Introduction

You will be writing a simplified chat service. You will not need to implement any kind of chat client; the purpose is to provide a service that other developers could consume. This is not a duplex service; clients will initiate queries to retrieve messages for a user. The design and implementation of the service may require some thought, but should not be tricky or especially time-consuming. Our goal is not to challenge; the right candidate should not find this test challenging. Instead, we are interested in getting a basic sense of your coding style and abilities.

Functional Requirements

Design and implement a service which allows users to send messages to each other directly and by subscribing to channels. A channel is a named group of users who have elected to broadcast messages to each other. Users and channels are each uniquely identified by a string. A message is also a string. All strings are legal; you do not need to perform input validation.

Users do not have to be authenticated. The service should be able to handle requests for users it has never seen before. There is no such thing as a user which "does not exist".

Users may elect to subscribe to and unsubscribe from channels. Channels are created on-demand when a user subscribes to one by name for the first time. For example, "testuser" may wish to subscribe to a channel "testchannel". This channel does not exist, and is created on demand. "othertestuser" may also wish to subscribe to "testchannel", in which case she will be associated with the existing channel. Both users would then be a part of the channel and would be able to communicate to each other by broadcasting to the channel.

You must support the following functions. Remember that you are implementing a service that clients will consume, and that in turn will be used by users who identify themselves by name. Do not confuse clients with users.

- Any user must be able to subscribe to any channel. If the user is already subscribed, nothing happens, but no errors occur. A channel can have any number of users.
- A user must be able to unsubscribe from a channel. If the user is not subscribed, nothing happens, but no errors occur.
- A user must be able to send a message directly to another user, by name. A user can send any number of messages.
- A user must be able to broadcast a message to a channel, if the user is subscribed to that channel. If the user is not subscribed to that channel, then nothing happens, and the failure is gracefully communicated back to the client. If the message is broadcast, then all users subscribed to that channel should receive it.
- A user must be able to query for a list of all messages they have received, whether directly from other users or from channel broadcasts. Each message should include a receipt timestamp, the name of the user who sent it, and if applicable, the name of the channel to which it was broadcast. A user can receive any number of messages.

Technical Requirements

- You must implement this as a web service in C#. We recommend using the ASP.NET Web API framework or Windows Communication Foundation (WCF), but you can choose any reasonable framework for web service development.
- Your service should be buildable in Visual Studio 2013 or 2015.
- Your service should be self-hosting or use IIS Express.
- Your service may not have any external runtime dependencies, including databases. Your service should maintain state between requests. It is, of course, expected that all state is lost when the service host is shut down.
- You may use NuGet packages to solve problems which are not domain-specific. More specifically, you may use packages to simplify or remove the boilerplate of building a web service, but not to implement chat-specific requirements.
- You do not need to synchronize access to the service or its state, unless you configure the service in a manner that requires it.
- The service is not expected to be scalable, but its operations should be efficient. Operations should not become unnecessarily slow as the size of the service's state increases.
- You must provide unit tests using a framework of your choice.

Guidelines and Deliverables

- We want to see code. Your entire solution, along with any documentation or supplementary materials, should be provided in source form in a publicly available repository (we recommend GitHub, but the choice is yours). We should be able to check out and build your code from source.
- Your repository should have meaningful history. That is, you should commit as though you were working on this project professionally. Do not complete and test the project and provide a single commit (to wit, we would not find this reasonable in a professional setting).
- Your code should pass your unit tests. Your unit tests should be meaningful and provide good coverage.
- We care most about correct functionality. But we care nearly as much about readability. Your code should be clear, concise, and easily maintainable and extendable.
- You can use any resources at your disposal, as long as the code is yours.
- There is no time limit, but the task is not a large one. Let us know how long you spent on it.

If you have any questions, feel free to write back and ask.