tổng hợp được không phải qua sách mà qua các trang web như <http://msdn.com>, <http://chanel9.com>, <http://stackoverflow.com>

Trong hai cuốn trên thì cuốn C# network Programming của Richard Blum là một cuốn rất đáng đọc. Được viết từ năm 2003 nhưng giá trị thì không hề thay đổi theo thời gian. Chính cuốn sách đó đã giúp em từ một người không biết gì về lập trình mạng trở thành một người khá tự tin trong lập trình mạng cũng như lập trình bất đồng bộ.