**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

 **KHOA CÔNG NGHỆ PHẦN MỀM**

**ĐỒ ÁN MÔN CÔNG NGHỆ .NET**

**Đề tài:**

**ASP.NET SIGNALR**

Giảng viên hướng dẫn: ThS. Phạm Thi Vương

Giảng viên thực hành : ThS. Trần Hạnh Xuân

Sinh viên thực hiện: Phạm Hoàng Hải Sơn 13520708

Tô Thành Thương 13520862

Nguyễn Văn Thuyền 13520873

***TP. Hồ Chí Minh, ngày 21 tháng 06 năm 2017***

**Lời Cảm Ơn**

Đầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến tập thể Thầy Cô Trường Đại học Công nghệ thông tin – Đại học Quốc gia TP.HCM và Thầy Cô khoa Công Nghệ Phần Mềm đã giúp cho nhóm chúng em có những kiến thức cơ bản làm nền tảng để thực hiện đề tài này.

Đặc biệt, nhóm chúng em xin gửi lời cảm ơn và lòng biết ơn sâu sắc nhất tới Thầy Phạm Thi Vương, cô Trần Hạnh Xuân. Đã giúp nhóm chúng em hoàn thành tốt báo cáo môn học của mình.

Trong thời gian một học kỳ thực hiện đề tài, nhóm chúng em đã vận dụng những kiến thức nền tảng đã tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới. Từ đó, nhóm chúng em vận dụng tối đa những gì đã thu thập được để hoàn thành một báo cáo đồ án tốt nhất. Tuy nhiên, trong quá trình thực hiện, nhóm chúng em không tránh khỏi những thiếu sót. Chính vì vậy, nhóm chúng em rất mong nhận được những sự góp ý từ phía các Thầy Cô nhằm hoàn thiện những kiến thức mà nhóm chúng em đã học tập và là hành trang để nhóm chúng em thực hiện tiếp các đề tài khác trong tương lai.

Nhóm em chân thành cảm ơn quý Thầy Cô!

NHẬN XÉT CỦA GIẢNG VIÊN

………………………………………………………………………………………………....

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

…………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………………………………………

**Mục lục**

[Chương I: Internet, bất đồng bộ, đa người dùng… wow! 1](#_Toc485669105)

[Chương II: Ứng dụng thời gian thực - Góc nhìn từ người dùng – developer 5](#_Toc485669106)

[2.1 Ứng dụng thời gian thực – góc nhìn từ người dùng 5](#_Toc485669107)

[2.2 Ứng dụng thời thực – góc nhìn từ developer 5](#_Toc485669108)

[2.3 Ứng dụng web thời gian thực 6](#_Toc485669109)

[Chương III: Các kỹ thuật để mô phỏng kết nối thời gian thực 7](#_Toc485669110)

[3.1 Hoạt động của HTTP 7](#_Toc485669111)

[3.2 Polling: Câu trả lời? 9](#_Toc485669112)

[3.3 Push: Server chủ động 11](#_Toc485669113)

[3.4 WebSockets 13](#_Toc485669114)

[3.5 Server-Sent Event (API Event Source) 15](#_Toc485669115)

[3.6 Pus today 17](#_Toc485669116)

[3.7 Long Polling 18](#_Toc485669117)

[3.8 Forever frame 19](#_Toc485669118)

[Chương IV: Giới thiệu SignalR 22](#_Toc485669119)

[4.1 SignalR là gì? 22](#_Toc485669120)

[4.2 SignalR và WebSocket 26](#_Toc485669121)

[4.3 Phương thức vận chuyển 27](#_Toc485669122)

[4.3.1 HTML 5 tranport 27](#_Toc485669123)

[4.3.2 Comet tranport 27](#_Toc485669124)

[4.3.3 Quá trình lựa chọn các cách thức kết nối của SignalR 28](#_Toc485669125)

[Chương V: Kiến trúc & các thành phần của SignalR 29](#_Toc485669126)

[5.1 Kiến trúc 29](#_Toc485669127)

[5.2 Thành phần của SignalR 30](#_Toc485669128)

[Chương VI: Các nền tảng được hỗ trợ 31](#_Toc485669129)

[6.1 Các yêu cầu hệ thống server 31](#_Toc485669130)

[6.2 Các hệ thống server được hỗ trợ 31](#_Toc485669131)

[6.3 Các phiên bản .NET Framework server được hỗ trợ 31](#_Toc485669132)

[6.4 Các phiên bản IIS server được hỗ trợ 31](#_Toc485669133)

[6.5 Yêu cầu hệ thống client 32](#_Toc485669134)

[6.6 Các trình duyệt web 32](#_Toc485669135)

[6.7 Các chương trình Window Desktop và Silverlight 33](#_Toc485669136)

[Chương VII: Triển khai & mở rộng SignalR 34](#_Toc485669137)

[7.1 Triển khai SignalR 36](#_Toc485669138)

[7.2 Mở rộng in SignalR 40](#_Toc485669139)

[Chương VIII: Demo SignalR 47](#_Toc485669140)

[8.1 Ứng dụng chat thời gian thực 47](#_Toc485669141)

[8.1.1 Giới thiệu 47](#_Toc485669142)

[8.1.2 Cài đặt Project 48](#_Toc485669143)

[8.1.3 Chạy ví dụ 53](#_Toc485669144)

[8.2 Ứng dụng bảng giá chứng khoán trực tuyến 56](#_Toc485669145)

[8.2.1 Giới thiệu 56](#_Toc485669146)

[8.2.2 Cài đặt project 56](#_Toc485669147)

[Tài liệu tham khảo 72](#_Toc485669148)

# Chương I: Internet, bất đồng bộ, đa người dùng… wow!

Một ứng dụng kết hợp Internet, không đồng bộ và nhiều người dùng tương tác cùng một lúc luôn xứng đáng là "wow!". Tại một thời điểm nào đó, chúng ta đã vô cùng ngạc nhiên trước sự tương tác mà một số hệ thống web hiện đại có thể cung cấp, chẳng hạn như Facebook, Twitter, Gmail, Google Docs, Office Web Apps,… nơi mà chúng ta nhận được nhiều thông tin mới hầu như trong thời gian thực mà không phải tải lại trang.

Ví dụ: khi chúng ta đang chỉnh sửa tài liệu trực tuyến bằng Office Web Apps và một người dùng khác cũng truy cập nó, chúng ta có thể thấy rằng họ đã nhập tài liệu và làm theo các thay đổi mà họ đang thực hiện. Ngay cả trong một tình huống hằng ngày hơn chẳng hạn như một cuộc trò chuyện trực tuyến đơn giản, các tin nhắn được gõ bởi bạn của chúng ta chỉ xuất hiện, như thể bằng ma thuật. Cả hai hệ thống sử dụng cùng một loại giải pháp: truyền dữ liệu không đồng bộ giữa máy chủ và máy khách trong thời gian thực.

Thế giới chắc chắn đòi hỏi tính trực tiếp này: người dùng cần phải biết ngay những gì đang xảy ra trong môi trường của họ, tài liệu họ đang làm việc, mạng xã hội, trò chơi trực tuyến của họ và ngày càng nhiều lĩnh vực trong cuộc sống hàng ngày của họ. Thay vì phải tìm kiếm thông tin như trước đây họ đã từng làm, bây giờ họ muốn thông tin đến với họ ngay khi nó được tạo ra.

Nhu cầu này rõ ràng ở cấp giao thức web một thời gian, bởi vì HTTP lâu dài, như được định nghĩa trong ngày, không thể đáp ứng chúng một cách hiệu quả. Trên thực tế, các tổ chức định nghĩa các tiêu chuẩn web và các giao thức, cũng như các nhà phát triển trình duyệt, đã nhận thức được điều này và đã làm việc trong nhiều năm với các cơ chế mới để truyền thông giữa máy khách và máy chủ theo hướng ngược lại với quy trình thông thường - nghĩa là, cho phép Máy chủ để chủ động trong truyền thông. Điều này đã được thực hiện thành các giao thức mới có thể được sử dụng với một mức độ tin cậy, mặc dù họ vẫn còn khá xa các giải pháp phổ quát. Sự đa dạng tuyệt vời của nền tảng khách hàng và máy chủ, Và ngay cả cơ sở hạ tầng mạng, làm cho việc áp dụng các cơ chế mới này trở nên khó khăn và chậm chạp. Sau đó, chúng ta sẽ nghiên cứu chi tiết về những khía cạnh này.

Tuy nhiên, đây không phải là vấn đề duy nhất phải được giải quyết khi phát triển các ứng dụng đa người dùng theo thời gian thực. Truyền thông, như chúng ta biết, tạo nên các biến không ổn định và không thể đoán trước, làm cho việc quản lý và phân phối các thông điệp đến người sử dụng khá phức tạp. Ví dụ: trong ứng dụng phòng trò chuyện, chúng ta có thể có người dùng với băng thông rất khác nhau được kết nối với cùng một phòng, và những băng thông đó có thể dao động trong suốt phiên trò chuyện. Để ngăn chặn sự mất mát các tin nhắn trong kịch bản này, máy chủ có thể lưu trữ chúng tạm thời, gửi chúng tới người nhận và theo dõi những người dùng đã nhận được chúng, luôn luôn có tính đến các điều kiện giao tiếp với từng người dùng và các sự cố có thể xảy ra. Có thể xảy ra trong khi giao hàng, gửi lại dữ liệu nếu cần. Theo một nghĩa nào đó, điều này rất giống với các tính năng mà chúng ta có thể tìm thấy trong các máy chủ SMTP truyền thống, nhưng với yêu cầu bổ sung về tính cấp thiết cần thiết của các hệ thống thời gian thực. Thật dễ dàng để hình dung sự phức tạp và khó khăn liên quan đến việc thực hiện một hệ thống như mô tả.

Cho đến gần đây, không có thành phần hoặc khuôn khổ trong lĩnh vực công nghệ .NET cung cấp bởi Microsoft có khả năng cung cấp một giải pháp hoàn chỉnh cho các vấn đề của việc áp dụng loại ứng dụng này. Chắc chắn, có rất nhiều công nghệ có khả năng cung cấp các dịch vụ kết nối và ngắt kết nối, chẳng hạn như các dịch vụ Web quen thuộc, WCF, hoặc Web API gần đây hơn. Tuy nhiên, không ai trong số chúng được thiết kế đặc biệt cho các môi trường không đồng bộ với sự cộng tác thời gian thực giữa nhiều người dùng. Trong thực tế, mặc dù nó có thể tạo ra loại hệ thống này với các loại đĩa như vậy, nhưng nó không phải là một công việc tầm thường ngay cả đối với các nhà phát triển có kinh nghiệm nhất và thường sản xuất ra những hệ thống không hiệu quả với nhiều vấn đề về hiệu năng và giới hạn khả năng mở rộng.

Trong các trang này, chúng ta sẽ học cách thực hiện các tính năng ấn tượng của loại này bằng cách sử dụng SignalR, một khung công tác mạnh mẽ, linh hoạt và có thể mở rộng - điều này sẽ tạo điều kiện cho nhiệm vụ của chúng ta trở nên tầm thường.

Với mục đích này, trước tiên chúng ta sẽ trình bày một bài đánh giá ngắn gọn về các vấn đề mà chúng ta tìm thấy khi phát triển các ứng dụng đa người dùng theo thời gian thực, một số trong đó chúng ta đã đề cập đến. Chúng tôi sẽ nhanh chóng xem xét các hoạt động HTTP và những hạn chế của nó để hỗ trợ các loại hệ thống, và chúng tôi sẽ giới thiệu khái niệm đẩy. Chúng tôi cũng sẽ mô tả các tiêu chuẩn hiện đang trong quá trình xác định bởi W3C và IETF, cũng như các kỹ thuật mà chúng tôi hiện có thể sử dụng để thực hiện đẩy trên HTTP. Điều này sẽ cho phép chúng ta hiểu sâu hơn về tình huống mà chúng ta đang làm việc và những thách thức xung quanh sự phát triển của các ứng dụng tự hào về tính tương phản và tính tương tác mà chúng tôi đã mô tả. Đổi lại, điều này sẽ giúp chúng ta hiểu rõ hơn về hoạt động của SignalR và cơ sở của nó như thế nào.

Tiếp theo, tôi sẽ chính thức giới thiệu SignalR, mô tả các tính năng chính của nó, vị trí của nó trong chồng công nghệ của Microsoft để phát triển web và các mức trừu tượng khác nhau mà nó cho phép qua các giao thức cơ bản và sẽ giúp chúng tôi tách biệt với các chi tiết cấp thấp hơn Để chúng tôi chỉ có thể tập trung vào việc tạo ra các tính năng ngoạn mục cho người dùng của chúng tôi. Chúng tôi cũng sẽ có cơ hội để nói về OWIN và Katana, hai đại lý mới ngày càng trở nên nổi bật trong các công nghệ khác nhau, bao gồm cả SignalR.

  Chúng ta sẽ nghiên cứu kỹ lưỡng các kỹ thuật và kỹ thuật khác nhau được cung cấp bởi khuôn khổ này để tạo ra các ứng dụng thời gian thực tương tác đa người dùng tương tác, cả trên máy khách và máy chủ, và chúng ta sẽ học cách tận dụng sức mạnh và tính linh hoạt của chúng. Đương nhiên, chúng ta sẽ cung cấp các ví dụ mã khác nhau để giúp chúng ta hiểu cơ sở của nó một cách thực tế và do đó minh hoạ cách chúng ta có thể sử dụng khung công việc này trong các dự án thực tế.

Chúng tôi cũng sẽ mô tả cách thức SignalR độc lập với môi trường web: mặc dù chúng dường như tạo thành môi trường tự nhiên của nó nhưng khung công tác này vượt xa chúng, cho phép cung cấp các dịch vụ thời gian thực từ bất kỳ loại ứng dụng nào, và do đó tiêu thụ của chúng từ thực tế Bất kỳ loại hệ thống nào. Chúng ta sẽ thấy một số ví dụ về điều này.

Một khía cạnh khác rất quan trọng, mà chúng tôi sẽ dành nhiều trang, đang xem xét triển khai và khả năng mở rộng của các ứng dụng SignalR. Chúng tôi sẽ nghiên cứu các công cụ "out-of-the-box" đi kèm với nền tảng này và chỉ ra các giải pháp khả thi khác khi giải quyết các kịch bản mà những công cụ như vậy không đủ mạnh. Ngoài ra, chúng tôi sẽ xem xét các kỹ thuật khác nhau được thiết kế để theo dõi trạng thái của các máy chủ của chúng tôi và cải thiện hiệu suất của chúng trong các môi trường đồng thời cao.

Cuối cùng, chúng ta sẽ đi sâu vào các khía cạnh tiên tiến của chương trình với SignalR, điều này sẽ cho chúng ta cái nhìn sâu sắc hơn về cách hoạt động của khuôn khổ, bao gồm bảo mật, tạo ra các thành phần tách rời bằng cách sử dụng tiêm phụ thuộc, khả năng mở rộng SignalR, kiểm tra đơn vị và các khía cạnh khác quan tâm.

Chào mừng bạn đến với nhiều ứng dụng không đồng bộ thời gian thực đa người dùng. Chào mừng bạn đến SignalR!

# Chương II: Ứng dụng thời gian thực - Góc nhìn từ người dùng – developer

## **2.1 Ứng dụng thời gian thực – góc nhìn từ người dùng**

* Người dùng muốn có thông tin mới nhất (dữ liệu)!
* Người dùng cần dữ liệu ngay!
* Thời gian thực (Real time)!
* Đó là một thực tại “ngày nay”!
* Ví dụ
  + Twitter / Facebook / …
  + Thông báo (Notifications)
  + Đấu giá / Kinh doanh chứng khoán / Ngân hàng
  + Collaborative apps (Google docs,…)

Thế giới chắc chắn đòi hỏi tính trực tiếp: người dùng cần phải biết ngay những gì đang xảy ra trong môi trường của họ, tài liệu họ đang làm việc, mạng xã hội, trò chơi trực tuyến của họ và ngày càng nhiều lĩnh vực trong cuộc sống hàng ngày của họ. Thay vì phải tìm kiếm thông tin như trước đây họ đã từng làm, bây giờ họ muốn thông tin đến với họ ngay khi nó được tạo ra.

## **2.2 Ứng dụng thời thực – góc nhìn từ developer**

* Kết nối liên tục giữa các điểm cuối.
* Truyền thông hai chiều (full-duplex).
* Độ trễ thấp.
* Chi phí thấp.

Các realtime web gần gũi nhất các bạn có thể thấy là facebook, gmail. Khi bạn nhận được 1 tin nhắn, 1 email hay 1 thông báo, gần như ngay lập tức trên giao diện sẽ hiển thị thông tin mà chúng ta không cần phải tương tác trước. Tức là không chỉ khi chúng ta gửi yêu cầu lên cho server thì server mới trả về, mà server sẽ tự biết khi nào cần gửi thông tin về cho chúng ta. Có thể coi đó như 1 quan hệ 2 chiều.

## **2.3 Ứng dụng web thời gian thực**

* Kẻ thù tồi tệ nhất của chúng ta: **HTTP**
  + Không bao giờ được thiết kế cho kết nối thời gian thực.
  + Mô hình kết nối: Yêu cầu - Phản hồi.
  + Web is stateless???

Web xưa nay vẫn được chạy trên giao thức http, tức là người dùng (client) sẽ gửi một yêu cầu, server sẽ nhận yêu cầu, xử lý sau đó trả vể kết quả cho client. Chúng ta coi nó như 1 quan hệ 1 chiều.

# Chương III: Các kỹ thuật để mô phỏng kết nối thời gian thực

**HTTP (HyperText Tranfer Protocol)** là “ngôn ngữ” trong đó client và server của một ứng dụng Web có thể tương tác với nhau. Nó được định nghĩa vào năm 1996, tính đơn giản và tính linh hoạt của nó ở mức độ nào đó, chịu trách nhiệm cho sự thành công và mở rộng của các trang web và toàn thể Internet.

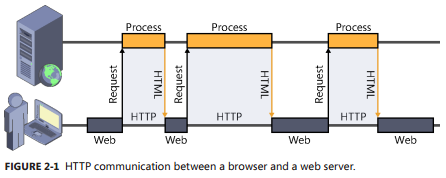
Mặc dù nó vẫn có giá trị trong các kịch bản web truyền thống, có những điểm khác, chẳng hạn như các ứng dụng thời gian thực hoặc các dịch vụ, mà nó là khá hạn chế.

## **3.1 Hoạt động của HTTP**

Một hoạt động của HTTP dựa trên một giản đồ phản hồi yêu cầu, luôn được khởi tạo bởi client. Thủ tục này thường được gọi là mô hình pull: Khi client cần truy cập vào một tài nguyên được lưu trữ bởi một server, nó sẽ bắt đầu kết nối với nó và yêu cầu thông tin mong muốn sử dụng "ngôn ngữ" được định nghĩa bởi giao thức HTTP. Server xử lý yêu cầu này, trả về tài nguyên đã được yêu cầu (có thể là nội dung của một tệp hiện có hoặc kết quả của quá trình chạy) và kết nối ngay lập tức đóng lại.

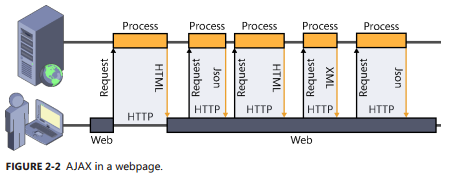
Nếu client cần có một tài nguyên mới, quá trình bắt đầu lại từ đầu: một kết nối đến server được mở ra, yêu cầu cho tài nguyên được gửi, server xử lý nó, nó trả về kết quả, và sau đó kết nối được đóng lại. Điều này xảy ra mỗi khi chúng ta truy cập trang web, hình ảnh hoặc tài nguyên khác do trình duyệt tải xuống, để đặt tên cho một vài ví dụ.

Như bạn có thể đoán bằng cách nhìn vào hình 2-1, nó là một quá trình đồng bộ: sau khi gửi yêu cầu đến server, client là để chờ đợi, không làm gì cho đến khi đáp ứng có sẵn.



Mặc dù hoạt động này là một hệ thống web cổ điển nhưng bản thân giao thức HTTP có thể hỗ trợ các yêu cầu cho sự không đồng bộ của các ứng dụng hiện đại do các kỹ thuật thường được gọi là AJAX (Asynchronous JavaScript and XML).

Sử dụng kỹ thuật AJAX, việc trao đổi thông tin giữa client và server có thể được thực hiện mà không cần rời khỏi trang hiện tại. Tại bất kỳ thời điểm nào, như thể hiện trong hình 2-2, client có thể bắt đầu một kết nối đến máy chủ bằng cách sử dụng JavaScript, yêu cầu một tài nguyên và xử lý nó (ví dụ, cập nhật một phần của trang).

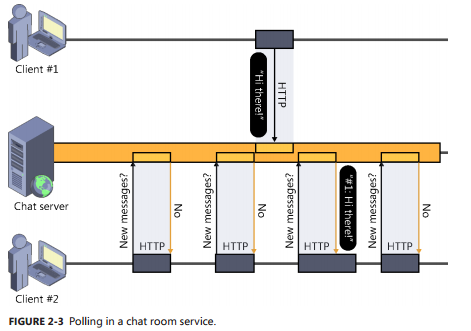
Điều thực sự thuận lợi và đã góp phần vào sự nổi lên của các dịch vụ rất năng động và tương tác, chẳng hạn như Facebook hay Gmail, là các hoạt động này được thực hiện không đồng bộ - có nghĩa là người dùng có thể tiếp tục sử dụng hệ thống trong khi đó liên lạc với server để gửi hoặc nhận thông tin.

Sơ đồ hoạt động này tiếp tục sử dụng và tuân thủ giao thức HTTP và mô hình phản hồi yêu cầu của client. Client luôn luôn là chủ động, quyết định khi nào kết nối với server. Tuy nhiên, có những tính huống trong đó HTTP không phải là rất hiệu quả. Với giao thức này, không dễ thực hiện các ứng dụng nhắn tin nhanh hoặc phòng chat, công cụ cộng tác, trò chơi trực tuyến nhiều người hoặc các dịch vụ thông tin thời gian thực, ngay cả khi sử dụng không đồng bộ.

Lý do rất đơn giản: HTTP không được định hướng theo thời gian thực. Có các giao thức khác, chẳng hạn như IRC phổ biến, thực sự tập trung vào việc đạt được giao tiếp nhanh hơn để cung cấp các dịch vụ năng động và tương tác hơn so với những gì chúng ta có thể thu được bằng cách sử dụng pull. Trong đó, server có thể chủ động và gửi thông tin cho client bất cứ lúc nào mà không cần chờ client yêu cầu nó một cách rõ ràng.

## **3.2 Polling: Câu trả lời?**

Là một nhà phát triển web, khi chúng ta phải đối mặt với một tình huống mà trong đó chúng ta cần server là một trong những thông tin gửi tới client một cách tự động tự chủ, giải pháp đầu tiên mà trực quan hóa đến suy nghĩ của chúng ta là sử dụng kỹ thuật được gọi là **Polling**. **Polling** về cơ bản bao gồm việc tạo các kết nối định kỳ từ client để kiểm tra xem có bất kỳ cập nhật nào có liên quan tại server như thể hiện trong hình 2-3.



Các ưu điểm chính của giải pháp này là: Trước tiên, việc triển khai dễ dàng và thứ hai là ứng dụng phổ cập của nó: nó hoạt động trong mọi trường hợp, với tất cả các trình duyệt và với tất cả các server, bởi vì nó không có gì hơn là sử dụng các tính năng tiêu chuẩn của HTTP. Và tất nhiên, chúng ta vẫn sử dụng mô hình pull.

Tuy nhiên, đôi khi giá Polling quá cao. Kết nối liên tục và ngắt kết nối có chi phí cao về băng thông và xử lý ở cả hai đầu của truyền thông. Phần tồi tệ nhất là chi phí này tăng tương ứng với nhu cầu cập nhật nhanh hơn và số lượng client sử dụng dịch vụ tại một thời điểm nhất định. Trong ứng dụng cung cấp cập nhật thời gian thực, bạn dễ dàng tưởng tượng được tải mà server phải chịu khi có hàng nghìn người dùng kết nối, yêu cầu nhiều bản cập nhật mỗi giây.

Có những kỹ thuật để giảm thiểu các vấn đề này càng nhiều càng tốt. Một trong số đó là sử dụng khoảng thời gian thích nghi để khoảng thời gian giữa các truy vấn thường xuyên thích nghi với tải hệ thống hiện tại hoặc để xác suất cập nhật mới. Giải pháp này khá dễ thực hiện và có thể cải thiện đáng kể tiêu dùng tài nguyên trong một số tính huống.

Có một biến thể bảo thủ hơn về polling, nhưng nó làm giảm trải nghiệm người dùng. Đó là kỹ thuật được gọi là “piggy backing”, bao gồm không phải là đưa ra các câu hỏi cố ý từ client và thay vào đó, lợi dụng bất kỳ sự tương tác giữa người sử dụng và hệ thống để cập nhật bất kỳ thông tin cần thiết. Để minh hoạ điều này, hãy xem xét một dịch vụ web mail: thay vì tạo các truy vấn định kỳ để kiểm tra sự xuất hiện của tin nhắn mới, những kiểm tra sẽ được thực hiện mỗi khi người dùng truy cập vào một trang, Email, hoặc bất kỳ tính năng nào khác. Điều này có thể hữu ích trong các kịch bản không đòi hỏi sự “immediacy” và trong đó các tính năng của hệ thống chính nó có nghĩa là chúng ta có thể chắc chắn rằng người dùng sẽ tương tác với ứng dụng thường xuyên.

Tất nhiên, các biến thể này có thể được kết hợp với nhau để đạt được hiệu quả sử dụng hơn nguồn lực, cung cấp đồng thời một kinh nghiệm người dùng sử dụng hợp lý. Ví dụ, để có được cập nhật, nó sẽ có thể cập nhật tình trạng của một client thông qua “piggy backing”, khi client tương tác với server, sử dụng Polling có hoặc không có sự tuần hoàn thích ứng khi không có sự tương tác.

Kết luận, Polling là một lựa chọn hợp lý mặc dù bất lợi của nó khi chúng ta muốn có một giải pháp đó là dễ thực hiện và có thể được sử dụng phổ quát và trong các tình huống trong đó một tần số rất cao cập nhật là không cần thiết. Trên thực tế, nó được sử dụng rất nhiều trong các hệ thống hiện tại. Một ví dụ thực tế của nó ứng dụng được tìm thấy trong phiên bản web của Twitter, nơi polling được sử dụng để cập nhật thời gian biểu mỗi 30 giây.

## **3.3 Push: Server chủ động**

Có những ứng dụng mà việc sử dụng pull không phải là rất hiệu quả. Trong số đó, chúng ta có thể đặt tên hệ thống nhắn tin tức thì, bộ công cụ cộng tác thời gian thực, trò chơi trực tuyến người dùng đa người, dịch vụ phát thanh thông tin và bất kỳ loại hệ thống nào cần gửi thông tin tới client đúng khi nó được tạo ra.

Đối với các ứng dụng như vậy, chúng ta cần server chủ động và có khả năng gửi thông tin cho client một cách chính xác khi một sự kiện có liên quan xảy ra, thay vì chờ đợi client yêu cầu.

Đây chính là ý tưởng đằng sau khái niệm Push, **Server chủ động** . Tên này không tham chiếu đến một thành phần, một công nghệ hoặc một giao thức: nó là một khái niệm, một mô hình truyền thông giữa client và server, nơi mà nó là một trong những chủ động liên lạc.

Khái niệm này không phải là mới. Có những giao thức được đẩy theo khái niệm, chẳng hạn như IRC, giao thức điều khiển hoạt động của các dịch vụ phòng chat cổ điển, hoặc SMTP, giao thức phụ trách điều phối việc gửi email. Những điều này đã được tạo ra trước khi thuật ngữ xác định kiểu giao tiếp này đã được đặt ra.

Để server có thể thông báo sự kiện theo thời gian thực cho một nhóm client quan tâm đến việc tiếp nhận chúng, tình huống lý tưởng sẽ có khả năng bắt đầu kết nối điểm-điểm trực tiếp với họ. Ví dụ, một server phòng chat sẽ giữ một danh sách với các địa chỉ IP của các client kết nối và mở một kết nối kiểu socket cho mỗi client để thông báo về sự xuất hiện của một tin nhắn mới.

Tuy nhiên, đó là kỹ thuật không thể. Vì lý do bảo mật, thường không thể thực hiện kết nối trực tiếp tới máy tính client vì có nhiều cấp độ trung gian sẽ từ chối nó, chẳng hạn như tường lửa, đường đi hoặc proxy. Vì lý do này, thực tiễn tập quán là cho client là bắt đầu kết nối chứ không phải ngược lại.

Để phá vỡ vấn đề này và quản lý để có được một hiệu quả tương tự, một số kỹ thuật xuất hiện được dựa trên các yếu tố tích cực được nhúng trong các trang web (Java applet, Flash, ứng dụng Silverlight, v.v.). Các thành phần này thường được sử dụng ổ cắm để mở một kết nối liên tục đến server- tức là một kết nối sẽ vẫn mở cho đến khi client kết nối với dịch vụ, lắng nghe bất cứ thứ gì mà máy chủ phải thông báo. Khi các sự kiện xảy ra liên quan đến client được kết nối, server sẽ sử dụng kênh mở này để gửi cập nhật theo thời gian thực.

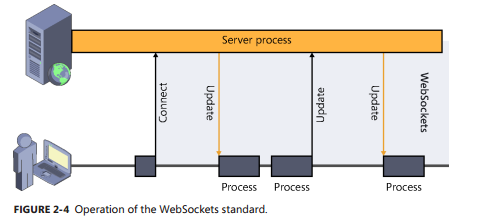
Mặc dù cách tiếp cận này đã được sử dụng trong nhiều giải pháp Push, nó có xu hướng biến mất. Các thành phần hoạt động được nhúng trong các trang đang được loại bỏ khỏi web với tốc độ nhanh chóng và đang được thay thế cho các giải pháp thay thế hiện đại, đáng tin cậy và phổ quát hơn như HTML5. Hơn nữa, các kết nối liên tục lâu dài dựa trên các ổ cắm tinh khiết là vấn đề khi có người trung gian.

Các yếu tố (tường lửa, proxy, v.v ...) có thể chặn các thông tin liên lạc này hoặc đóng các kết nối sau một khoảng thời gian không hoạt động. Chúng cũng có thể gây nguy hiểm cho server. Do nhu cầu về các giải pháp đáng tin cậy để đề cập đến các loại kịch bản này, cả W3C và IETF - các tổ chức chính thúc đẩy và xác định các giao thức, ngôn ngữ và tiêu chuẩn cho Internet - bắt đầu làm việc trên hai tiêu chuẩn cho phép truyền thông trực tiếp và thông thạo từ server cho client. Chúng được biết đến như là WebSockets và Server-Sent Events, và cả hai đều nằm dưới ô dù của tên thương mại "HTML5".

## **3.4 WebSockets**

Chuẩn WebSockets bao gồm một API phát triển, được xác định bởi W3C (World Wide Web Consortium, http://www.w3.org), và một giao thức truyền thông, trong đó IETF (Internet Engineering Task Force, http: //www.ietf.org) đang làm việc.

Về cơ bản, nó cho phép thiết lập một kết nối liên tục mà client sẽ bắt đầu bất cứ khi nào cần thiết và sẽ vẫn mở. Một kênh hai chiều giữa client và server được tạo ra, nơi mà có thể gửi thông tin đến đầu kia vào bất kỳ lúc nào, như thể hiện trong hình 2-4.



Mặc dù vào thời điểm đó, các yêu cầu kỹ thuật của cả API và giao thức đều khá cao, chúng ta vẫn chưa thể công nhận công nghệ này được áp dụng rộng rãi.

Chúng ta có thể tìm thấy các hiện thực của WebSockets trong nhiều trình duyệt hiện tại, chẳng hạn như Internet Explorer 10, Internet Explorer 11, Chrome và Firefox. Một số tính năng chỉ triển khai một phần (Opera mini, trình duyệt Android) và ở những trình duyệt khác, WebSockets đơn giản không có sẵn.

Bên cạnh vấn đề các cấp thực hiện khác nhau ở phía client, thực tế là tiêu chuẩn bao gồm một giao thức độc lập cho truyền thông (mặc dù ban đầu được thương lượng trên HTTP) có nghĩa là thay đổi cũng phải được thực hiện trên một số yếu tố cơ sở hạ tầng, và thậm chí trên các server, Để kết nối bằng WebSockets được chấp nhận.

Ví dụ, không thể sử dụng WebSockets một cách dễ dàng trên các công nghệ của Microsoft cho đến khi làn sóng phát triển mới nhất (Internet Explorer 10, ASP.NET 4.5, WCF, IIS 8, v.v ...), trong đó nó đã bắt đầu Được hỗ trợ natively.

Từ quan điểm của một nhà phát triển, WebSockets cung cấp một API JavaScript thực sự đơn giản và trực quan để bắt đầu các kết nối, gửi tin nhắn, và đóng các kết nối khi không cần thiết nữa, cũng như các sự kiện để nắm bắt các tin nhắn nhận được:



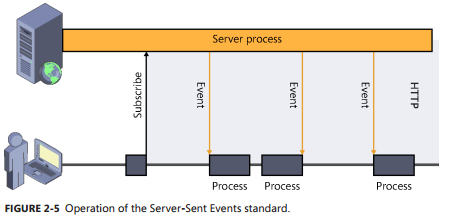
Như bạn thấy, kết nối được mở một cách đơn giản bằng cách khởi tạo một đối tượng WebSockets trỏ đến URL của điểm cuối dịch vụ. URL sử dụng giao thức ws:// để chỉ ra rằng nó là một kết nối WebSockets.

Bạn cũng có thể thấy chúng ta dễ dàng nắm bắt các sự kiện được tạo ra khi chúng ta thành công trong việc mở kết nối, dữ liệu được nhận, hoặc kết nối bị đóng.

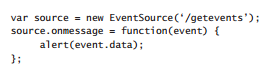
Không nghi ngờ gì, WebSockets là công nghệ của tương lai để thực hiện dịch vụ push thời gian thực.

## **3.5 Server-Sent Event (API Event Source)**

Các Sự kiện gửi bởi server, còn được gọi là Nguồn sự kiện API, là tiêu chuẩn thứ hai mà W3C đã làm việc. Hiện tại, tiêu chuẩn này đang trong trạng thái đề xuất của nhà nước. Nhưng lần này, bởi vì đó là một API JavaScript tương đối đơn giản và không cần thay đổi đối với các giao thức cơ bản, việc triển khai và áp dụng nó đơn giản hơn trong trường hợp của chuẩn WebSockets.

Ngược với sau này, Server-Sent Events đề xuất việc tạo ra một kênh một chiều từ server đến client, nhưng được mở bởi client. Nghĩa là client "đăng ký" vào một nguồn sự kiện có sẵn tại server và nhận các thông báo khi dữ liệu được gửi qua kênh, như được minh họa trong Hình 2-5.

Tất cả các giao tiếp được thực hiện trên HTTP. Sự khác biệt duy nhất liên quan đến kết nối truyền thống là việc sử dụng dòng văn bản/sự kiện-dòng-nội dung trong phản hồi, cho biết rằng kết nối sẽ được giữ nguyên bởi vì nó sẽ được sử dụng để gửi một chuỗi các sự kiện liên tục - Hoặc tin nhắn-từ server.

Thực hiện tại client thậm chí còn đơn giản hơn so với ứng dụng mà chúng ta đã thấy ở WebSockets:

Như bạn có thể đoán, instantiating các đối tượng EventSource bắt đầu đăng ký của client cho dịch vụ mà URL được cung cấp trong contructor, và các tin nhắn sẽ được xử lý trong chức năng gọi lại được chỉ định cho rằng có hiệu lực.

Hiện nay, hầu hết các trình duyệt hỗ trợ tiêu chuẩn này ngoại trừ Internet Explorer và một số trình duyệt di động, và điều này hạn chế việc sử dụng nó trong các ứng dụng thực tế. Ngoài ra, nếu chúng ta nhìn vào nó từ quan điểm cơ sở hạ tầng, chúng ta thấy rằng mặc dù dựa trên HTTP rất đơn giản hóa việc tổng quát của nó, nó đòi hỏi sự trợ giúp của các proxy hoặc các loại trung gian khác, nó phải có khả năng giải thích các loại nội dung được sử dụng và không xử lý các kết nối giống như cách truyền thống - ví dụ như tránh phản ứng đệm hoặc ngắt kết nối do thời gian chờ.

Cũng cần nhấn mạnh những hạn chế do thực tế là kênh được thiết lập cho giao thức này là một chiều từ server đến client: nếu client cần gửi dữ liệu đến server, nó phải thực hiện điều đó thông qua một kết nối khác, thường là một yêu cầu HTTP khác, ví dụ như có mức tiêu thụ tài nguyên lớn hơn nếu WebSockets được sử dụng trong cùng một kịch bản này.

## **3.6 Pus today**

Như chúng ta đã thấy, tiêu chuẩn và trình duyệt đều đang chuẩn bị để giải quyết các kịch bản push cổ điển, mặc dù hiện tại chúng ta không có đủ an ninh để sử dụng chúng một cách phổ quát.

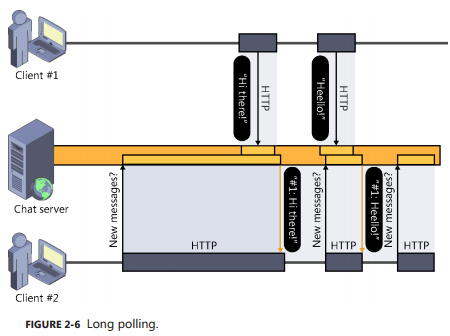
Tuy nhiên, đẩy là cái gì đó mà chúng ta cần ngay bây giờ. Người dùng yêu cầu các ứng dụng tương tác, nhanh nhẹn hơn và hợp tác hơn. Để phát triển chúng, chúng ta phải sử dụng các kỹ thuật cho phép chúng ta đạt được sự thúc đẩy ngay tức khắc nhưng phải tính đến những hạn chế hiện tại trong các trình duyệt và cơ sở hạ tầng. Tại thời điểm này, chúng ta có thể có được điều đó chỉ bằng cách tận dụng lợi thế của HTTP và sự phổ biến của nó.

Với các cơ sở này, rất dễ dàng tìm thấy nhiều đề xuất khái niệm trên Internet, chẳng hạn như Comet, push HTTP, reverse AJAX, AJAX push, và như vậy, mỗi mô tả các giải pháp (đôi khi coinciding) để đạt được các mục tiêu mong muốn. Trong cùng một cách, chúng ta có thể tìm thấy các kỹ thuật cụ thể khác nhau mô tả làm thế nào để thực hiện đẩy trên HTTP ít nhiều hiệu quả, chẳng hạn như “long polling”, XHR streaming, hoặc oforever frame.

Bây giờ chúng ta sẽ nghiên cứu hai trong số đó, cuộc thăm dò lâu dài và khung vĩnh cửu, vì hai lý do chính. Thứ nhất, bởi vì chúng là những hệ thống phổ quát nhất (chúng hoạt động trong tất cả các loại client và hệ thống server), và thứ hai, bởi vì chúng được sử dụng bởi SignalR, như chúng ta sẽ thấy sau này. Vì vậy, sẽ hướng tới mục tiêu của cuốn sách này.

## **3.7 Long Polling**

Kỹ thuật Push này khá giống với polling đã mô tả nhưng nó giới thiệu những sửa đổi nhất định để nâng cao hiệu quả truyền thông và tính trực tiếp.

Trong trường hợp này, client cũng polling cho các bản cập nhật, nhưng, không giống như trong polling, nếu không có dữ liệu nào đang chờ nhận, kết nối sẽ không tự động đóng lại và được khởi tạo lại sau. Trong Long Polling, kết nối vẫn mở cho đến khi server có thông báo gì đó, như trong hình 2-6.

Kết nối, luôn được khởi xướng bởi client, có thể được đóng lại vì chỉ có hai điều:

* Máy chủ gửi dữ liệu tới client thông qua kết nối.
* Lỗi thời gian chờ xảy ra do thiếu hoạt động về kết nối.

Trong cả hai trường hợp, một kết nối mới sẽ được thiết lập ngay lập tức, mà sẽ vẫn chờ đợi bản cập nhật.

Kết nối này được sử dụng độc quyền để nhận dữ liệu từ server, vì vậy nếu client cần gửi thông tin trở lên, nó sẽ mở một kết nối HTTP song song được sử dụng riêng cho mục đích đó.

Ưu điểm chính của việc “long polling” là độ trễ thấp trong việc cập nhật client, bởi vì ngay khi server có dữ liệu để cập nhật trạng thái của client, nó sẽ được gửi qua kênh đã mở, do đó đầu kia sẽ nhận được trong thời gian thực.

Ngoài ra, vì số lượng các lỗ nối và đóng cửa đã được giảm, tối ưu hóa tài nguyên ở cả hai đầu là cao hơn nhiều so với polling.

Hiện nay, đây là một giải pháp được sử dụng rộng rãi do thực hiện tương đối đơn giản của nó và thực tế là nó là hoàn toàn phổ quát. Không có tính năng trình duyệt cụ thể nào được sử dụng-chỉ khả năng được cung cấp bởi HTTP.

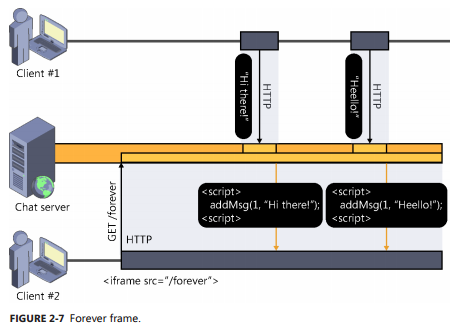
Tiêu thụ tài nguyên với thời gian “long polling” là hơi cao hơn so với các kỹ thuật khác, nơi kết nối được giữ mở. Lý do là vẫn có nhiều kết nối mở và đóng nếu tỷ lệ cập nhật cao, không quên các kết nối bổ sung đã được sử dụng khi client muốn gửi dữ liệu đến server. Đồng thời, thời gian cần thiết để thiết lập kết nối có nghĩa là có thể có sự chậm trễ giữa các thông báo. Những sự chậm trễ này có thể trở nên rõ ràng hơn nếu các server đã gửi một loạt các thông báo kế tiếp cho client. Trừ khi chúng ta triển khai một số loại tối ưu hóa, chẳng hạn như đóng gói nhiều tin nhắn vào cùng một phản hồi HTTP, mỗi tin nhắn sẽ phải chờ để được gửi trong khi client nhận được thông báo trước đó trong chuỗi, xử lý và mở lại kênh để yêu cầu cập nhật.

## **3.8 Forever frame**

Kỹ thuật khác mà chúng ta sẽ nhìn vào được gọi là **Forever frame** và sử dụng thẻ HTML <IFRAME> một cách khéo léo để có được một kết nối mở vĩnh viễn. Theo một cách nào đó, điều này rất giống với các sự kiện đã gửi của server.

Nói chung, nó bao gồm nhập một thẻ <IFRAME> trong trang đánh dấu của client. Trong nguồn của <IFRAME>, URL mà máy chủ đang nghe được xác định. Server sẽ duy trì kết nối này vĩnh viễn mở và sẽ sử dụng nó để gửi các bản cập nhật dưới dạng các cuộc gọi đến các chức năng kịch bản được xác định tại client. Theo cách nào đó, chúng ta có thể nói rằng kỹ thuật này bao gồm các kịch bản lệnh trực tuyến được thực hiện tại client khi chúng nhận được.

Bởi vì kết nối được giữ mở vĩnh viễn, các nguồn lực được sử dụng hiệu quả hơn bởi vì chúng không bị lãng phí trong quá trình kết nối và ngắt kết nối. Vì vậy, thực tế có thể đạt được thời gian thực của coveted trong hướng server-client.

Giống như trong kỹ thuật trước đó, việc sử dụng HTML, JavaScript và HTTP làm cho phạm vi ứng dụng của nó hầu như phổ biến, mặc dù rõ ràng là rất hướng về phía client hỗ trợ những công nghệ đó, chẳng hạn như các trình duyệt web. Nghĩa là, việc thực hiện các loại client khác, chẳng hạn như ứng dụng máy tính để bàn, hoặc các quá trình khác hoạt động như người tiêu dùng của các dịch vụ đó sẽ khá phức tạp, như thể hiện trong hình 2-7.

Kỹ thuật này cũng không được miễn trừ những bất lợi. Trong quá trình thực hiện, cần phải lưu ý rằng có thể xảy ra lỗi thời gian do client, server hoặc yếu tố trung gian (như proxy và tường lửa) gây ra. Ngoài ra, để có được trải nghiệm thời gian thực tốt nhất, phản hồi phải được gửi đến client ngay lập tức và không được giữ lại trong bộ đệm hoặc bộ nhớ cache. Và, bởi vì các phản hồi sẽ tích lũy bên trong khung nội tuyến, trong bộ nhớ của client, chúng ta có thể sẽ chiếm quá nhiều RAM, vì vậy chúng ta phải "tái chế" hoặc loại bỏ nội dung định kỳ.

Cuối cùng, thực tế là kết nối được sử dụng chỉ để gửi dữ liệu từ server tới client làm cho nó cần thiết phải sử dụng một kết nối bổ sung khi chúng ta muốn gửi nó theo hướng ngược lại - nghĩa là từ client đến server.

**Thế giới cần nhiều hơn!!!**

Cho đến nay, chúng ta đã thấy những kỹ thuật cho phép chúng ta đạt được sự thúc đẩy, nghĩa là chúng cho phép server có thể gửi thông tin đến client không đồng bộ khi nó được tạo ra. Đã có sáng kiến ​​cho một yếu tố thông thường sẽ đảm nhận vai trò thụ động trong giao tiếp với client.

Tuy nhiên, trong bối cảnh các ứng dụng không đồng bộ, đa người dùng và thời gian thực, đẩy là một trong những khía cạnh không thể thiếu. Để tạo ra những hệ thống này luôn luôn đáng ngạc nhiên, chúng ta cần nhiều khả năng hơn. Ở đây chúng tôi liệt kê một vài trong điểm:

* Quản lý người dùng kết nối server phải luôn biết người dùng nào được kết nối với các dịch vụ, những người đó ngắt kết nối và về cơ bản phải kiểm soát tất cả các khía cạnh liên quan đến giám sát một số lượng client không xác định.
* Quản lý đăng ký server phải có khả năng quản lý "đăng ký" hoặc nhóm các client muốn nhận các loại tin nhắn cụ thể. Ví dụ: trong dịch vụ phòng trò chuyện, chỉ những người dùng kết nối với một phòng cụ thể mới nhận được thông báo được gửi đến phòng đó. Bằng cách này, việc phân phối thông tin được tối ưu hóa và client không nhận được thông tin không liên quan đến chúng, giảm thiểu lãng phí nguồn lực.
* Nhận và xử lý các hành động các server có khả năng không chỉ gửi thông tin cho client trong thời gian thực mà còn nhận nó và xử lý nó trên bay.
* Giám sát các bài nộp Bởi vì không thể bảo đảm rằng tất cả các client kết nối trong cùng một điều kiện, có thể có các kết nối ở các tốc độ khác nhau, sự không ổn định đường dây, hoặc sự cố đột xuất, và điều này có nghĩa là nó cần phải cung cấp cho các cơ chế có khả năng xếp hàng và quản lý thông tin Riêng biệt để đảm bảo rằng tất cả client đều được cập nhật.
* Cung cấp một API linh hoạt, có khả năng tiêu thụ dễ dàng bởi nhiều client, điều này thậm chí còn đúng hơn ngày nay, khi có rất nhiều thiết bị từ đó có thể truy cập các dịch vụ trực tuyến.

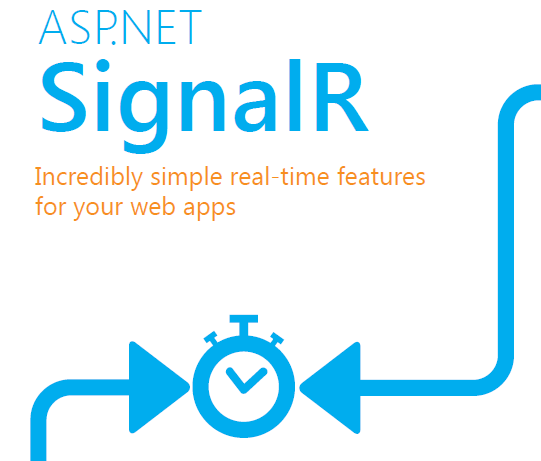
Chắc chắn có thể liệt kê nhiều hơn, nhưng những ví dụ này là quá đủ để cho bạn một ý tưởng về sự phức tạp vốn có trong việc phát triển các loại ứng dụng.

# Chương IV: Giới thiệu SignalR

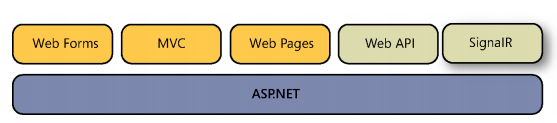
## **4.1 SignalR là gì?**

  SignalR là một thư viện lập trình do 2 thành viên của đội phát triển ASP.net Microsoft phát triển: David Fowler & Damien Edwards.

SignalR là 1 thư viện mã nguồn mở viết cho .NET giúp xây dựng các ứng dụng web sử dụng tương tác thời gian thực giữa người dùng với máy chủ. Ví dụ như: Mạng xã hội, games online, tin tức, ứng dụng thời tiết, bản tin tài chính, giá chứng khoán, ứng dụng chat,...

  
Mô hình bình thường của web là client gửi request đến server, nhưng giờ ta có 1 cách thức nữa để tương tác: server “push” trực tiếp dữ liệu về client, theo thời gian thực. Ngoài ra, mô hình này cũng giúp ta xây dựng ứng dụng bất đồng bộ lên đến hàng nghìn kết nối 1 lúc với máy chủ.

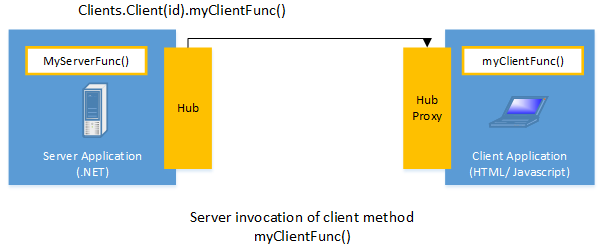
* SignalR cho phép lập trình viên tạo kênh liên lạc giữa server và client theo cơ chế notification. Trong dự án ASP.net chỉ cần bổ xung thư viện SignalR server lập trình bằng C#, còn tại browser dùng thư viện javascript SignalR client.
* Bây giờ nó là một sản phẩm tích hợp chính thức trong các stack of web technologies.
* Hình dưới đây đưa ra một ý tưởng đơn giản về vị trí của nó trong ngăn xếp ASP.NET, nơi chúng ta có thể xem Web Forms, MVC và Web Pages như các khuôn khổ để xây dựng các ứng dụng web và các trang, và Web API và SignalR để xây dựng các dịch vụ.

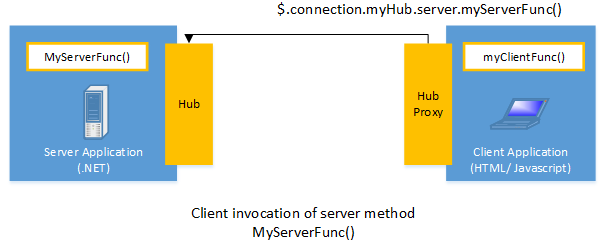


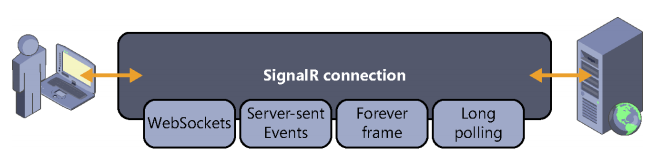
SignalR có thể được dùng thêm bất kỳ loại chức năng web “thời gian thực” cho ứng dụng ASP.NET của bạn. Trong khi chat thường được sử dụng làm ví dụ, thật ra bạn có thể làm được nhiều hơn thế. Bất kỳ lúc nào người dùng refreshes một trang web để xem dữ liệu mới, hoặc các trang thực hiện long polling (Client sẽ gửi request đến server, server tiến hành kiếm tra dữ liệu và đến khi nào có dữ liệu mới thì mới gửi response về cho client. Sau đó client lại tiếp tục tự động gửi 1 request mới và đợi dữ liệu mới trả về.) để nhận dữ liệu mới, nó là một đại diện dùng SignalR. Các ví dụ khác bao gồm các dashboard và các ứng dụng giám sát, các ứng dụng có sự hợp tác (chẳng hạn như chỉnh sửa đồng thời các tài liệu), cập nhật tiến độ công việc, và các form thời gian thực.

Signal cũng cho phép tạo ra các loại ứng dụng web hoàn toàn mới yêu cầu tần xuất cập nhật cao từ server, ví dụ, game thời gian thực. Ví dụ tuyệt với cho trường hợp này là ShootR game.

SignalR cung cấp một API đơn giản cho tạo thủ tục server-to-client từ xa gọi là RPC. Nó gọi các hàm JavaScript ở các browers client (và các nên tảng client khác) đến phía server .NET code. SignalR cũng thêm API cho quản lý kết nối (cho instance, sự kiện connect và disconnect) và nhóm các kết nối.





SignalR thực hiện quản lý kết nối một cách tự động, và cho phép bạn phát đi các tin nhắn cho tất cả các client kết nối cùng một lúc, giống một chat room. Bạn cũng có thể gởi các thông điệp đến các clients cụ thể. Kết nối giữa client và server là liên tục, không giống như một kết nối HTTP cơ bản, được thiết lập lại cho mỗi lần giao tiếp.

Theo một cách rõ ràng đối với nhà phát triển, SignalR có trách nhiệm xác định kỹ thuật nào tốt nhất có sẵn cả ở máy khách và máy chủ (long polling, forever frame, WebSockets,...) và sử dụng nó để tạo ra một cơ sở Kết nối và giữ nó liên tục mở, cũng tự động quản lý ngắt kết nối và nối lại khi cần thiết.

Như thể hiện trong hình, chúng ta sẽ thấy SignalR chỉ sử dụng một kết nối mở vĩnh viễn.

SignalR hỗ trợ chức năng “server push”, trong đó code server có thể gọi đến code của client trên browser dùng Remote Procedure Calls (RPC). Thay vì mô hình request-response phổ biến trên web hiện nay.

Các ứng dụng SignalR có thể mở rộng ra hàng ngàn client dùng Service Bus, SQL Server hoặc Redis.

SignalR là open-source, có thể truy cập thông qua GitHub.

## **4.2 SignalR và WebSocket**

SignalR dùng WebSocket (chỉ hỗ trợ trên HTLM5) để vận chuyển các thông tin. Khi bạn có thể viết ứng dụng của bạn dùng trực tiếp WebSocket, việc dùng Signal có ý nghĩa rằng nhiều tính năng mà bạn sẽ phải hiện thực SignalR đã làm hết cho bạn. Quan trọng nhất, điều này có nghĩa bạn có thể code chương trình của bạn tận dụng WebSocket mà không phải lo lắng về tạo các mã riêng cho các client sử dụng công nghệ cũ không phải HTML5. SignalR cũng giúp bạn khỏi phải lo lắng vì việc cập nhật của WebSocket. Vì SignalR sẽ tiếp tục được cập nhật để hỗ trợ sự thay đổi trong vận chuyển dữ liệu bên dưới. Cung cấp cho ứng dụng của bạn một giao diện nhất quán trên các phiên bản của WebSocket.

Mặc dù bạn chắc chắn có thể tạo ra một giải pháp sử dụng mỗi WebSocket, SignalR cung cấp tất cả các tính năng bạn sẽ cần để viết code, chẳng hạn như dự phòng cho các phương thức vận chuyển khác và sửa đổi ứng dụng của bạn khi triển khai các bản cập nhật WebSocket.

## **4.3 Phương thức vận chuyển**

SignalR là một sự trừu tượng hóa với một số các vận chuyển được yêu cầu làm việc thời gian thực giữa client và server. Một kết nối SignalR bắt đầu như HTTP, và sau đó được đẩy lên một kết nối WebSocket nếu có sẵn. WebSocket là phương tiện lý tượng để vận chuyển cho SignalR, vì nó làm tối ưu việc sử dụng bộ nhớ máy hiệu quả nhất, độ trễ thấp nhất, và có nhiều tính năng ngầm nhất (như truyền thông song song giữa client và server), nhưng nó cũng có những yêu cầu nghiêm ngặt nhất: WebSocket yêu cầu server dùng Windows Server 2012 hoặc Windows 8, và .NET Framework 4.5. Nếu những yêu không đó không được đáp ứng, SignalR sẽ cố gắng các phương thức vận chuyển khác để kết nối.

Trong phần này sẽ nói về các cách thức mà SignalR sử dụng để gửi dữ liệu về client. SignalR cung cấp một loại các cơ chế để gửi dữ liệu về client, tùy theo điều kiện đáp ứng của nền tảng máy chủ và client mà SignalR sẽ tự động lự chọn phương thức phù hợp nhất. Ở đây chia ra làm 2 loại như sau:

### **4.3.1 HTML 5 tranport**

* **WebSocket:**Là phương thức duy nhất hỗ trợ kết nối liên tục, hai chiều giữa client và server. Tuy nhiên nó đòi hỏi những điều kiện nghiêm ngặt về môi trường máy chủ cũng như trình duyệt như: **Windows Server 2012 or Windows 8, and .NET Framework 4.5,**trình duyệt tham khảo link sau [*http://caniuse.com/websockets*](http://caniuse.com/websockets)
* **Server Send Event:**hay còn được gọi là EventSource hỗ trợ tất cả các trình duyệt trừ IE

### **4.3.2 Comet tranport**

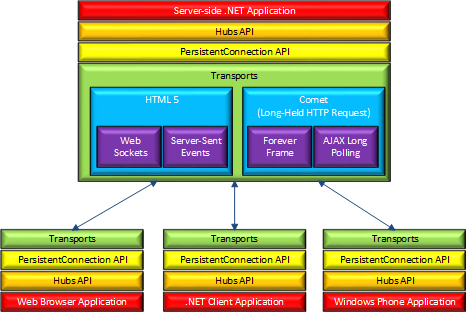
* **Forever Frame** (for Internet Explorer only) Kỹ thuật Forever Iframes sử dụng một thẻ Iframe ẩn đặt trong trang với thuộc tính src trỏ đến đường dẫn servlet nhằm trả về các sự kiện máy chủ. Mỗi khi nhận được một sự kiện, servlet sẽ viết và đổ vào một thẻ script với mã JavaScript bên trong. Nội dung của iframe sẽ được thêm vào thẻ script này và được thực thi.
* **Ajax long polling** Kỹ thuật này không tạo một kết nối liên tục, nhưng nó vẫn mở kết lối cho đến khi nhận được dữ liệu trả về từ server lúc này kết lỗi cũ sẽ được đóng lại và một kết lối mới sẽ được mở ra. Điều này có thể dẫn đến một độ trễ nhất định.

### [signalr_webvietgroup-2](http://webvietgroup.com/wp-content/uploads/2014/07/signalr_webvietgroup-2.jpg)**4.3.3 Quá trình lựa chọn các cách thức kết nối của SignalR**

1. IE 8 hoặc nhỏ hơn thì sử dụng Long Polling.
2. Nếu JSONP được cấu hình thì sử dụng Long Polling.
3. Web Sockets được sử dụng khi các yếu tố sau được đảm bảo.
   1. Client hỗ trợ CORS (Cross-Origin Resource Sharing)
   2. Client hỗ trợ Web Sockets
   3. Server hỗ trợ Web Sockets
   4. Nếu một trong ba yếu tố trên không được đáp ứng thì Long Polling sẽ được sử dụng.
4. Nếu JSONP không được cấu hình và kết nối không hỗ trợ cross-domain. Web Sockets sẽ được sử dụng nếu cả server và client đều hỗ trợ Web Sockets.
5. Nếu client hoặc server không hỗ trợ Web Sockets, Server Send Events sẽ được sử dụng nếu nó có sẵn.
6. Nếu Server Send Events không có thì Forever Frame sẽ được sử dụng.
7. Nếu Forever Frame không có thì Long Polling sẽ được sử dụng.

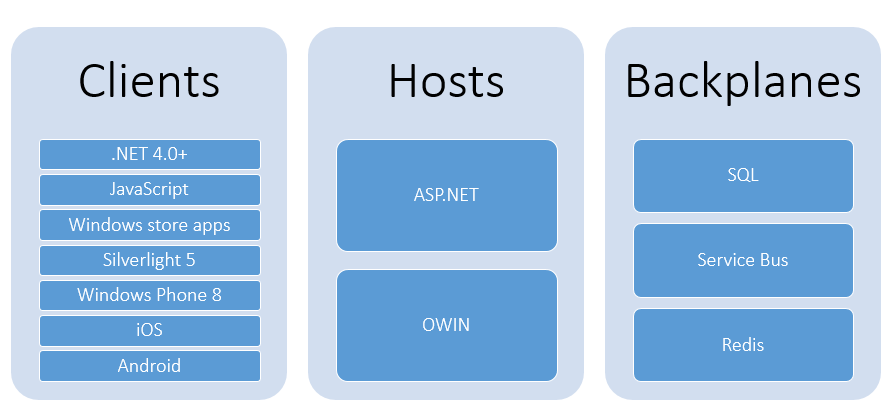
# Chương V: Kiến trúc & các thành phần của SignalR

## **5.1 Kiến trúc**



* SignalR cung một loại các cơ chế để gửi dữ liệu về client, tùy theo điều kiện đáp ứng của nền tảng máy chủ và client mà SignalR sẽ tự động lự chọn phương thức phù hợp nhất.
* Một kết nối SignalR được bắt đầu bằng giao thức HTTP, sau đó nó sẽ được chuyển sang kết nối kiểu WebSocket nếu nó được hỗ trợ.
* Ở đây được chia ra làm 2 loại như sau: **HTML 5 transport (WebSocket, Server Send Event) - Comet transport (Forever Frame, Ajax long polling)**.

## **5.2 Thành phần của SignalR**



* Server-side nó có thể được lưu trữ trong môi trường khác nhau (ASP.NET, console apps, windows services, ..).
* Client-side có hỗ trợ cho: Javascript clients, .NET clients, WP; Do cộng đồng cung cấp: iOS, Android.
* Ứng dụng sư dụng signalR có thể mở rộng cho hàng ngàn kết nối bằng cách sử dụng Service Bus, SQL server, hoặc [redis](http://redis.io/).
* **Redis là** 1 trong số các hệ quản trị cơ sở dữ liệu phát triển mang phong cách NoSQL. **Redis là** hệ thống lưu trữ key-value với rất nhiều tính năng và được sử dụng rộng rãi.

# Chương VI: Các nền tảng được hỗ trợ

## **6.1 Các yêu cầu hệ thống server**

Các thành phần Server SignalR có thể được tổ chức trên nhiều cấu hình máy chủ. Phần này mô tả các phiên bản được hỗ trợ của hệ điều hành, .NET framework, Internet Information Server, và các thành phần khác.

## **6.2 Các hệ thống server được hỗ trợ**

Các thành phần server SignalR có thể được lưu trữ trong các hệ thống server hoặc client. Chú ý rằng SignalR yêu cầu dùng WebSocket, Windows Server 2012 hoặc Windows 8

* Windows Server 2012
* Windows Server 2008 r2
* Windows 8
* Windows 7
* Windows Azure

## **6.3 Các phiên bản .NET Framework server được hỗ trợ**

SignalR 2 chỉ được hỗ trợ trên .NET Framework 4.5.

## **6.4 Các phiên bản IIS server được hỗ trợ**

Khi SignalR được lưu trữ trong IIS, các phiên bản sau được hỗ trợ. Lưu ý rằng nếu một hệ điều hành khách hàng được sử dụng, chẳng hạn như để phát triển (Windows 8 or Windows 7), không nên sử dụng phiên bản đầy đủ của IIS hoặc Cassini, vì sẽ có giới hạn 10 kết nối đồng thời được áp đặt, mà sẽ đặt được rất nhanh chóng kết từ khi kết nối được thông qua, thường xuyên thiết lập, và không được xử lý ngay khi không còn được sử dụng. IIS Express nên được sử dụng trên hệ điều hành client.

Cũng chú ý rằng SignalR dùng WebSocket, IIS 8 hoặc IIS Express phải được dùng, Server phải dùng Windows 8, Windows Server 2012, hoặc mới hơn, và WebSocket phải khích hoạt trên IIS.

* IIS 8 hoặc IIS 8 Express
* IIS 7 và 7.5. Hỗ trợ extensionless URLs được yêu cầu
* IIS phải chạy trong chế độ tích hợp, chế độ cổ điển không được hỗ trợ. Message trễ lên đến 30 giây có thể được trải nghiệm nếu IIS chạy trên chế độ cổ điển sử dụng phương thức vận chuyển Server-Sent Events. Ứng dụng lưu trữ phải chạy ở chế độ tin cậy đầy đủ.
* Ứng dụng lưu trữ phải chạy ở chế độ tin cậy đầy đủ.

## **6.5 Yêu cầu hệ thống client**

SignalR có thể được dùng trong một loạt các nền tảng client. Phần này mô tả các yêu cầu hệ thống sử dụng SignalR trong các trình duyệt web, chương trình Windows desktop, chương trình Silverlight, và các thiết bị di động.

## **6.6 Các trình duyệt web**

SignalR có thể được sử dụng trong nhiều trình duyệt web, nhưng thông thường, chỉ có 2 phiên bản mới nhất được hỗ trợ

Các chương trình dùng SignalR trong các trình duyệt phải dùng jQuery version 1.6.4 hoặc các phiên bản chính sau (như 1.7.2, 1.8.2 hoặc 1.9.1)

Signal có thể được dùng trong các trình duyệt sau:

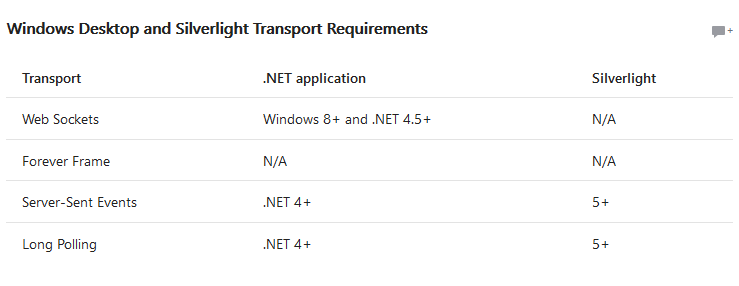
* IE phiên bản 8, 9, 10, và 11. Phiên bản hiện tại thì máy tính để bàn và di động đều được hỗ trợ.
* Mozilla Firefox: hiện tại phiên bản 1, cả phiên bản Windows và Mac
* Google Chrome: Hiện tại phiên bản 1, cả phiên bản Windows và Mac
* Safari: Phiên bản hiện tại 1, cả phiên bản MAC và iOS
* Opera: phiên bản hiện tại là 1, chỉ dùng cho winodws.

## **6.7 Các chương trình Window Desktop và Silverlight**

Ngoài việc chạy trên một trình duyệt web, SignalR có thể được lưu trữ trên máy client Windows độc lập hoặc các ứng dụng Silverlight. Các ứng dụng Windows Desktop và Silverlight thì SignalR có các yêu cầu sau:

* Các chương trình dùng .NET 4 được hỗ trợ trên Windows XP SP3 hoặc mới hơn
* Các chương trình dùng .NET Framework 4.5 được hỗ trợ trên Window Vista hoặc mới hơn.

Ngoài hệ điều hành và các yêu cầu .NET Framework. Các phương thức vận chuyển sẵn sàng cho SignalR có yêu cầu riêng. Các phương thức vận chuyển sau được hỗ trợ như hình sau:



# Chương VII: Triển khai & mở rộng SignalR

Nếu chúng ta phải tóm tắt quá trình triển khai bằng một câu, chúng ta có thể nói rằng đó là tập hợp các hành động cần được thực hiện để cài đặt phần mềm trong các môi trường mà nó có thể được sử dụng bởi người dùng. Trong bối cảnh các ứng dụng SignalR, các kịch bản triển khai có thể thay đổi rất nhiều do có nhiều nền tảng mà chúng ta có thể tìm thấy ở cả phía khách hàng và phía máy chủ và sự bùng nổ tổ hợp của các nền tảng này. Rõ ràng, việc triển khai các dịch vụ SignalR bên trong một ứng dụng ASP.NET thực hiện trên IIS không giống như việc thực hiện nó trong một dịch vụ hệ điều hành hoặc trong vai trò của một Windows Azure. Thủ tục triển khai ứng dụng Windows Phone 8 cũng không phải là cách sử dụng máy khách SignalR để truy cập vào các thông báo thời gian thực giống như quá trình triển khai ứng dụng cho Windows 8 hoặc một JavaScript client đơn giản.

Mỗi một trong những nền tảng này đều có các thủ tục triển khai riêng, sẽ vẫn còn nguyên vẹn với SignalR. Nghĩa là, bao gồm các thành phần SignalR trong một ứng dụng phía máy chủ không có ảnh hưởng gì đến cách mà nó phải được triển khai, cũng không ảnh hưởng đến phía client.

Ví dụ: trong các hệ thống web cổ điển, triển khai bao gồm cài đặt ứng dụng trên máy chủ, thường là theo các cách sau:

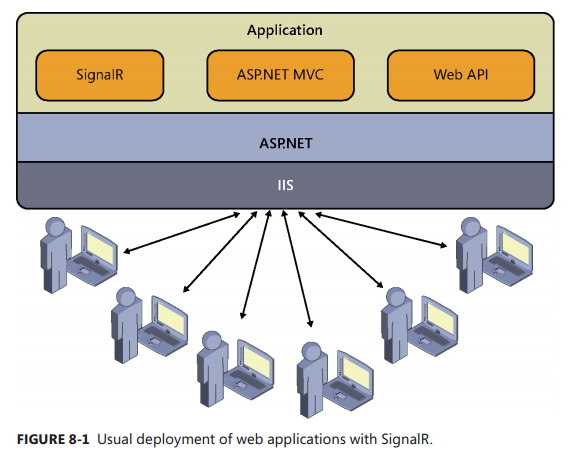
■ Sử dụng các công cụ xuất bản được tích hợp trong môi trường phát triển, chẳng hạn như tiện ích xuất bản Visual Studio.

■ Trong trường hợp các quá trình tự động, sử dụng Web Deploy 1 hoặc FTP, …

■ Bằng tay, sử dụng bất kỳ loại giao thức truyền tải nào cho phép chúng ta sao chép trực tiếp các tập tin vào máy chủ.

Tuy nhiên, thường không có hành động cụ thể để tạo thuận lợi cho việc triển khai các bộ phận đến phía khách hàng. Chúng sẽ được phân phối theo yêu cầu khi truy cập vào các trang hoặc các tính năng khác được cung cấp, và chúng sẽ được thực hiện trên trình duyệt của người dùng.

Trong các ứng dụng web với SignalR, nó là chính xác như nhau. Trên thực tế, vì tất cả các thành phần máy chủ của các ứng dụng SignalR được biên dịch (các hub, các kết nối liên tục, các lớp cấu hình ...), thì phải tuân theo bất kỳ quy trình xuất bản nào mà chúng ta thường sử dụng để nhận các dịch vụ thời gian thực của chúng tôi hoạt động trên các môi trường hoạt động . Các thành phần client - các tệp JavaScript cơ bản - sẽ được bao gồm trong gói cài đặt và sẽ được nhập vào ứng dụng dưới dạng tệp tĩnh.

Chúng ta thường gặp ứng dụng ASP.NET được triển khai theo cách này, với cấu trúc thể hiện trong hình 8-1

## **7.1 Triển khai SignalR**

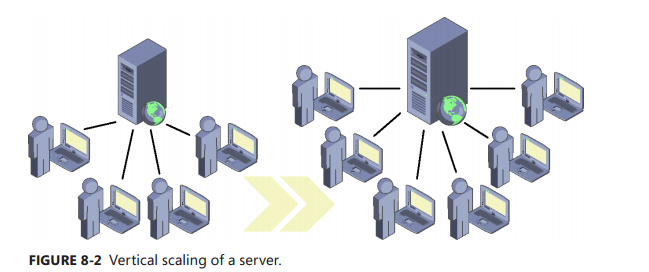
Vâng, sau nhiều giờ làm việc vô tận, nhiều tuần lễ với hầu như không có đêm nghỉ ngơi và không ngủ, cuối cùng chúng tôi đã tung ra hệ thống xã hội cách mạng của chúng tôi, trong đó, tất nhiên, chúng tôi đã sử dụng SignalR để thực hiện các tính năng ngoạn mục sử dụng truyền thông đẩy và thời gian thực như nhân loại chưa bao giờ được biết đến trước đây. Bây giờ chúng ta phải tiết lộ nó với thế giới và bắt đầu thu hút người dùng bởi hàng trăm. Có thể nói rằng các vấn đề của chúng ta đã kết thúc. (Hoặc bạn có thể nói rằng họ vừa mới bắt đầu.)

Từ điểm này, về cơ bản hai điều có thể xảy ra, trạng thái trung gian sang một bên. Trong trường hợp đầu tiên, ứng dụng siêu của chúng tôi sẽ trở nên không cần thiết như chúng tôi nghĩ, hoặc có lẽ người dùng không hiểu được giá trị đích thực của nó, và nó sẽ không bị ai sử dụng và cuối cùng biến mất trong sự quên lãng

Hoặc chúng ta có thể có trường hợp ngược lại: ứng dụng của chúng tôi là một hit thực sự. Thu hút bởi tính mới lạ và lời hứa cuối cùng là nhận được những trải nghiệm trực tuyến được chờ đợi suốt đời của họ, người dùng bắt đầu đăng ký một cách nhút nhát. Lời truyền miệng và truyền nhanh qua mạng xã hội khiến số lượng đăng ký tăng lên theo cấp số nhân sau vài tuần. Người dùng bắt đầu có số trong hàng trăm, sau đó hàng ngàn, và sau đó đến những khiếu nại đầu tiên về sự chậm trễ và các vấn đề với dịch vụ. Tác động này thường được gọi là "cái chết do sự thành công" và đó là lý do cho sự sụp đổ của nhiều dự án thú vị. Đây là một khoảnh khắc tinh tế, và sự sống sót của sinh vật của chúng ta sẽ phụ thuộc phần lớn vào khả năng của chúng ta để giải quyết những vấn đề này.

Đó là thời gian để quy mô - nghĩa là, để tìm cách để cung cấp chất lượng dịch vụ cho một số lượng ngày càng tăng của người sử dụng

Thật may mắn, luôn luôn có thể mở rộng máy chủ mà chúng tôi đã thuê để lưu trữ ứng dụng của chúng tôi, vì vậy cách tiếp cận đầu tiên của chúng tôi trong trường hợp này là đầu tư ở đó: tối đa hóa bộ nhớ RAM, cài đặt nhiều bộ vi xử lý như bảng hoặc máy ảo cho phép và cải tiến Lưu trữ liên tục về tốc độ và không gian. Cách nâng cao năng lực của cơ sở hạ tầng của chúng tôi để hỗ trợ nhiều người dùng được gọi là khả năng mở rộng theo chiều dọc. (Xem Hình 8-2.) Nó đòi hỏi hầu như không có thay đổi phần mềm hoặc các biện pháp phòng ngừa đặc biệt trong quá trình phát triển; Nó chỉ đơn giản liên quan đến việc mua hoặc thuê thêm kim loại hoặc thêm nhiều nguồn lực vào VM của chúng tôi.

Chú ý: Quy mô sử dụng phương pháp tiếp cận này thường được gọi là tăng quy mô.

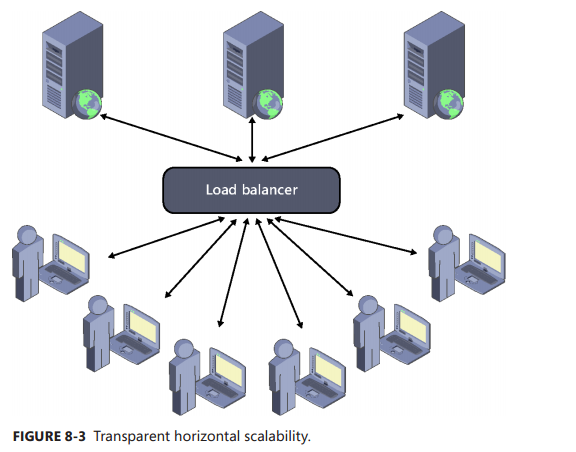
Các thành viên lạc quan hơn của nhóm có thể nghĩ rằng việc đầu tư lớn vào phần cứng này sẽ vượt qua được vấn đề. Và nó sẽ ... ít nhất trong một khoảng thời gian. Vấn đề với khả năng mở rộng theo chiều dọc là do hạn chế vốn có: tuy nhiên chúng ta muốn mở rộng một máy chủ, sẽ có một điểm vượt ra ngoài nó là thể chất không thể tăng khả năng của nó. Ở đây chúng tôi sẽ chuyển sang máy chủ có kiến trúc cho phép mở rộng nhiều hơn, nhưng với tốc độ tăng trưởng của ứng dụng siêu của chúng tôi.

Điều này sẽ chỉ phục vụ cho chúng ta một khoảng trống trong khi chúng ta tìm kiếm một giải pháp dứt khoát hơn. Và chúng ta không nên quên chi phí: thêm bộ nhớ hoặc CPU vào một hệ thống có xu hướng đắt tiền, và bình thường có một sự trở lại khá hạn chế về đầu tư. Nó cũng không phải là một lựa chọn rất linh hoạt. Nếu chúng ta biết rằng người dùng của chúng tôi kết nối chủ yếu vào giờ làm việc, trong suốt thời gian còn lại của ngày, tất cả sức mạnh và tài nguyên của máy chủ sẽ không được sử dụng, vì vậy chúng tôi sẽ không nhận được nhiều nhất từ đầu tư.

Tóm lại, khả năng mở rộng theo chiều dọc có giá trị đối với các môi trường khép kín hoặc kiểm soát chặt chẽ, chẳng hạn như các ứng dụng hoặc hệ thống của công ty có ít người dùng vì bản chất của dịch vụ và ngữ cảnh của nó. Nói chung, chúng tôi không cần nhiều hơn nữa nếu chúng tôi tạo một ứng dụng trò chuyện dành cho những người hâm mộ khó tính của nhóm trẻ sơ sinh bơi đồng bộ trong khu phố lân cận - chọn cho các giải pháp phức tạp hơn trong kịch bản này có thể được coi là quá mức, quá kỹ thuật, hoặc chỉ cần overkill.

Khi khả năng mở rộng theo chiều dọc là không đủ, giải pháp được đưa ra thông qua một cách tiếp cận mạnh mẽ hơn: khả năng mở rộng theo chiều ngang. Quy mô theo chiều ngang bao gồm việc tăng số lượng máy chủ cho đến khi họ có thể đáp ứng nhu cầu của người dùng.

Trong các môi trường web truyền thống, giải pháp lý tưởng là giới thiệu một cơ chế để tự động phân phối tải giữa các nút khác nhau theo cách minh bạch đối với người dùng, hành động vì mọi ý định và mục đích giống như ứng dụng đã làm việc trên một máy chủ duy nhất, Yêu cầu (thậm chí cả những người từ cùng một người dùng) có thể được xử lý bởi một nút khác trong mạng. Các yếu tố này được gọi là bộ cân bằng tải được đặt tại lối vào trang trại máy chủ và chuyển hướng yêu cầu đến máy chủ ít nhất bị bận để được xử lý ở đó. Xem Hình 8-3



Và nhờ các công nghệ hiện có, chúng ta thậm chí có thể sử dụng các dịch vụ đám mây như những gì được cung cấp bởi Windows Azure 2 để tự động điều chỉnh số lượng máy chủ theo nhu cầu thực tế tại mỗi thời điểm, do đó đảm bảo rằng ứng dụng của chúng tôi sẽ luôn sẵn sàng phục vụ Người sử dụng cũng như đạt được sự dễ dàng quản lý cơ sở hạ tầng và mức độ chịu lỗi tốt.

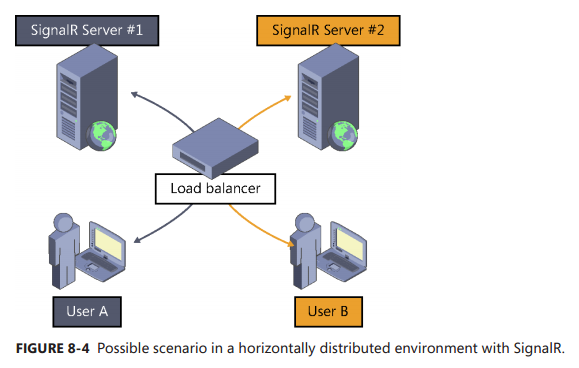
Ngoài ra, để giảm độ trễ và tăng hiệu suất, có thể đưa nội dung và dịch vụ gần hơn với người dùng tiêu thụ chúng, sử dụng CDN 3 (Mạng phân phối nội dung) hoặc các máy chủ có chiến lược. Lợi thế chính so với mở rộng quy mô là trong quy mô-out có một giới hạn về thể chất priorino: chúng ta luôn có thể thêm các máy chủ mới để chịu tải lớn hơn. Tuy nhiên, để làm việc trên kiến trúc này, các ứng dụng phải được thiết kế để dự đoán khả năng này, bởi vì mỗi yêu cầu có thể được xử lý bởi một máy chủ khác, bạn không thể sử dụng các tài nguyên cục bộ của bộ nhớ thứ cấp (như bộ nhớ hoặc hệ thống tập tin) để lưu trữ Dữ liệu liên tục hoặc để lưu trữ trạng thái của ứng dụng.

Là một lưu ý phụ, có một sự thay thế được gọi là sự liên quan, bao gồm việc cân bằng tải luôn gán các yêu cầu đến từ một người dùng nhất định vào cùng một máy chủ, theo đó thông tin trạng thái có thể được lưu trữ trong các máy chủ. Còn được gọi là phiên dính, kỹ thuật này tốt hơn so với quy mô theo chiều dọc nhưng tồi tệ hơn quy mô theo chiều ngang với cách tiếp cận không quốc tịch.

## **7.2 Mở rộng in SignalR**

Chúng tôi đã giải thích rằng trong các môi trường web được điều chỉnh theo chiều ngang, chúng ta không nên sử dụng các tài nguyên máy chủ cục bộ như bộ nhớ hoặc đĩa. Những điều cần được thậm chí nhiều hơn omplicated khi nói đến các ứng dụng SignalR.

Khi chúng ta xem xét các kỹ thuật bảo trì của nhà nước tại máy chủ trong Chương 5, "Hubs", chúng tôi mô tả các vấn đề về việc có nhiều máy chủ SignalR phục vụ song song và chúng tôi đưa ra một số hướng dẫn để giải quyết chúng. Tuy nhiên, như chúng tôi đã giải thích, SignalR đặt ra một thách thức thực sự trong các kịch bản phân tán theo chiều ngang, đặc biệt là bởi vì mỗi máy chủ chỉ nhận thức được các khách hàng trực tiếp kết nối với nó. Vấn đề trình bày này được minh họa trong ví dụ "tối giản" thể hiện trong hình 8-4: người dùng A được kết nối với dịch vụ SignalR, và bộ cân bằng tải phân bổ nó đến máy chủ số 1. Sau đó đến người dùng B và nó được cấp phát cho máy chủ #2.

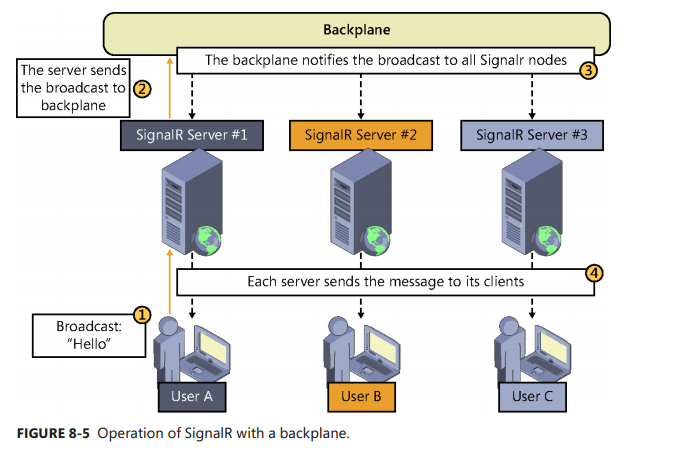


Mặc dù cả hai người dùng về mặt lý thuyết sẽ kết nối với cùng một kết nối hoặc trung tâm liên tục - sẽ chỉ có một URL truy cập-bộ cân bằng sẽ ủy quyền xử lý cho một máy chủ cụ thể, một máy chủ khác trong mỗi trường hợp. Và bởi vì mỗi một kết nối vật lý với một máy chủ khác, thông điệp phát sóng do máy chủ số 1 sẽ chỉ nhận được bởi người dùng A và cũng giống với máy chủ khác.

Ngoài ra, việc phân bổ khách hàng này cho các máy chủ có thể thay đổi mỗi lần kết nối vật lý mới mở ra, gây ra tình huống khó quản lý do chuyển vùng. Ví dụ: hãy tưởng tượng rằng người dùng A được kết nối với máy chủ số 1 và trải qua thời gian chờ do sự cố mạng, kết nối lại và được phân bổ cho máy chủ số 2. Từ quan điểm của máy chủ số 1, người dùng A đã rời ứng dụng, trong khi từ máy chủ # 2, người dùng A sẽ được coi là người dùng mới. Hơn nữa, trong thời gian thực hiện cho quá trình kết nối lại, các tin nhắn mới có giá trị được gửi đến người dùng có thể đã vào hệ thống.

Tác động tương tự này có thể xảy ra với các kết nối song song được thực hiện bởi khách hàng để gửi thông tin tới máy chủ trong các cuộc vận chuyển đẩy một chiều như các sự kiện do máy chủ gửi hoặc các cuộc thăm dò lâu dài. Mỗi một có thể được xử lý bởi một máy chủ khác nhau, nhưng kết nối push chính sẽ vẫn hoạt động và cố định từ đầu. Như bạn có thể đoán, những vấn đề này không phải là dễ dàng để giải quyết.

Để giải quyết các tình huống này, SignalR đi kèm với một cơ chế mạnh mẽ cho phép triển khai các thành phần máy chủ của nó trong các môi trường phân tán ngang thông qua các lớp nền. Như trong hình 8-5, bảng nối đa năng là một thành phần hoạt động như một hệ thống nhắn tin giữa các nút SignalR, tương tự như một bus để liên lạc nội bộ giữa các hệ thống. Khi được kích hoạt trên máy chủ, tất cả các thông điệp được tạo ra từ nó sẽ được gửi qua bảng nối tiếp, nơi mà các máy chủ còn lại sẽ lắng nghe để chuyển tiếp chúng tới các khách hàng của họ.



Có nhiều loại backplanes khác nhau, mỗi nhóm sử dụng một công nghệ khác nhau để quản lý các ấn phẩm và đăng ký sử dụng trong hệ thống nhắn tin. Hiện tại, SignalR chính thức cung cấp backplanes cho Windows Azure Service Bus, SQL Server và Redis, nhưng cộng đồng nhà phát triển đã cung cấp một số bổ sung. Chúng tôi sẽ mô tả từng chi tiết sau.

Mặc dù sử dụng backplane cho phép sử dụng SignalR trong một trang trại máy chủ và phân phối tải người dùng giữa các máy chủ của nó, nhưng làm như vậy cũng có những hạn chế của nó. Hạn chế chính là nó chắc chắn sẽ dẫn đến sự gia tăng thời gian cần thiết để gửi mỗi tin nhắn và do đó giảm đáng kể số lượng tin nhắn có thể di chuyển qua hệ thống - đó là thông lượng. Nghịch lý, giải pháp tương tự được sử dụng để cho phép khả năng mở rộng ngang của các dịch vụ SignalR có thể trở thành một nút cổ chai trong các kiến trúc đã được thu nhỏ. Trên thực tế, khả năng mở rộng không đồng nghĩa với việc tăng hiệu suất trong trường hợp này.

Do đó, chúng ta không nên nghĩ đến các backplanes như là giải pháp cuối cùng cho các vấn đề về khả năng mở rộng của tất cả các loại ứng dụng SignalR, nhưng chỉ như là cơ chế cho phép sử dụng khuôn khổ này với các trang trại máy chủ cân bằng, trong trường hợp đây sẽ là kiến trúc triển khai cuối cùng Chọn cho hệ thống của chúng tôi.

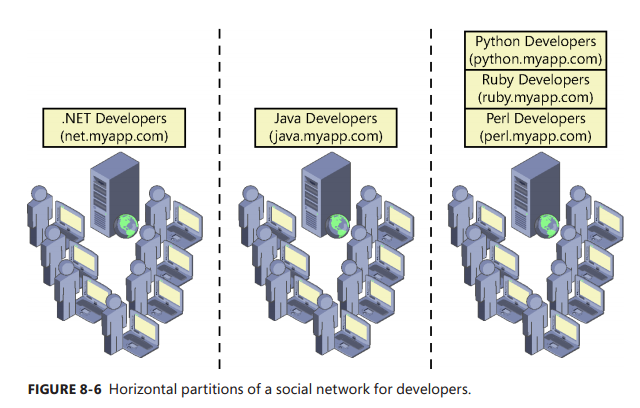
Việc sử dụng backplanes được khuyến cáo trong các ứng dụng có một máy chủ phát sóng cùng một thông điệp cho nhiều khách hàng hoặc nhóm khách hàng. Điều này là do, trong SignalR, các thư được xử lý sao cho gửi cùng một thông điệp tới nhiều khách hàng sẽ không làm say đắm cho chiếc xe buýt. Việc sử dụng của họ cũng thích hợp khi có tần suất gửi rất thấp hoặc nếu số tiền của họ không liên quan trực tiếp đến sự gia tăng số lượng khách hàng. Trong tất cả các trường hợp này, bảng nối đa năng có thể mang lại hiệu suất tốt và có khả năng đồng thời khả năng mở rộng gần như không giới hạn.

Tuy nhiên, ngay khi thông tin được cá nhân hóa cần được gửi đến từng khách hàng hoặc ngay lập tức là rất quan trọng, chúng ta phải dùng đến các giải pháp khác

Khi thiết kế chiến lược khả năng mở rộng cung cấp hiệu suất phù hợp cho loại ứng dụng được đề cập, có nhiều giải pháp như các loại hệ thống và các nhóm phát triển đang cố gắng thực hiện chúng. Điều gì sẽ là cách tốt nhất để thiết kế một ứng dụng SignalR mở rộng? Câu trả lời tốt nhất mà chúng tôi có thể cho là "nó phụ thuộc".

Nói chung, công thức để nhân rộng các hệ thống của chúng tôi luôn luôn bao gồm phân phối tải trên nhiều máy chủ, nhưng nếu chúng ta muốn duy trì hiệu năng tốt trong các dịch vụ thời gian thực dựa trên SignalR, chúng ta phải thiết kế và thực hiện các giải pháp của riêng mình. Và trong trường hợp này, chúng hầu như luôn luôn liên quan đến việc tìm kiếm các kỹ thuật phân vùng hoặc phân phối người dùng theo các tiêu chí logic cụ thể cho miền của ứng dụng mà chúng tôi đang phát triển để nhóm chúng vào cùng một máy chủ. Nếu tất cả những người dùng mà chúng tôi muốn truyền thông như một nhóm đều được tìm thấy trên cùng một máy chủ, các bản đệ trình sẽ được trực tiếp và chúng tôi sẽ tránh những người trung gian có thể giới thiệu độ trễ, chẳng hạn như khi sử dụng backplanes. Chúng tôi cũng sẽ đạt được hiệu quả và chúng tôi sẽ có thể mở rộng quy mô của mình theo chiến lược nhóm mà chúng tôi đã lựa chọn cho phép chúng tôi, mặc dù thường có chi phí tăng nỗ lực phát triển.

Cách tiếp cận đơn giản và cơ bản nhất có thể nảy sinh nếu chúng ta biết trước và chắc chắn về những gì mà người dùng chúng tôi đang xử lý và phân phối của họ, bởi vì chúng tôi có thể định nghĩa một kiến trúc tĩnh cho họ. Chúng ta sẽ phải chuẩn bị cơ sở hạ tầng của các máy chủ được cấu hình và thích nghi với các nhu cầu ban đầu của hệ thống, như thể hiện trong hình 8-6, và thích ứng nó theo sự tiến triển của nhu cầu.



Trong ví dụ này, chúng tôi có thể tạo các dịch vụ chat mới vì các công nghệ mới xuất hiện trong tương lai và lưu trữ các dịch vụ này trên các máy chủ có dung lượng dự phòng hoặc thêm các máy chủ mới. Tuy nhiên, để làm cho tùy chọn này có thể, do tính chất của dịch vụ, chúng ta nên đảm bảo rằng không có phân vùng sẽ phát triển vượt ra ngoài những gì đã được giả định khi quy hoạch kích thước của cơ sở hạ tầng.

Chúng ta cũng có thể có một cách tiếp cận năng động hơn và linh hoạt, điều này sẽ được khuyến khích trong hầu hết các kịch bản. Theo khái niệm, điều này có nghĩa là tạo ra một hệ thống cân bằng "thông minh", nhạy cảm với miền của ứng dụng, có thể bao gồm một dịch vụ phía trước có thể chuyển hướng khách hàng đến máy chủ phù hợp, dựa trên các tiêu chí nhất định.

Ví dụ: trong dịch vụ chỉnh sửa văn bản cộng tác, tiêu chí nhóm người dùng có thể là chính tài liệu. Nếu người dùng nhập vào ứng dụng và quyết định mở một tài liệu, một máy chủ SignalR có thể được gán cho chúng trên máy bay - máy chủ bận rộn nhất trong một tập hợp các máy chủ có sẵn. Lập bản đồ này sẽ được lưu trữ trong một hệ thống tồn tại tập trung, và người sử dụng sẽ được chuyển hướng đến máy chủ nói rằng cô ấy có thể bắt đầu làm việc trên tài liệu. Nếu một số người dùng khác truy cập vào cùng một tài liệu sau đó, ứng dụng sẽ biết rằng đã có một máy chủ thời gian thực được gán cho nó và sẽ chuyển hướng người dùng mới này tới nó. Trong trường hợp này, tiêu chí phân chia hoặc phân phối người dùng sẽ là tài liệu.

Chúng ta có thể tìm thấy một ứng dụng khác trong một hệ thống nhắn tin nội bộ trên một hệ thống ERP đa cấp được phân phối như là SaaS. Trong trường hợp này, tiêu chí phân bổ người dùng có thể là chính người thuê nhà - nghĩa là công ty sử dụng dịch vụ - bởi vì mỗi người trong số họ hoàn toàn độc lập với những người khác, và do đó các dịch vụ thời gian thực có thể được cung cấp trong các máy chủ riêng biệt.

Lưu ý rằng một cách tiếp cận tương tự hiện đang được sử dụng trong nhiều ứng dụng. Ví dụ: có nhiều trò chơi trực tuyến, nơi người dùng có khu vực có thể sử dụng để tìm đối thủ hoặc thực hiện các hoạt động khác, nhưng tại thời điểm bắt đầu trò chơi, một máy chủ hiện có được chỉ định và tất cả người dùng liên quan đến cùng một trò chơi được chuyển hướng tới nó. Trong trường hợp này, tiêu chí phân vùng sẽ là trò chơi.

Cũng không khả thi khi sử dụng máy tính xách tay trong các hệ thống khi cần truyền thông theo thời gian thực với tần suất cập nhật cố định mỗi giây, chẳng hạn như trong một game bắn súng nhiều người chơi. Kịch bản này, có lẽ là một trong những điểm cực đoan nhất mà chúng ta phải đối mặt khi phát triển ứng dụng thời gian thực, có thể được tiếp cận theo cách tương tự: làm cho tất cả người dùng tham gia vào hành động hoặc trò chơi kết nối với cùng một máy chủ để có thể gửi tin nhắn của họ trực tiếp và không có người trung gian.

Một khả năng chúng ta có thể xem xét khi sử dụng backplanes đang cố gắng tối ưu hóa lưu lượng truy cập được tạo ra trong bus nhắn tin dựa trên nhu cầu của ứng dụng của chúng tôi. Ví dụ, sẽ có thể tạo ra một hệ thống để giữ cho các thuê bao người dùng nhận được cập nhật trên mỗi nút, và thực hiện một hệ thống lọc ngăn chặn các thông điệp mà một nút không cần đến từ nút đó qua bảng nối đa năng.

Tất nhiên, chúng ta có thể kết hợp một số kỹ thuật này hoặc đưa ra các chiến lược mới phù hợp với nhu cầu chính xác của từng ứng dụng, nhưng khác với các backplanes, chúng ta sẽ phải thực hiện giải pháp của riêng mình.

# Chương VIII: Demo SignalR

## **8.1 Ứng dụng chat thời gian thực**

Hướng đẫn này cho bạn biết làm thế nào dung SignalR để tạo một ứng dụng chat thời gian thực. Bạn sẽ thêm SignalR vào một chương trình web ASP.NET và tạo một HTML page để gởi và hiển thị các message

Các phiên bản phần mềm được dùng trong hướng dẫn này:

* Visual Studio 2013
* .NET 4.5
* SignalR version 2

### **8.1.1 Giới thiệu**

Hướng dẫn này giới thiệu phát triển SignalR bằng cách xây dựng ứng dụng chat đơn giản trên trình duyệt. Bạn sẽ thêm thư viện SignalR vào một chương trình web APS.NET trống, tạo một hub class cho việc gởi các message đến clients, và tạo một HTML page để các người dùng gởi và nhận chat message.

SignalR là một opensource thư viện .NET dành cho việc xây dựng các chương trình web yêu cầu tương tác trực tiếp và cập nhật dữ liệu thời gian thực. Ví dụ bao gồm các ứng dụng xã hội, game nhiều người chơi, hợp tác kinh doanh, và tin tức, thời tiết, hoặc các ứng dụng cập nhật tài chính. Chúng thường được gọi là các ứng dụng thời gian thực.

SignalR đơn giản quá trình xây dựng một ứng dụng thời gian thực. Nó bao gồm một thư viện máy chủ ASP.NET và một thư viện client JavaScript để làm cho nó dễ dàng hơn việc quản lý các kết nối giữa client-server và đẩy các nội dung cập nhật cho các client. Bạn có thr thêm thư viện SignalR vào một ứng dụng ASP.NET hiện có để đặt được tính năng thời gian thực.

Hướng dẫn này trình bày các việc cần làm để phát triển chương trình dùng SignalR:

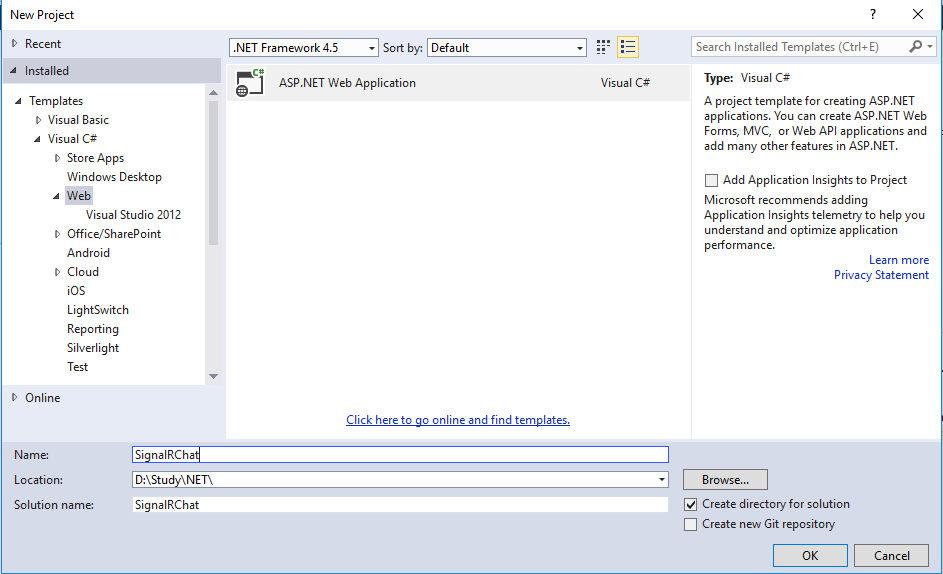
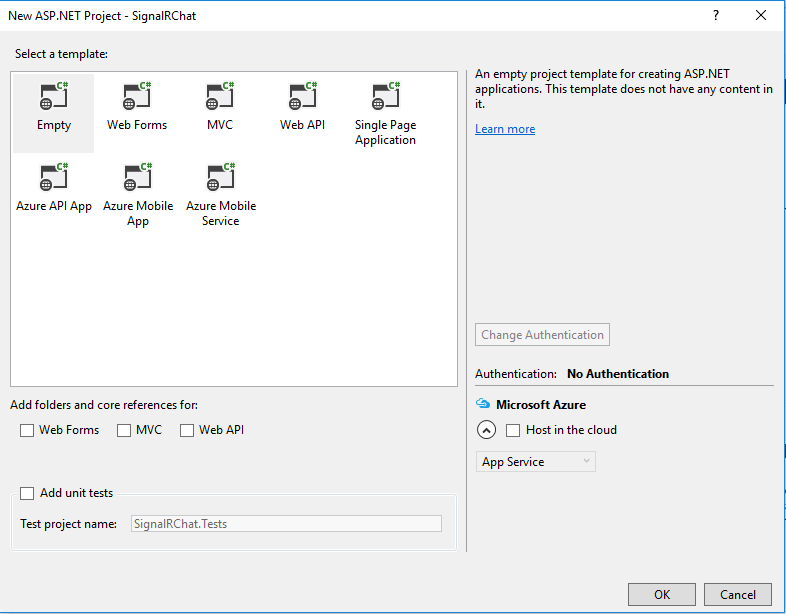
* Thêm thư viện SignalR vào chương trình web ASP.NET
* Tạo một class hub để đấy nội dung về các client
* Tạo mọt class startup OWIN để configure chương trình
* Dùng thư viện jQuery SignalR trong một trang web để gởi các message và hiển thị các cập nhật từ hub

Ảnh chụp màn hình sau đây hiển thị ứng dụng chat đang chạy trên trình duyệt. Mỗi người dùng mới có thể đăng nhận xét và xem nhận xét sau khi người dùng tham gia cuộc trò chuyện.

### **8.1.2 Cài đặt Project**

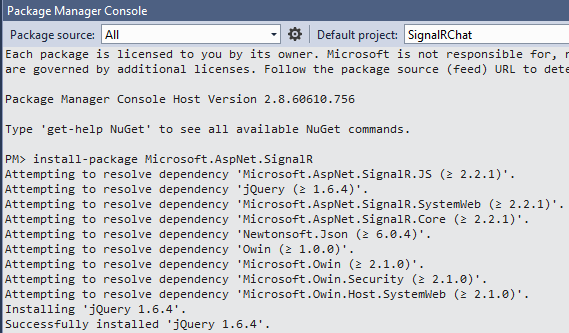
Phần này chỉ ra cách sử dụng Visual 2013 và SignalR 2 để tạo một ứng dụng web ASP.NET rỗng, thêm SignalR và tạo ứng dụng trò chuyện

Các bước sau đây sử dụng Visual Studio 2013 để tạo một ứng dụng web ASP.NET trống và thêm thư viện SignalR:

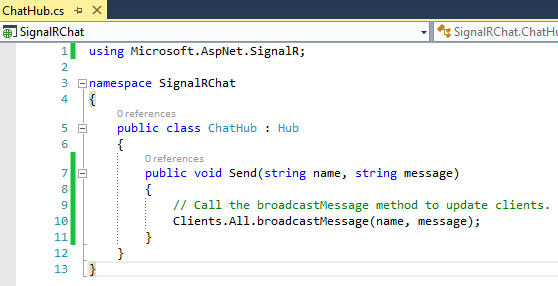
1. Trong Visual Studio, tạo một ASP.NET Web Application.
2. Trong cửa sổ New ASP.NET Project, chọn Empty và chọn Create Project
3. Trong Solution Explorer, chuột phải project, chọn Add | SignalR Hub Class (v2). Tên class là ChatHubs.cs và thêm nó vào project. Bước này tạo class ChatHub và thêm các project một tập các file script và các assembly references có hỗ trợ SignalR.

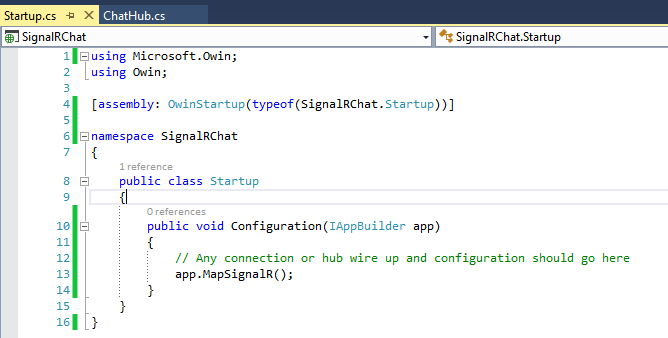
Chạy dòng lệnh sau trong Package Manager Console:

install-package Microsoft.AspNet.SignalR



Nếu bạn dùng console thêm SignalR, tạo SignalR hub class như một bước riêng biệt sau khi bạn thêm SignalR (chuột phải project và chọn Add | SignalR Hub Class (v2))

1. Trong Solution Explorer, mở thư mục Scripts, các thư viện Script cho jQuery và SignalR được nhìn thấy trong project.
2. Thay đoạn code trong class mới ChatHub với đoạn code bên dưới.
3. Trong Solution Explorer, chuột phải vào project, sau đó click Add | OWIN Startup Class. Đặt tên class là Startup và click OK
4. Thay đổi các nội dung của Startup class như bên dưới.



1. Trong Solution Explorer, chuột phải project, sau đó click Add | HTML Page. Đặt tên cho trang mới này là index.html
2. Trong Solution Explorer, chuột phải trang HTML bạn vừa tạo và click Set as Start Page
3. Thay code mặc định trong trang này bằng đoạn code sau:

<!DOCTYPE html>

<html>

<head>

<title>SignalR Simple Chat</title>

<style type="text/css">

.container {

background-color: #99CCFF;

border: thick solid #808080;

padding: 20px;

margin: 20px;

}

</style>

</head>

<body>

<div class="container">

<input type="text" id="message" />

<input type="button" id="sendmessage" value="Send" />

<input type="hidden" id="displayname" />

<ul id="discussion">

</ul>

</div>

<!--Script references. -->

<!--Reference the jQuery library. -->

<script src="Scripts/jquery-1.6.4.min.js" ></script>

<!--Reference the SignalR library. -->

<script src="Scripts/jquery.signalR-2.1.0.min.js"></script>

<!--Reference the autogenerated SignalR hub script. -->

<script src="signalr/hubs"></script>

<!--Add script to update the page and send messages.-->

<script type="text/javascript">

$(function () {

// Declare a proxy to reference the hub.

var chat = $.connection.chatHub;

// Create a function that the hub can call to broadcast messages.

chat.client.broadcastMessage = function (name, message) {

// Html encode display name and message.

var encodedName = $('<div />').text(name).html();

var encodedMsg = $('<div />').text(message).html();

// Add the message to the page.

$('#discussion').append('<li><strong>' + encodedName

+ '</strong>:&nbsp;&nbsp;' + encodedMsg + '</li>');

};

// Get the user name and store it to prepend to messages.

$('#displayname').val(prompt('Enter your name:', ''));

// Set initial focus to message input box.

$('#message').focus();

// Start the connection.

$.connection.hub.start().done(function () {

$('#sendmessage').click(function () {

// Call the Send method on the hub.

chat.server.send($('#displayname').val(), $('#message').val());

// Clear text box and reset focus for next comment.

$('#message').val('').focus();

});

});

});

</script>

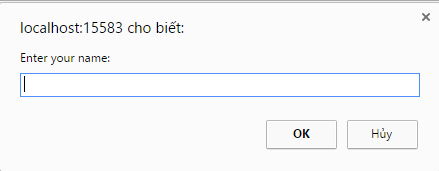
</body>

</html>

1. Save All project

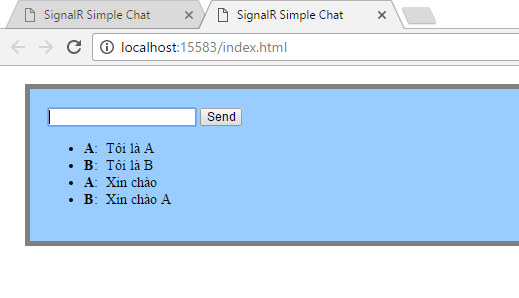
### **8.1.3 Chạy ví dụ**

1. Bấm F5 để chạy project ở chế độ debug. Trang HTML tải lên trình duyệt và hộp thoại nhập tên người dùng xuất hiện.



1. Nhập 1 tên.
2. Copy URL của trình duyệt và dùng nó để mở nhiều tab khác. Trong mỗi tab, nhập một tên duy nhất.
3. Trong mỗi trình duyệt, thêm một comment và click Send. Comment sẽ hiển thị trong tất cả các trình duyệt còn lại.

Ảnh chụp màn hình sau đây hiển thị ứng dụng chạy trên 3 tab trình duyệt khác nhau, tất cả đều được cập nhật khi một cá nhân gởi 1 thông báo.



**Giải thích Code**

Ứng dụng chat Signal thẻ hiện hai nhiệm vụ phát triển Signal cơ bản: Tạo ra một hub như là đối tượng điều phối chính trên máy chủ, và dùng thư viện jQuery SignalR để gởi và nhận các message

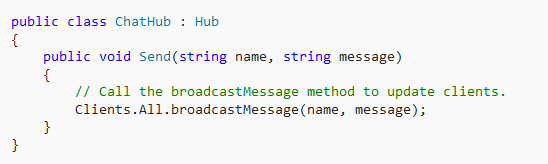
**Các SignalR Hubs**

Trong code ví dụ, ChatHub class kế thừa từ class Microsoft.AspNet.SignalR.Hub. Kế thừa từ class Hub là một cách hữu ích để xây dựng một ứng dụng SignalR. Bạn có thể tạo các phương thức public trên class hub và sau đó truy cập phương thức đó bằng cách gọi chúng từ script trong một trang web

Trong chat code, các client gọi phương thức ChatHub.Send để gởi một message mới. Hub lần lượt gởi message đến tất cả các client bằng cách gọi Client.All.broadcastMessage.

Phương thức Send phải có những phần sau:

* Dùng phương thức public trên một hub để các client có thể gọi chúng
* Dùng Microsoft.AspNet.SignalR.Hub.Clients để truy cập tất cả client được connect tới hub
* Gọi một hàm trên client (như broadcastMessage) để cập nhật cho các client



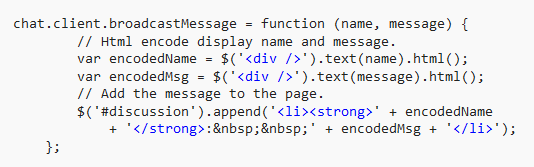
**SignalR và jQuery**

Trang HTML trong code ví dụ cho biết làm thể nào để dung thư viện jQuery SignalR để truyền thông với một SignalR hub. Các nhiệm vụ trong mã diễn tả một proxy tham chiếu đến hub, diễn tả một hàm server có thể gọi để đẩy nội dung đến các client, và bắt đầu kết nối để gởi các message tới hub.

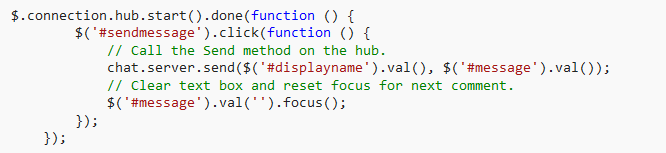
Code sau mô tả một tham chiếu đến một hub proxy



Đoạn mã dưới đây chỉ ra cho bạn làm thế tạo để tạo một lời gọi hàm trong script. Hub class trên server gọi hàm này để đẩy nội dung update đến mỗi client. 2 dong HTML encode nội dung trước khi hiển thị nó là tùy chọn và hiển thị một cách đơn giản để ngăn chặn script injection.



Đoạn mã dưới đây chỉ ra cách mở một kết nối với hub. Code bắt đầu kết nối và chuyển nó một hàm để xử lý sự kiện click trên button Send trên trang web



## **Ứng dụng bảng giá chứng khoán trực tuyến**

### **8.2.1 Giới thiệu**

* + Ứng dụng web thời gian thực hiển thị biến động của thị trường chứng khoán (Hiển thị bảng giá chứng khoán trực tuyến).

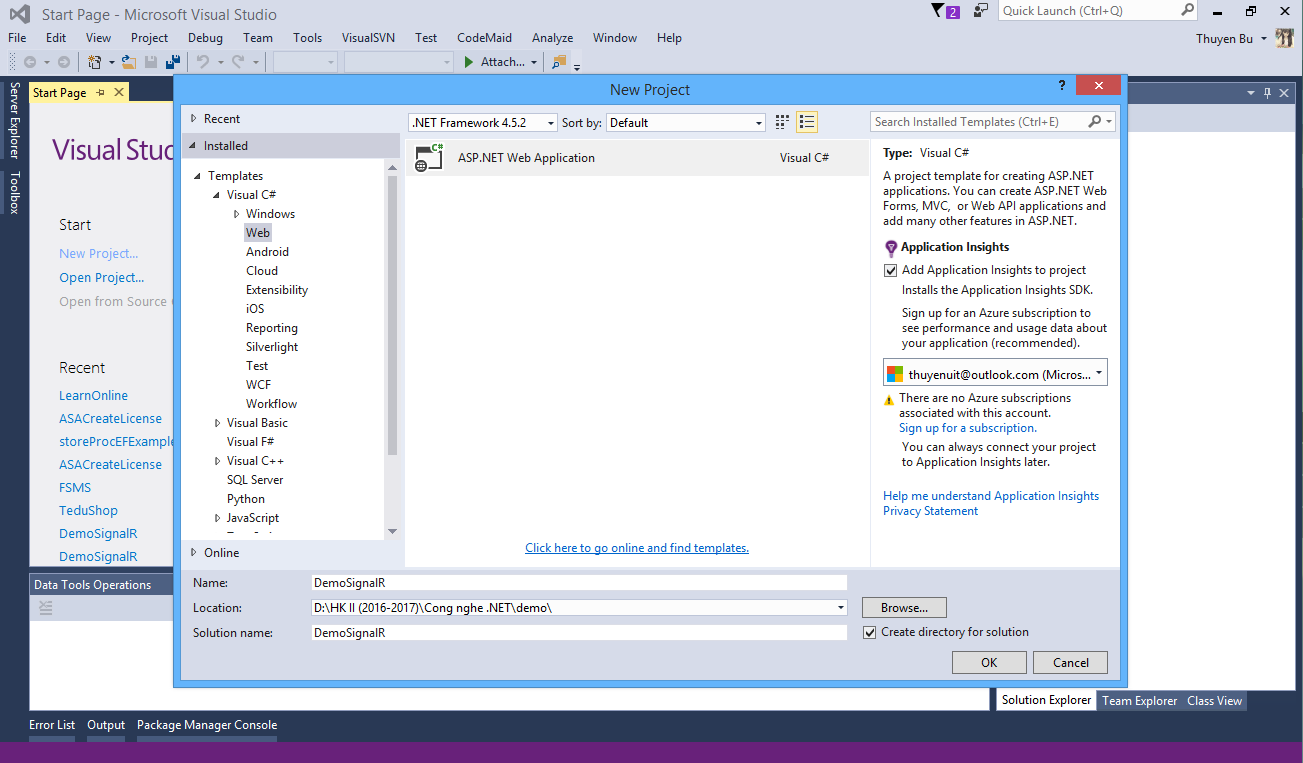
### **8.2.2 Cài đặt project**

Yêu cầu: Chuẩn bị Visual Studio 2013 trở lên

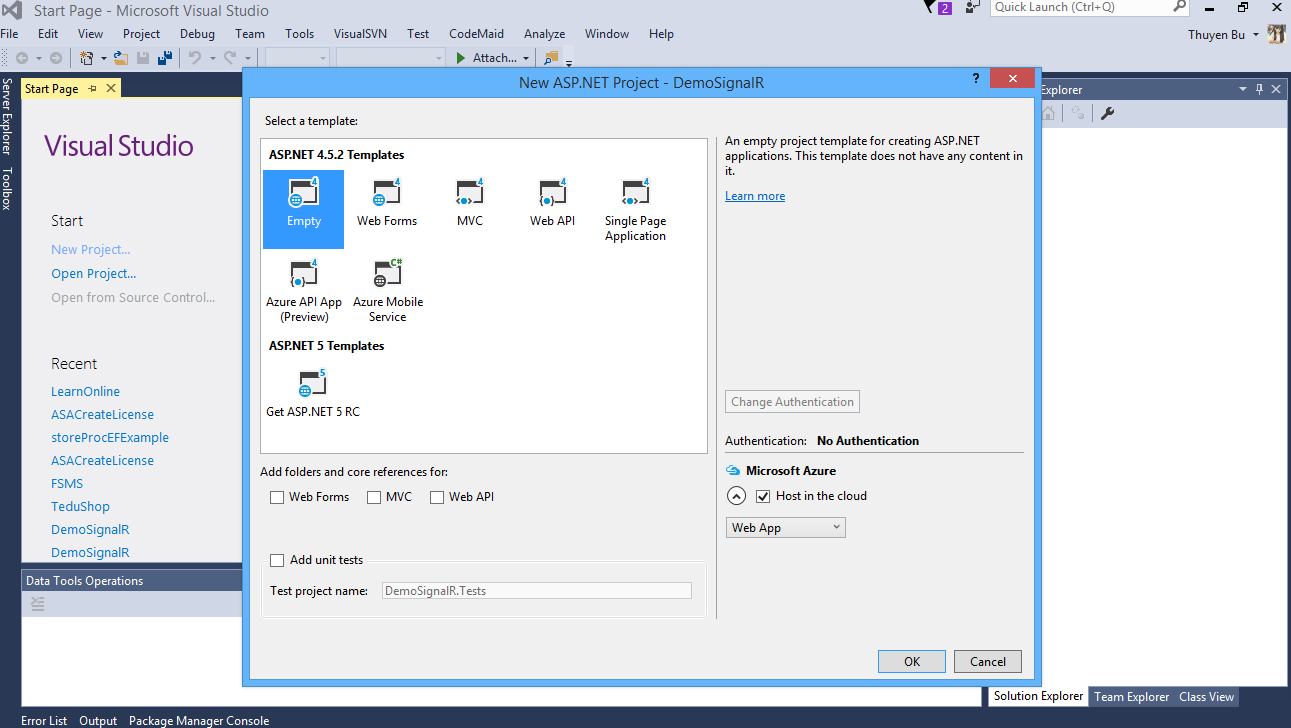
Bước 1: New project.

Chọn **File** -> **New project**

Chọn **Templates** -> **Web** và đặt tên cho project là **DemoSignalR** và click **OK**.

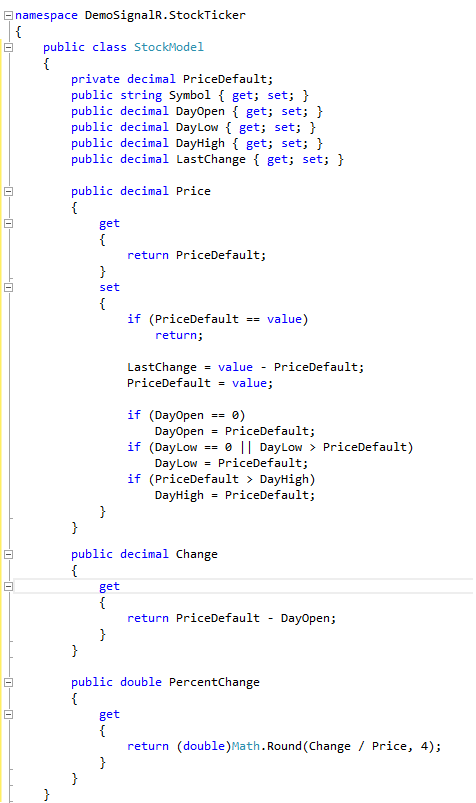


Một của sổ New ASP.NET Project hiện ra, chọn **Empty** và click OK.



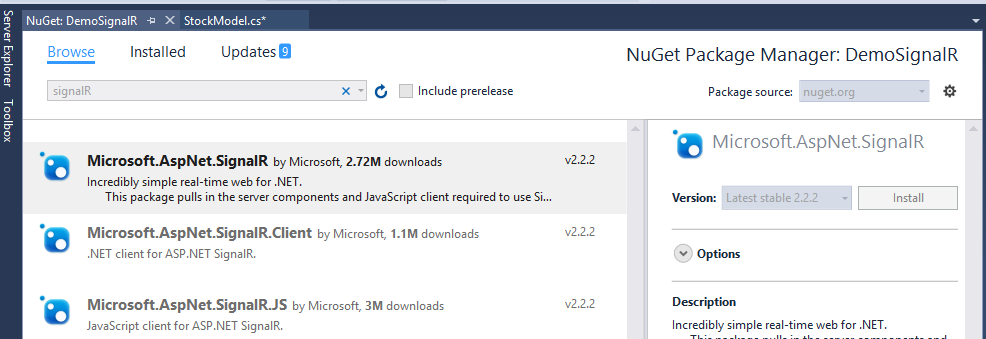
Bước 2: Tạo một thư mục **StockTicker** trong project **DemoSignalR**

Tạo một class **StockModel.cs** trong thư mục **StockTicker** và mã code trong class như sau:

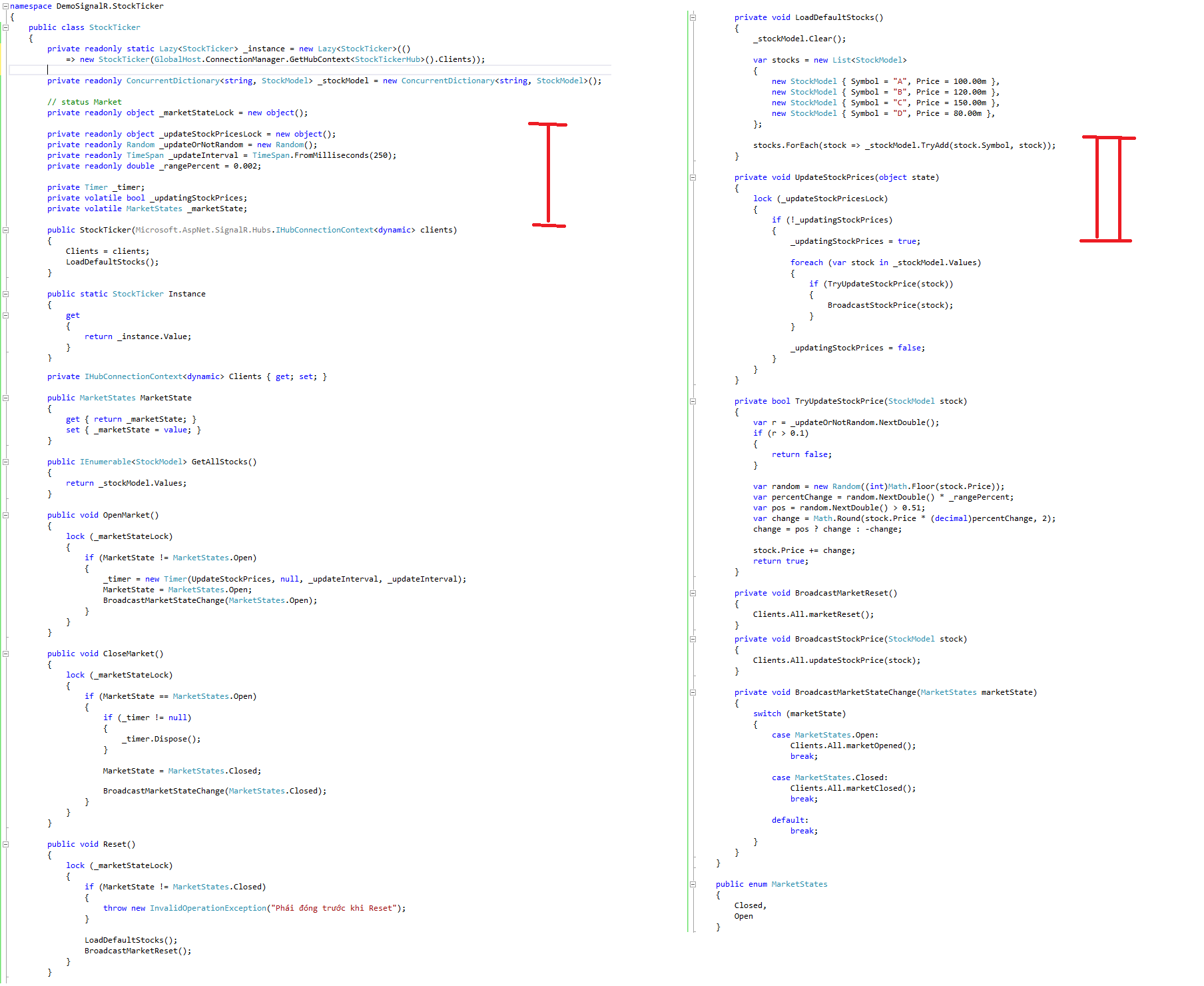


Bước 3: Tạo StockSticker và StockStickHub class

Click vào **References** -> **Manage Nuget Packages**… để cài đặt **Microsoft.AspNet.SignalR** packge để tải thư viện **SignalR**



Tạo một StockTicker.cs class trong thư mục StockStick và mã code như sau:



public class StockTicker

{

private readonly static Lazy<StockTicker> \_instance = new Lazy<StockTicker>(()

=> new StockTicker(GlobalHost.ConnectionManager.GetHubContext<StockTickerHub>().Clients));

private readonly ConcurrentDictionary<string, StockModel> \_stockModel = new ConcurrentDictionary<string, StockModel>();

// status Market

private readonly object \_marketStateLock = new object();

private readonly object \_updateStockPricesLock = new object();

private readonly Random \_updateOrNotRandom = new Random();

private readonly TimeSpan \_updateInterval = TimeSpan.FromMilliseconds(250);

private readonly double \_rangePercent = 0.002;

private Timer \_timer;

private volatile bool \_updatingStockPrices;

private volatile MarketStates \_marketState;

public StockTicker(Microsoft.AspNet.SignalR.Hubs.IHubConnectionContext<dynamic> clients)

{

Clients = clients;

LoadDefaultStocks();

}

public static StockTicker Instance

{

get

{

return \_instance.Value;

}

}

private IHubConnectionContext<dynamic> Clients { get; set; }

public MarketStates MarketState

{

get { return \_marketState; }

set { \_marketState = value; }

}

public IEnumerable<StockModel> GetAllStocks()

{

return \_stockModel.Values;

}

public void OpenMarket()

{

lock (\_marketStateLock)

{

if (MarketState != MarketStates.Open)

{

\_timer = new Timer(UpdateStockPrices, null, \_updateInterval, \_updateInterval);

MarketState = MarketStates.Open;

BroadcastMarketStateChange(MarketStates.Open);

}

}

}

public void CloseMarket()

{

lock (\_marketStateLock)

{

if (MarketState == MarketStates.Open)

{

if (\_timer != null)

{

\_timer.Dispose();

}

MarketState = MarketStates.Closed;

BroadcastMarketStateChange(MarketStates.Closed);

}

}

}

public void Reset()

{

lock (\_marketStateLock)

{

if (MarketState != MarketStates.Closed)

{

throw new InvalidOperationException("Phái đóng trước khi Reset");

}

LoadDefaultStocks();

BroadcastMarketReset();

}

}

private void LoadDefaultStocks()

{

\_stockModel.Clear();

var stocks = new List<StockModel>

{

new StockModel { Symbol = "A", Price = 100.00m },

new StockModel { Symbol = "B", Price = 120.00m },

new StockModel { Symbol = "C", Price = 150.00m },

new StockModel { Symbol = "D", Price = 80.00m },

};

stocks.ForEach(stock => \_stockModel.TryAdd(stock.Symbol, stock));

}

private void UpdateStockPrices(object state)

{

lock (\_updateStockPricesLock)

{

if (!\_updatingStockPrices)

{

\_updatingStockPrices = true;

foreach (var stock in \_stockModel.Values)

{

if (TryUpdateStockPrice(stock))

{

BroadcastStockPrice(stock);

}

}

\_updatingStockPrices = false;

}

}

}

private bool TryUpdateStockPrice(StockModel stock)

{

var r = \_updateOrNotRandom.NextDouble();

if (r > 0.1)

{

return false;

}

var random = new Random((int)Math.Floor(stock.Price));

var percentChange = random.NextDouble() \* \_rangePercent;

var pos = random.NextDouble() > 0.51;

var change = Math.Round(stock.Price \* (decimal)percentChange, 2);

change = pos ? change : -change;

stock.Price += change;

return true;

}

private void BroadcastMarketReset()

{

Clients.All.marketReset();

}

private void BroadcastStockPrice(StockModel stock)

{

Clients.All.updateStockPrice(stock);

}

private void BroadcastMarketStateChange(MarketStates marketState)

{

switch (marketState)

{

case MarketStates.Open:

Clients.All.marketOpened();

break;

case MarketStates.Closed:

Clients.All.marketClosed();

break;

default:

break;

}

}

}

public enum MarketStates

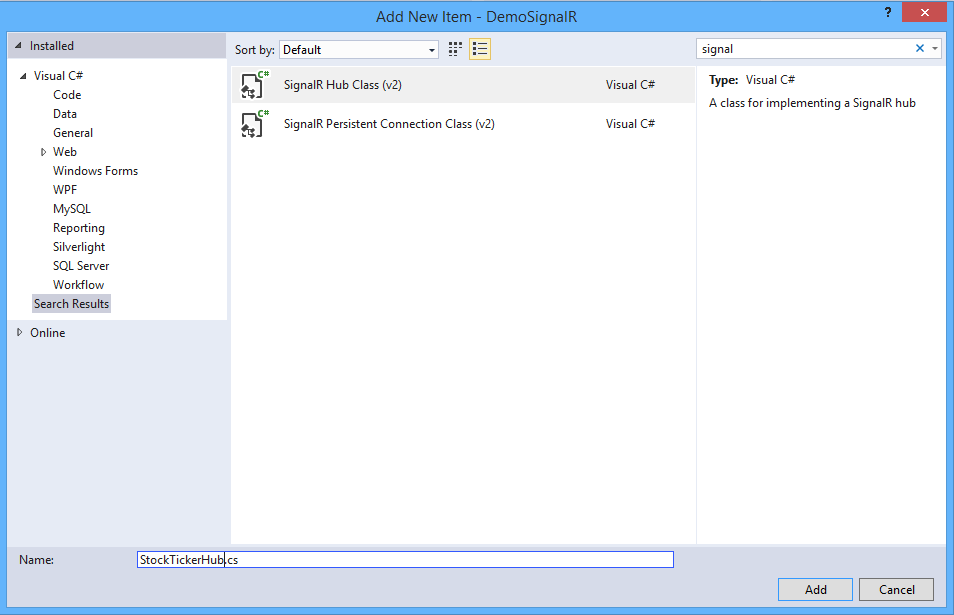
{

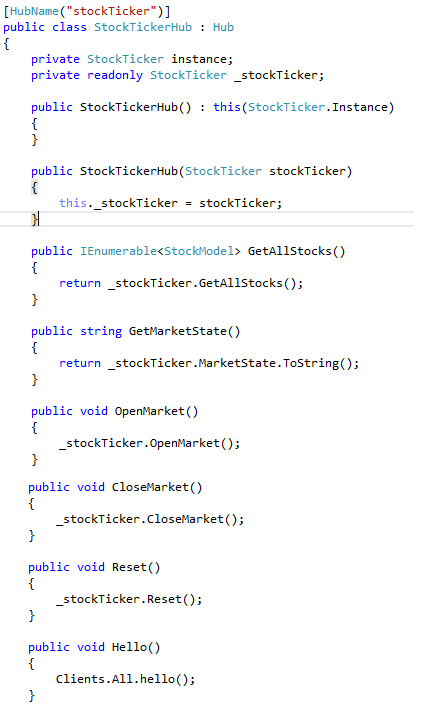
Closed,

Open

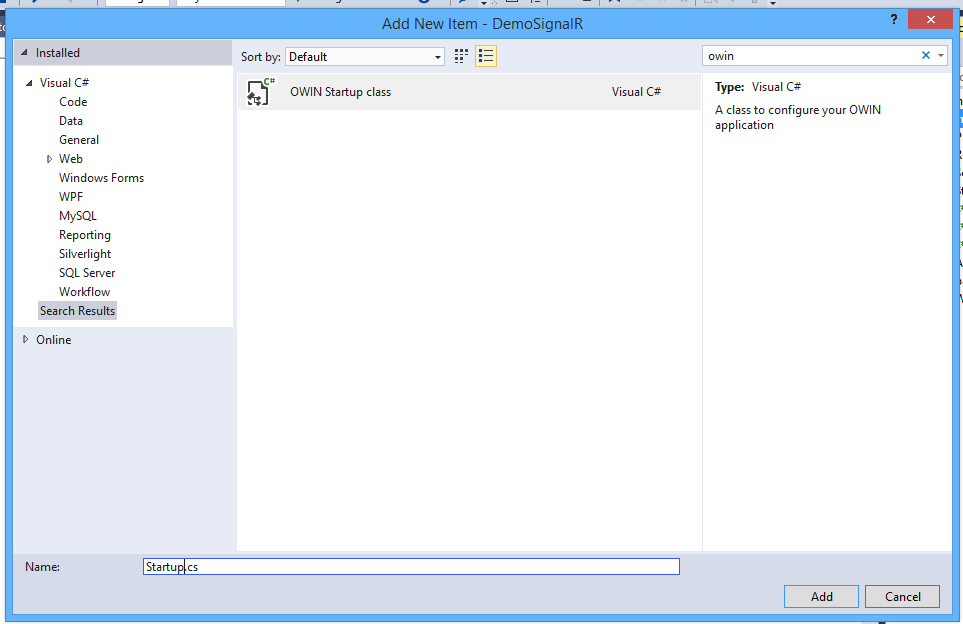
}

Tạo một class SignalR Hub có tên là **StockStickHub.cs** và mã code của hub này như sau:

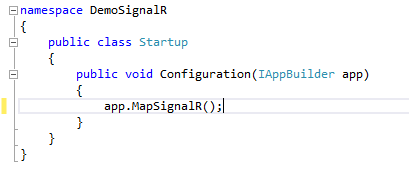




Bước 4: Tạo một OWIN Startup Class

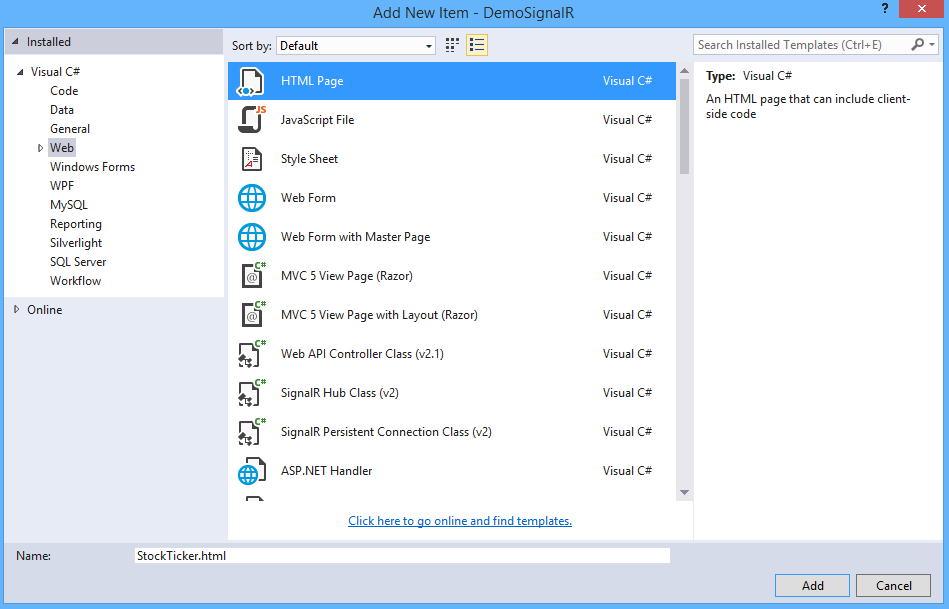


Map SignalR vào **Startup.cs** class vừa tạo

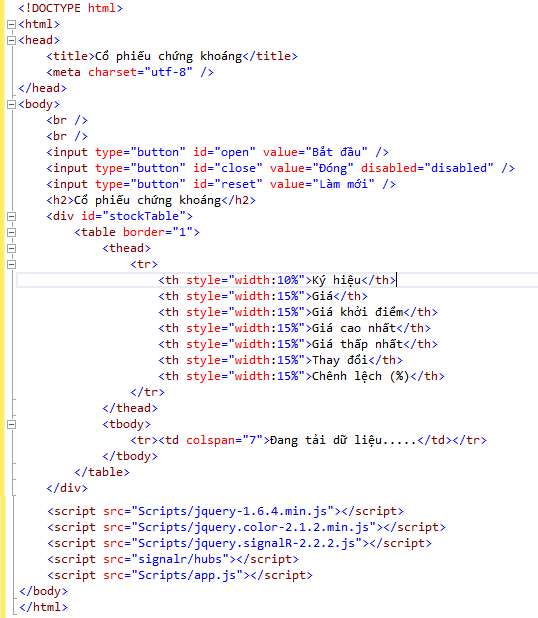


Bước 5 : Tạo view StockTicker bằng HTML đặt tên là **StockTicker.html**

Click DemoSignalR -> Add -> New Item -> Đặt tên cho view và click OK



Mã code view như sau:



Lưu ý:

* jquery.color-2.1.2.min.js có thể tải tại địa chỉ sau: <https://github.com/jquery/jquery-color> sau đó add vào thư mục Scripts của project
* app.js sẽ được thực hiện ở Bước 6

Bước 6: Tạo một JavaScript file đặt tên **app.js** trong thư mục **Scripts** tương tự như tạo view HTML và mã code như sau:

if (!String.prototype.supplant) {

String.prototype.supplant = function (o) {

return this.replace(/{([^{}]\*)}/g,

function (a, b) {

var r = o[b];

return typeof r === 'string' || typeof r === 'number' ? r : a;

}

);

};

}

// hiệu ứng màu

jQuery.fn.flash = function (color, duration) {

var current = this.css('backgroundColor');

this.animate({ backgroundColor: 'rgb(' + color + ')' }, duration / 2)

.animate({ backgroundColor: current }, duration / 2);

};

$(function () {

var ticker = $.connection.stockTicker,

up = '▲',

down = '▼',

$stockTable = $('#stockTable'),

$stockTableBody = $stockTable.find('tbody'),

rowTemplate = '<tr data-symbol="{Symbol}"><td>{Symbol}</td><td>{Price}</td><td>{DayOpen}</td><td>{DayHigh}</td><td>{DayLow}</td><td><span class="dir {DirectionClass}">{Direction}</span> {Change}</td><td>{PercentChange}</td></tr>',

$stockTicker = $('#stockTicker'),

$stockTickerUl = $stockTicker.find('ul'),

liTemplate = '<li data-symbol="{Symbol}"><span class="symbol">{Symbol}</span> <span class="price">{Price}</span> <span class="change"><span class="dir {DirectionClass}">{Direction}</span> {Change} ({PercentChange})</span></li>';

function formatStock(stock) {

return $.extend(stock, {

Price: stock.Price.toFixed(2),

PercentChange: (stock.PercentChange \* 100).toFixed(2) + '%',

Direction: stock.Change === 0 ? '' : stock.Change >= 0 ? up : down,

DirectionClass: stock.Change === 0 ? 'even' : stock.Change >= 0 ? 'up' : 'down'

});

}

function stopTicker() {

$stockTickerUl.stop();

}

// khởi tạo

function init() {

return ticker.server.getAllStocks().done(function (stocks) {

$stockTableBody.empty();

$.each(stocks, function () {

var stock = formatStock(this);

$stockTableBody.append(rowTemplate.supplant(stock));

});

});

}

// bên client thêm phương thức hub khi server gọi

$.extend(ticker.client, {

updateStockPrice: function (stock) {

var displayStock = formatStock(stock),

$row = $(rowTemplate.supplant(displayStock)),

bg = stock.LastChange < 0

? '255,148,148' // red

: '154,240,117'; // green

$stockTableBody.find('tr[data-symbol=' + stock.Symbol + ']')

.replaceWith($row);

$row.flash(bg, 1000); // call funtion tạo hiệu ứng màu

},

marketOpened: function () {

$("#open").prop("disabled", true);

$("#close").prop("disabled", false);

$("#reset").prop("disabled", true);

},

marketClosed: function () {

$("#open").prop("disabled", false);

$("#close").prop("disabled", true);

$("#reset").prop("disabled", false);

stopTicker();

},

// reset

marketReset: function () {

return init();

}

});

// Bắt đầu connection

$.connection.hub.start()

.then(init)

.then(function () {

})

.done(function (state) {

if (state === 'Open') {

ticker.client.marketOpened();

} else {

ticker.client.marketClosed();

}

// open

$("#open").click(function () {

ticker.server.openMarket();

});

// clode

$("#close").click(function () {

ticker.server.closeMarket();

});

// reset

$("#reset").click(function () {

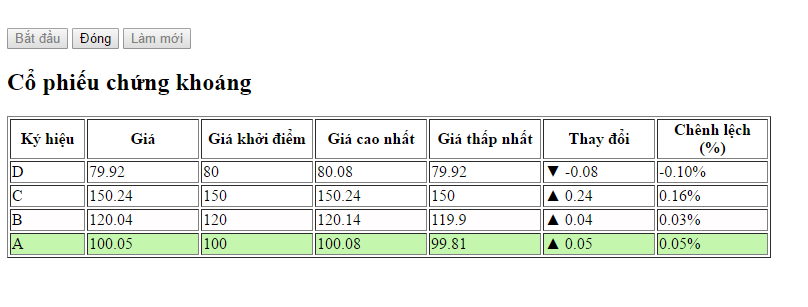
ticker.server.reset();

});

});

});

Bước 7: Run project



# Tài liệu tham khảo

* Book: SignalR Programming in Microsoft ASP.NET.
* https://www.asp.net/signalr.