qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm

|  |
| --- |
| Project Report  BlahBox Application  4/28/2014  Owen Byrne (x13117343), Shane Burke (x13109839), Donal Devine (x13117611), Denis Stepanenko (x13100793) |

Contents

[Motivation 4](#_Toc386818736)

[Project Scope 5](#_Toc386818737)

[Functional Requirements 5](#_Toc386818738)

[Non-Functional Requirements 6](#_Toc386818739)

[Technical Requirements 6](#_Toc386818740)

[Area of contribution 6](#_Toc386818741)

[State of the Art Review 7](#_Toc386818742)

[Rich applications achieved through WebRTC 7](#_Toc386818743)

[Persistent Connections for Web Applications 9](#_Toc386818744)

[Ajax polling 9](#_Toc386818745)

[Long polling 10](#_Toc386818746)

[Browser Plugins 10](#_Toc386818747)

[WebSockets 10](#_Toc386818748)

[Signal R 10](#_Toc386818749)

[Responsive Design 10](#_Toc386818750)

[Twitter Bootstrap 11](#_Toc386818751)

[Skeleton 11](#_Toc386818752)

[Foundation 11](#_Toc386818753)

[Frameworks for Rich Internet Applications 11](#_Toc386818754)

[Angular 12](#_Toc386818755)

[Backbone 12](#_Toc386818756)

[Knockout 12](#_Toc386818757)

[Authentication/Authorisation 12](#_Toc386818758)

[SAML 13](#_Toc386818759)

[OpenID 13](#_Toc386818760)

[OAuth2 13](#_Toc386818761)

[Comparing Options 13](#_Toc386818762)

[Testing 14](#_Toc386818763)

[JavaScript Unit Testing 14](#_Toc386818764)

[Test Driven Development 14](#_Toc386818765)

[Behaviour Driven Development 15](#_Toc386818766)

[User Interface Design 17](#_Toc386818767)

[Responsive Framework 17](#_Toc386818768)

[Website Layout 17](#_Toc386818769)

[Navigation Bar 17](#_Toc386818770)

[Jumbotron 17](#_Toc386818771)

[Page Content 17](#_Toc386818772)

[How to improve 18](#_Toc386818773)

[Accessibility 18](#_Toc386818774)

[Browser Compatibility 18](#_Toc386818775)

[Architecture 19](#_Toc386818776)

[Application Architecture 19](#_Toc386818777)

[Security 21](#_Toc386818778)

[OAuth2 Integration 21](#_Toc386818779)

[ASP.Net Membership Provider 21](#_Toc386818780)

[Cross-Site Request Forgery (CSRF/XSRF) 21](#_Toc386818781)

[Toolkits and Frameworks 22](#_Toc386818782)

[AngularJS 22](#_Toc386818783)

[JQuery 24](#_Toc386818784)

[PeerJs 24](#_Toc386818785)

[Bootstrap 25](#_Toc386818786)

[Jasmine 26](#_Toc386818787)

[JQuery SignalR Library 26](#_Toc386818788)

[Data Transfer Strategies 26](#_Toc386818789)

[Evaluation and Testing 27](#_Toc386818790)

[Summary 28](#_Toc386818791)

[Bibliography 29](#_Toc386818792)

# Motivation

The motivation for this project was to design a Rich Internet Application utilising the latest client side web technologies. The solution we came up with comprises a real-time browser-to-browser client side video chat application primarily using client side technologies. We chose this type of application as we felt that it would demonstrate use of some of the rich client side technology that is available in some of the latest web browsers. One such technologies is WebRTC.

WebRTC is a recently available client side technology that enables web browsers to have Real-Time Communication (RTC) capabilities. This functionality can be accessed on the client side using the WebRTC JavaScript API. We felt that by using WebRTC we could demonstrate an understanding of the latest technologies available for use in a client side Rich Internet Application.

We also wanted to demonstrate an understanding of how to design and build a Single Page Application using a Model View Controller or Model View Whatever design pattern. We also wanted to show an understanding of one of the most important JavaScript frameworks available today. Therefore, we decided to build the application from the ground up using AngularJS. Using this approach we have architected a solution that demonstrates the use of the latest technologies as well as the latest ‘best practice’ in designing client side Rich Internet Applications.

# Project Scope

The Scope of this project incorporates designing a Single Page Application using the AngularJS JavaScript Framework. The solution will be architected using a Model View Controller design pattern. The solution design and implementation will incorporate the following principles:

* Behaviour Driven Development: Using this approach we will implement feature specifications and then implement the functionality to satisfy these specifications.
* Single Page Application architecture: The solution will be architected as a Single Page Application implementation using a client side web framework such as AngularJS.
* Model View Whatever design pattern: We will use the Model View Whatever or Model View Controller design pattern to segregate out the Models, Views and the controllers to make the solution more testable and maintainable.
* Testability: The solution should be testable. We will use Dependency Injection to ensure we can test application logic without having to worry about external dependencies.
* Responsive UI Design: The application User Interface should incorporate a responsive design. This means that the application will be capable of being rendered on multiple devices and the UI should resize itself to suit the device it is being accessed on.
* Security: The solution should incorporate security best practices. It should incorporate OAuth2 integration. The solution should also implement the logic to prevent a Cross-site request forgery (CSRF or XSRF) attack. In addition Cross-site Scripting attack is prevented by default as Angular escapes HTML unless "told" otherwise.
* REST: All server side logic and data should be presented via REST HTTP services. The client side should communicate with the server side logic and data via JSON using asynchronous Ajax requests.

## Functional Requirements

1. The user shall be able to login using a pre-existing account from a third-party vendor such as Google. There will be no need to register a propriety account with *Chatter*.
2. A profile page shall display all of the user’s viewable public details.
3. The user shall be able to view a video stream of their friend’s webcam when engaging in a chat with them.
4. The user shall be able to converse with their friend via voice communication when engaging with a chat session with them.
5. The user shall be able to converse with their friend via text when engaging in a chat session with them.
6. A chat history window will exist to store all of the previous message that were sent between the user and their friends.
7. There will be a facility to allow the user to add new contacts as well as deleting existing contacts
8. There will be a facility to search for friends so that they can be added to the contact list

## Non-Functional Requirements

1. All functionality for chatting with contacts will appear on a single page: video stream, chat feature, history, friend list, add/remove friends
2. The application will be responsive and therefore will render appropriately for different screen sizes including tablets and smart phones
3. All client side features will be tested using an appropriate test strategy and framework
4. The application will be limited to three pages: login page, profile page and main chat page

## Technical Requirements

1. Login – This will be implemented using Oauth2. This is an open standard will allow users to login to the application using credentials from a third party provider.
2. Communication protocol – PeerJS (peerjs.com). This will allow the use of API keys to realise a peer-to-peer connection.
3. Responsiveness - Bootstrap (getbootstrap.com). This will be used for making the web application responsive for rendering on various devices. The default bootstrap theme will be used for the application.
4. Data Storage – As this implementation is focused on client side technologies a simple file based databases will be used.
5. Testing – Jasmine (jasmine.github.io). Client side features will be tested using the Jasmine JavaScript framework.
6. Backend – ASP.NET. The web application will be managed using Visual Studio and will render from the default.apsx page.
7. Endpoints – ASP.NET Web API will be used for REST endpoints.

# Area of contribution

Our application solves a very common issue that is prevalent even today. To place a video or voice call from a compute, users generally need to download and install proprietary software either using a browser plugin or a standalone application. Our application builds on the recent trend where the web browser has become the application. Our application facilitates browser-to-browser real-time voice and video communication without requiring the installation of a plugin or software.

Most modern browsers have in built support for browser-to-browser communication. Our application access this functionality to deliver a real-time web based peer-to-peer communication solution. Our application utilises the WebRTC implementation in some modern browsers to deliver a rich internet application enabling browser-to-browser voice and video communication. WebRTC is a recent standard for communication between browsers, it enables browsers to communicate audio and video.

# State of the Art Review

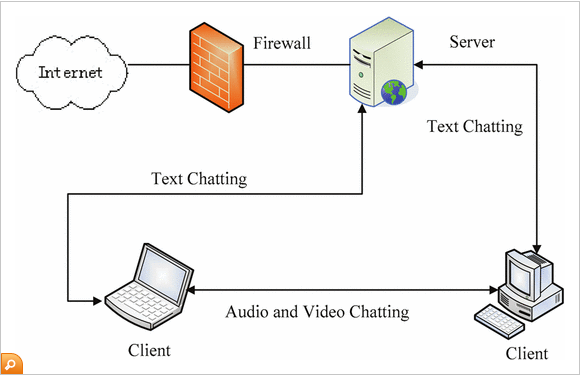
This section comprises a state of the art review for some of the technologies which were researched for developing the *BlahBox* chat application. As the application by definition is required to be a rich internet application, certain assumptions can be made. Firstly the application will be built using HTML5 and styled using CSS3, with a heavy reliance on JavaScript to provide heavy client side functionality. First, specific areas will need to be researched for the application. This will include looking at using WebRTC for a browser/server web-based application and researching methods for realising persistent connections between clients and servers or between peers. Research will also include looking at technologies needed for particular project deliverables such as security and testing. This will include pieces on client side testing frameworks and methods of authorisation/authentication. There will also be a general discussion of some of the JavaScript frameworks being used for developing rich internet applications. While not all technologies/frameworks were used in the final implementation, all were considered during the design phase of the application.

## Rich applications achieved through WebRTC

When creating a program for chatting over the web, one option is to build a native application. This will likely involve downloads, installs or plugins in order to operate. The other option is to use Web Real-Time Communication (WebRTC), a rapidly developing project supported by the World-Wide Web Consortium (W3C). This allows peer-to-peer communication through the browser to realise voice/video chat as well as file sharing, without the need for plugins. This is particularly suited for implementing an ARIA project as WebRTC’s mission statement is ‘to enable rich, high quality, RTC applications to be developed in the browser via simple JavaScript APIs and HTML5’ (webrtc.org, n.d.).

One of the major features of WebRTC is the use of the Web API from the W3C working group. Three major components of the API include [getUserMedia], [RTCPeerConnection] and [RTCDataChannels] (Hickson, 2014). All of these are important for achieving rich video-chat applications. Firstly, [getUserMedia] allows the browser to gain access to the user’s microphone and camera. Secondly, [RTCPeerConnection] facilitates audio and video calls. Finally, [RTCDataChannels] allows the sharing of data over a peer connection. Internally, WebRTC contains a Voice Engine, Video Engine and Transport Layer.

One of the major reasons for using WebRTC is to create a chat application. It can be considered as an attractive alternative to desktop application such as Skype. WebRTC promises ubiquitous endpoints capable of voice, video and Instant Messaging (Michels, 2013). This contrasts with Skype which is a hybrid of peer-to-peer and client-server, where resources are used from the client, as the software is installed on their device. Also, WebRTC provides an open set of capabilities while Skype provides proprietary software with limited access to its public API. WebRTC may be described as having a browser/server (b/s) architecture rather than a client/server (c/s) architecture. Consider the following diagram:



(Yuzhuo, 2013)

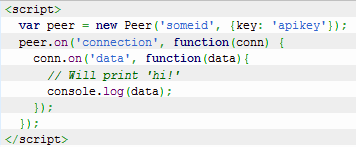
The above is an example implementation for a rich internet chat application. Here the chat messages are forwarded between browsers, realised through HTML5 WebSocket technology. Audio and video chatting is through a point-to-point connection, using WebRTC for establishing a connection between browsers. This is a simple prototype for achieving communication through browsers without any software or plugins.

One of the most important parts of using WebRTC for chatting is the establishing of a peer connection. The logical flow of the WebRTC peer connection is as follows:

* User A: (i) Starts video stream, (ii) creates RTCPeerConnection, (iii) creates offer, (iv) sends offer to server
* Server: Sends offer to User B
* User B: (i) Receives offer, (ii) starts video stream, (iii) creates peer connection, (iv) attaches own video stream to RTCPeerConnection, (v) receives User A’s video stream, (vi) sends answer to server
* Server: Sends answer to User A
* User A: (i) Receives answer, (ii) receives User B’s video stream

(Cohen, 2014)

One technology which makes use of the RTCPeerConnection is *Peer.js,* which is intended to wrap the browser’s implementation of WebRTC to allow a user to create a media stream to a remote peer (PeerJS, n.d.). The physical architecture of Peer.js consists of a client side script for communicating with other clients over WebRTC, and a Node.js server component for brokering connections between the clients (Govett, 2013). The server keeps track of all users that are online so that other users can have knowledge of these peers. RTCPeerConnection is then used so that a user can connect to any of these peers. In terms of implementation, only the id of the other peer is needed in order to establish a connection. The following is an example of creating a peer using Peer.js:



(Petereit, 2013)

An example of a connection to a peer being established is as follows:



## Persistent Connections for Web Applications

One major requirement for the proposed chat application is that it needs to work in real time. The main protocol used on the web is HTTP. The problem here is that it works on a request-response style of communication. This means that the client requests data and the server responds with data. The issue is that with a chat application, information such as instant messages need to be sent by the server is real time, without a request from the client. Therefore an open channel allowing the client and server to have a full-duplex bidirectional communication with each other is needed for transferring data (Basu, 2013). Some common methods of achieving this style of persistent connection are listed below.

## Ajax polling

This method does not implement a true open channel but it is a commonly used method for receiving refreshed information. In this case polling occurs at regular intervals. An AJAX request is sent to the server and any new information is retrieved each time. One issue is that this is not real time as the user has to wait for the next poll interval in order to see the new data. The other problem is that the request will occur whether or not there is new information. Therefore if there is a long period of inactivity on the server then there will be a large number of wasted requests.

## Long polling

This form of polling differs in that it does not use a large number or requests at short intervals but uses the same request for a long period of time. The client sends a request and then the server ‘holds’ the request open by managing time-outs and thread issues. This is done to give an always connected feeling.

## Browser Plugins

A powerful alternative to polling is to use browser plugins which can support sockets to allow two-way communication over an open connection. The major issue here is that it is dependent on the browser, requiring the user to have the plugin installed. Also, the area of rich internet applications is largely moving away from plugins through technologies such as HTML5 and WebRTC.

## WebSockets

The standardising of WebSocket protocols in 2011 has led the way for true two-way communication between a web browser and a web server. This is achieved by standardising how a server can send information when they have not received an explicit request from the client. This allows data such as streaming media and instant messages to be sent to the client over a single TCP connection. WebSockets are supported by most modern browsers and communicate over port 80 which is useful for firewalls which block connections on other (non-web) ports.

## Signal R

SingalR is an asynchronous signalling library for ASP.NET which is used to build real-time multi-user web applications (Hanselman, 2011). It is a complete client-server solution for persistent connections using JavaScript on the client-side and ASP.NET on the server side. It is similar to technologies such as Socket.IO and NowJS, which are JavaScript libraries that expect some form of convention on the server side for communication. It is particularly suited to those using Microsoft technologies and as an alternative to WebSockets where there is a lack of support. The main feature of SignalR is that it abstracts the transportation mechanism used for the persistent connection (Basu, 2013). Therefore the developer states that they need a persistent connection and SignalR will use the transport mechanism that it deems most suitable. It can use any combination of mechanisms such as the polling techniques listed above, or other mechanisms such as forever frames and server-sent events.

## Responsive Design

Rich internet applications have traditionally been achieved using plugins such as Adobe Flash and Microsoft Silverlight. Developing RIAs for mobile devices has generally been seen as distinct from desktop applications due to the need for ubiquity and context awareness. These are known as rich mobile applications (RMAs). However the increasing move from browser plugin based applications for both desktop and mobile has seen HTML5 emerge as a forerunner for Rich Internet Applications (Bishop, 2011). Due to the widespread cross-compatibility of HTML5 among devices, it is becoming increasingly possible to develop one application to suit all devices (desktops, laptops, tablets, phones etc.). Responsive design is a development process where an application responds to the user’s behaviour and environment based on their screen size, platform and orientation (Knight, 2011). The idea is to eliminate the need for a new design and development phase to support each emerging device. The following is a look at some of the frameworks that are available for aiding with responsive design.

### Twitter Bootstrap

Twitter’s bootstrap is the most popular framework and is used on many popular websites. It is one of the most feature-heavy frameworks for responsive design and was developed to encourage consistency across applications. Bootstrap has a 12-grid responsive layout, 13 JQuery plugins for common UIs and a Bootstrap customizer (Gube, 2013). It is an open source project, is well documented and is the number one project on GitHub in terms of fork and stars (GitHub, n.d.).

### Skeleton

Skeleton is a CSS boilerplate designed for HTML5 apps and is designed with a focus on being simple and clean. Skeleton is not considered as a framework in the traditional sense, as it is more of development kit, providing a basic foundation but also allowing for a large amount of custom styling (Skeleton, n.d.). It is designed to be used with rapid development and is recommended for those who would like to get started quickly with responsive design.

### Foundation

Another popular framework is Foundation, whose selling point is the ability to design first for mobile devices. This allows the mobile version of the application to be highly optimised and un-cluttered, with features being added afterwards for different versions (Designer, 2013). It also supports rapid prototyping, and like most of the frameworks, supports a responsive grid system.

## Frameworks for Rich Internet Applications

Along with the specific technologies to aid the implementation of *BlahBox*, there are a number of frameworks which are useful for rich internet applications in general. In general these frameworks all generate HTML dynamically in the browser to take advantage of resources on the client side. Frameworks can help with UI rendering, separation of concerns, two-way binding and client-side routing. Advantages to using frameworks in general include efficiency due to increased functionality in fewer lines of code, increased protection due to inbuilt security features, and community support (D'Oliveira, 2011). Disadvantages include the limitations of the specific framework and the fact that the developer learns the frameworks rather than the language itself. The following is a brief description of the some of the state of the art frameworks being used in industry for rich internet applications.

### Angular

Angular is an MV\* (Model-View-Whatever) framework for JavaScript. It uses DOM-based templates and is heavily focused on Test Driven Development. It has specific conventions and modes of operation that the developer is expected to adhere to in order to use efficiently. Angular is designed to build complex client-side applications whose unique selling point is the ability is to extend HTML by creating custom tags and attributes (Stovall, 2013). Angular is seen as a powerful framework, but is often noted for its complexity, with developers having to invest a lot of time to get up and running. For example the documentation has been described as “incredibly thin...briefly covers each topic...and suffers from a lack of code examples” (Pretorius, 2013).

### Backbone

Backbone differs from Angular in that it has no defined way of doing things or conventions for developing. This means that it is largely up to the developer to use backbone in any way they would like. This could be either an advantage or disadvantage depending on the viewpoint. Therefore there is a trade-off between the flexibility of leaving decisions to the developer and the lack of support from having few conventions (Smus, 2012). It is based on the MVP (Model-View-Presenter) pattern, and is used for developing single-page applications and for keeping all parts of the application synchronised. Another thing to note is the application’s dependency on the Underscore.js framework.

### Knockout

Knockout is another framework with similar functionality for generating HTML. Its pattern is based on MVVM (Model-View-ViewModel). It is heavily focused on using declarative two-way binding. Some of the key features include automatic UI refresh, tracking dependencies and templating using native or third-party template engine. Advantages include the ability to control bindings at a granular level, while disadvantages include its reliance on plugins due to no native implementation of paging or routing.

## Authentication/Authorisation

The system will require some mechanism for identifying users when they visit the website. Identification to any application is generally required for two reasons: authentication and authorisation. We need to be sure that the user is who they say they are and that they are allowed to perform relevant actions within the system. For the proposed chat application, this will mostly involve identifying users on the network so that they can communicate with other users. For this project we explored using forms of federated identity as an alternative to a proprietary server-side login system. The discussion below will involve the major providers, security concerns and the advantages and disadvantages.

### SAML

Security Assertion Markup Language (SAML), developed by OASIS is an XML-based framework for user authentication and entitlement (Klingenstein, 2012). There are three separate entities involved in this communication protocol: the principal, the service provider and the identity provider. Here the user (principal) visits a website (service provider), which redirects to a third party sign-in application (identification provider). In technical terms, this involves a SAML request sent to the identification provider, which is parsed and then a SAML response is sent back to service provider who then knows whether to authenticate the user or not.

### OpenID

OpenID is an interoperable authentication protocol based on Oauth2. This is implemented using JSON/REST message flows and is designed with the goal of allowing developers to authenticate users across sites without having to manage password files (OpenID.NET, n.d.). Unlike SAML, OpenID works on a relay system. Once again, there are three parties: end-user, OpenID provider and relayer. Here the user visits the website (relayer), identifying their OpenID provider. The relayer then finds and creates an association with the OpenID provider before redirecting the user. Here, the user logs in with their credentials and is verified by the OpenID provider. The user then presents this verification to the relayer/service provider.

### OAuth2

OAuth is similar in many ways to OpenID but is solely concerned with authorisation rather than authentication. Its aim is to provide an open protocol to facilitate secure authorisation in a standardised way for web, desktop and mobile web applications (OAuth.NET, n.d.). Here the three parties involved are the end user, the OAuth provider and the OAuth consumer. The user visits a website (OAuth consumer) which in return requests use of the users access token from the OAuth provider. A communication then takes place between the provider and the user, where they agree to share their token. This token is then sent to website who in turn includes the token each time they make a request from the provider.

### Comparing Options

SAML was one of the forerunners of single sign-on options. However, is has been accused of ‘showing its age’, only being suited to web applications rather than native mobile applications (Dennis, 2013). This is due to the requirement that HTTP Post method must be used when sending tokens to the server. There are also some security concerns with its reliance on XML. One study found that XML Signature Wrapping could be manipulated to impersonate any user (Somorovsky, 2011).

OpenID and OAuth share similar backgrounds and are in many ways complimentary techniques of identification. One of the main differences is that OpenID is solely for authentication while OAuth is purely for authorisation. In this way, OpenID lacks any real data exchange, therefore lacking any authorisation of specific information. This may be an advantage or disadvantage depending on the viewpoint.

In terms of security, one of OpenID’s biggest threats is that it is open to phishing attacks. This is because the visited website is responsible for redirecting the user to the OpenID provider. A malicious website could therefore redirect the user to a mocked-up OpenID provider in order to extract the user’s credentials. OAuth1 had a number of vulnerabilities including token fixing. OAuth2 has also been criticised as an ‘inherently insecure protocol’ with an overreliance on TLS and no support for signature, encryption, channel binding or client verification (Koussa, 2013). However it is also noted that OAuth is particularly suited to those looking for API-based solution between applications, and that with proper implementation there should be added convenience without further security threats.

## Testing

Testing and security considerations are major deliverables for this project. As the aim is to develop a rich internet application, the majority of the testing will likely focus on unit testing on the client side. This section will examine some of the JavaScript testing frameworks along with some of the testing methodologies being used in industry.

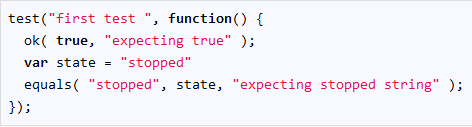
### JavaScript Unit Testing

One of the major challenges with testing code on the client-side is that there is often a lack of any real units. This differs from server side coding where the structure is largely modular with the functionality broken down into units making it suitable for testing. JavaScript may be mixed in with back-end logic or in worst cases may be written differently for each page of the application and completely mixed in with the HTML. (Zaefferer, 2012). This will often involve a process of ‘making things testable’, by splitting the JavaScript into testable units. This generally involves various staging of refactoring. However, unit testing is much more usable and easily applied if the code is written with a view to testing it. This can be achieved through methods such as Test Driven Development (TDD) and Behaviour Driven Development (BDD).

### Test Driven Development

Test-driven development evolved out of extreme programming (XP) and has since become a popular practice in many development lifecycles. The process involves writing test cases first and forcing them to fail. This is followed by writing enough code to pass the failing test case. After this, the code may be refactored and the tests can be run again. Some of the major advantages are that it helps to eliminate a number of defects before reaching the QA team, the code base is more efficient and it instils confidence that new code will not break existing functionality (Mardanov, n.d.). The concepts of TDD are often heavily linked to agile methodologies such as Continuous Integration, Scrum and Pair Programming.

One of the main frameworks used for unit testing on the client side is QUnit, which is part of the JQuery project, but does not necessarily have to be used for JQuery (QUnit, n.d.). It uses a syntax similar to the no longer supported FireUnit (for Firefox) and JSUnit (a port of JUnit for Java). An example of the coding style is:



(Zaparka, 2011)

This style of testing is more functional based rather than the behavioural based style seen with ‘Spec’ type languages. It is also easy to integrate, with only the qunit.js file and qunit.css file needed to write tests. When integrated it can be used for TDD, with rapid prototyping and re-running of tests. Other testing frameworks of a similar nature include screw.unit and JSUnit test.

### Behaviour Driven Development

Another option which evolved out of TDD is Behaviour-Driven Development (BDD). One of the driving ideas behind BDD is that it is designed to be accessible. This is particularly important to agile where the technical and non-technical team members are embedded in the same project. One of the main focuses of BDD is that it emphasises behaviour over testing. General rules include (North, 2006):

* Test method names should be plain sentences
* Behaviours map to requirements
* Provides ubiquitous language for analysis
* Acceptance criteria should be executable

This form of development is often described as following the process of given…when…then (Lero, 2011). For example if we need to write a test for viewing a webcam, it would be written as:

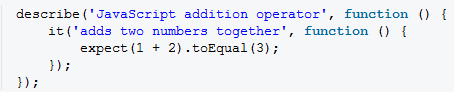
*Given…*that I need to communicate with other users

*When…*I choose the chat option

*Then…*I should see the user’s image

*And...*Hear the user’s voice

In terms of implementation, Jasmine is a popular framework for implementing BDD for JavaScript development. This involves writing short concise tests for a wide range of functionality. Such functions include matching results, spying, testing callbacks and mocking timeouts (Jasmine, n.d.). However the cases are written in such a way as to be meaningful even to non-technical stakeholders. Like with TDD this involves writing failing tests followed by writing code to pass the tests. Semantically, Jasmine is similar to RSpec and associated frameworks with its use of *describe…it* keywords. A simple example of a Jasmine test case can be seen below:



# User Interface Design

Chatter was designed using a simple approach. The idea behind the design was to utilise a responsive template and add our functionality within the template. We decided upon this approach because none of the team members were experienced designing a website interface from scratch, and it would have increased the development time to build a custom responsive template to work across all devices.

## Responsive Framework

We decided to use a 3rd party library to handle the responsiveness of the user interface. Twitter Bootstrap was selected as the framework to use, and an existing Bootstrap template was used. There were a couple of reasons why Bootstrap was used (Chouhan, 2013). The first is because Bootstrap is quick to get started with. We downloaded the template which allowed us to give a global look and feel to all web pages very quickly. The second reason was because Bootstrap included CSS for custom HTML elements such as text boxes, tables, icons and typography. This helped us to create custom HTML elements without adding any custom CSS or JS, to give our application a more custom look than using browser default elements. Another is reason is an extensive list of components. These include navigation bars, button groups and tabs. We used some of these components to help give a better mobile experience. The final reason is the inclusion of a JavaScript library which handles everything from mobile screens to custom components.

## Website Layout

The Chatter webpage contains a couple of different HTML sections. These include the navigation bar, jumbotron or header and the page content.

### Navigation Bar

The navigation is a default Bootstrap component. The bar contains the website logo and links to other web pages. This component is responsive and automatically adjusts to smaller screens. The advantage of using this, is because no matter how many links are in the navigation, the layout does not break on screens with less real-estate.

### Jumbotron

The jumbotron or header, is the next section on the web page after the navigation bar. It contains the page header, and has space to include a paragraph about the web page. This section is a different colour to the rest of the page to distinguish it and draw attention of the user to which page they are on. As it this is a simple section it doesn’t require any fancy transformation on smaller screens, as the content wraps neatly on all devices.

### Page Content

This section consumes the majority of the webpage and takes advantage of Bootstraps grid system. Bootstrap, by default uses a 12 column layout system. This means that on smaller screens, bootstrap will try to display all of the column beside each other, and will continue to wrap each column below that can’t be displayed correctly on the page. On our main page, the chat page, we use a use the grid system but only two grids which take up 6 columns each. The left column contains the list of friends, and the right column contains the video and chat sections. On smaller screens, the video and chat column will neatly wrap below the list of friends column.

The list of friends column contain tabs. This was implemented to save space between switching from current friends and finding friends. Both tabs also contain a pagination component. This is helpful not to overload the user with too many search results. The chat and video column also contains a pagination component for the chat history. This was implemented so a user would not be overloaded with too many messages if they have a large history.

### How to improve

The user interface on the chat page could be easily improved by allowing multiple chat sessions simultaneously. This feature was not implemented as the video call functionality is tightly coupled to active friend, so the interface was not correctly designed to handle this. A UI design that resembles Facebook chat windows may have been a better approach to take.

## Accessibility

The use of Bootstrap does not make any website fully accessible, but it does support accessibility best practices. In Chatter, we use a <h1> on each page which is a best practice requirement. The custom components used Bootstrap, support all browser default actions. For example, a form can navigated using the tab key, and buttons and be called by tabbing to and pressing enter.

## Browser Compatibility

Bootstrap provides pretty good compatibility among all devices and browsers including almost full support for Internet Explorer 8 and above. The noticeable exceptions include Firefox on Android devices, Opera on both Android and iOS and Safari on Windows. Although Bootstrap provides a consistent look and feel among a large of browsers and devices, it does not mean the website if fully supported on all these browsers and devices. The reason for this is because of WebRTC, as this has not be adopted by all major browsers it is not fully supported.

# Architecture

## Application Architecture

The application has been architected and designed in accordance with the principles outlined in the project scope:

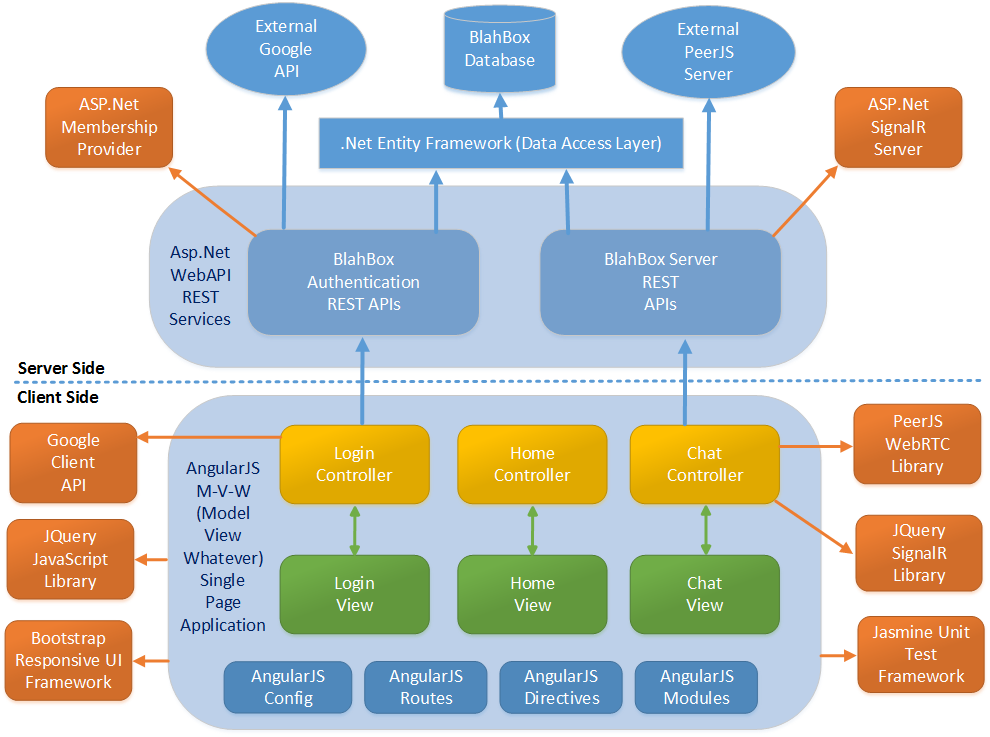
* Behaviour Driven Development: We have implemented test specifications using Jasmine.
* Single Page Application architecture: The solution has been architected and designed as a Single Page Application implementation using AngularJS.
* Model View Whatever design pattern: We have used the Model View Whatever design pattern as prescribed by the AngularJS framework.
* Testability: We have made extensive use of AngularJS Dependency Injection techniques to ensure we can fully test the client side application logic.
* Responsive UI Design: The application User Interface has been made fully responsive using the Twitter Bootstrap UI framework.
* Security: The application incorporates security best practices. It has incorporated OAuth2 integration using Google OAuth APIs. We have also implemented the logic to prevent a Cross-site request forgery (CSRF or XSRF) attack. This has been done using a combination of server side logic and AngularJS capabilities.
* REST: All server side logic has been interfaced via REST HTTP services. The client side logic communicates with the server side logic via JSON using asynchronous Ajax requests. This has been implemented using AngularJS constructs.

The client side architecture of the application consists of a Single Page Application based on the AngularJS MVW (Model View Whatever) framework. AngularJS is a prescriptive client-side JavaScript framework, which means it has a recommend approach to designing the architecture of web applications. We have stuck to this recommended approach in designing our application.

AngularJS provides us with a declarative programming approach for the User Interface and an imperative programming approach for the application logic. AngularJS allows us to decouple the DOM manipulation from the application logic.

The AngularJS architecture closely resembles the Model View Controller pattern, however it can be more accurately be described as a Model View Whatever pattern. This is because it borrows elements from both the Model View Controller and Model View ViewModel (MVVM) patterns.

The following diagram illustrates the main components of the BlahBox application architecture. The BlahBox application is a Single Page Application implemented using the AngularJS framework. A Single Page Application means that the application is hosted in a single page and views are loaded by the client side framework as and when required.



As can be seen from the diagram above, the basis of the client side of the BlahBox application is a Single Page Application. The Single Page Application is built using the AngularJS JavaScript Framework. The AngularJS JavaScript framework manages a Single Page Application that is designed using the Model View Whatever (MVW) design pattern. As a result of this we have architected our solution to consist of a set of Models, Views and Controllers. The Views and Controllers reside on the client side and the Models are retrieved through a series of Ajax requests to our custom REST HTTP services implemented on the server side.

One of the main benefits of a Single Page Application is that it is more fluid and responsive. There is less of the jarring effect of reloading and re-rendering the page. By sending the app data as JSON it creates a separation between the presentation (HTML) and application logic (AJAX requests plus JSON responses).

In a traditional web based application each page request would result in a request to the server and the HTML page would be returned and rendered on the client. A Single Page Application lifecycle is different in that the initial server request retrieves the HTML, JS and CSS to the client. Subsequent requests to the server are typically Ajax requests to retrieve JSON from REST based HTTP services.

The Models are accessed through XHR (XMLHttpRequest) requests to the Server APIs. The Server APIs are implemented using ASP.Net WebAPI REST HTTP Services. WebAPI is a server side framework for building HTTP services using the Microsoft .Net Framework. The WebAPI talks to the backend database via .Net Entity Framework. This framework provides an ORM (Object Relational Mapping) interface to the database.

## Security

Our application solution architecture has implemented security best practices at the design stage and during implementation. We have employed OAuth2 authentication and integrated it with ASP.Net Membership Provider authentication and AngularJS Cross Site Request Forgery prevention capabilities to ensure the application provides a robust security implementation.

### OAuth2 Integration

We have implemented OAuth2 authentication using Google’s authentication service. We have implemented this using an AngularJS directive as this provides us with a nice clean implementation in the Login View. Once the user authenticates with Google, this authentication is sent to the applications server side REST APIs to authenticate with the server using the ASP.Net Membership Provider.

### ASP.Net Membership Provider

We have used the ASP.Net Membership Provider to authenticate users on the server side. We have used this technology as it allows us to store user details for every user in a database. We have also used this technology as we can use the ASP.Net Membership Provider authentication token to create a special token which can be used to check XHR requests to our REST APIs for Cross-Site Request Forgery (CSRF/XSRF) attacks.

### Cross-Site Request Forgery (CSRF/XSRF)

We have implemented a complete and robust solution for preventing Cross Site Request Forgery (CSRF/XSRF) attacks. We have implemented this solution using security best practices and the latest capabilities of the technologies we have employed in our solution.

CSFR/XSRF attacks are seen as one of the worst kinds of vulnerabilities. This is because they are very easy to exploit by attackers and they are frequently misunderstood by developers. CSFR/XSRF vulnerabilities occur when an application allows an unauthenticated user to perform a sensitive action without verifying that the user is authorised to perform the action.

Typically, web application or website does not verify that a request came from an authorised user. In addition, sometimes the application or website will only check that the request came from the browser of an authorized user – not that it is an authorized user performing the action.

Because browsers can execute code from multiple sites, there is the possibility that a malicious site will send a request to a vulnerable site. The vulnerable site will mistakenly assume that the user is authorized to make the request.

To perform a malicious XSRF/CSRF attack a malicious web page just needs to obtain a validated login cookie from the user’s browser. The malicious web page then just needs to fool the user into performing an action – at which point the web page can use the cookie to convince the vulnerable site that the request is legitimate.

In order to prevent XSRF/CSRF attacks we have injected a XSRF/CSRF token into the HTTP header of XHR requests to the server. To do this we have leveraged the powerful in-built XSRF/CSRF attack prevention capabilities of AngularJS. All of our server requests are handled by AngularJS XHR.

Once a user authenticates with our application, using a combination of Oauth2 and ASP.Net Membership Provider, the user will be given a unique ASP.Net Membership Authentication token which is unique to that user’s authenticated session. Our application will then take that user’s unique token and compute a hash by combining the token with a unique salt. This computed hash is generated using a SHA256 hash algorithm. This hash will be used to create a cookie that is sent to the client. AngularJS will then pick up this cookie and inject it into the HTTP Header of all XHR requests from the application.

On the server side, we have implemented the necessary logic to check all requests to our REST APIs. Once a request is made to one of our HTTP REST APIs, the HTTP header of the request is checked for a specific token. If the token exists, the token is checked by comparing it with a computed hash of the current user’s ASP.Net authentication token. If the hash is equal we can guarantee that the request came from the authenticated user.

## Toolkits and Frameworks

### AngularJS

AngularJS (angularjs.org) is the primary client side framework we have employed in building our solution. AngularJS is the basis of our client side implementation. AngularJS is a framework for building Single Page Applications. It leverages the Model View Controller or Model View Whatever pattern to enable us to separate out the model from the presentation of the model. The pattern also separates out the logic used to control the user interaction with the system. The MVC pattern was invented by Trygve Reenskaug in a paper published, on December 10th, 1979. The MVC pattern describes how to separate parts of an application into the following constructs:

* Models: Models represent knowledge. Models are classes which are used to store state or data.
* Views: A View is a visual representation of its model. It is the HTML used required to render the model to the user.
* Controller: A controller is the link between a user and the system. The controller interprets the user’s input, what impact this input has on the model. The controller then determines the view that is presented to the user.

AngularJS provides controllers which allow us to encapsulate the business logic and behaviour of features with our application. The Model View Whatever pattern means that we can de-couple this logic from the presentation layer which is represented as a View. Using Angular, we were able to organise our solution components into distinct loosely coupled components. This loose coupling greatly enhances the testability of the application. In addition, AngularJS makes extensive of dependency injection techniques which means that we can test client side logic in the absence of dependencies such as web services. AngularJS has helped us structure and test our JavaScript code

AngularJS provides the following capabilities:

* MVC: AngularJS provides a Model View Whatever implementation of the Model View Controller design pattern. The Model View Whatever implementation in AngularJS is a combination of the Model View Controller (MVC) pattern and the Model-View-ViewModel (MVVM) pattern. We have used this pattern to separate our Models from our Views and also separate out the logic used to control the interaction between the application and the user.
* Dependency Injection: AngularJS has an in-built dependency injection subsystem that enables us to implement logic that is easy to maintain and test. Dependency Injection allows us to inject dependencies as required into our logic. This means we can test the logic without the dependencies becoming an issue.
* Two-way Data-Binding: This automatically updates the View when the underlying Model changes and automatically updates the Model when the View changes.
* Templates: Templates are HTML combined with rendering instructions known as directives.
* Directives: Directives allow us to define our own custom HTML syntax. This custom HTML syntax can be used as a placeholder for extended functionality implemented in JavaScript. AngularJS will then inject this functionality when rendering the HTML to the client. We have also created a custom Google Authentication AngularJS Directive. Directives are a very powerful feature of Angular. Directives allow us to extend the HTML to meet our requirements using a clean and maintainable implementation.
* Modules: AngularJS Modules are used as containers to group AngularJS components together. AngularJS does not have a main method. Modules declaratively specify how an AngularJS Single Page Application should be bootstrapped.
* Routing: With the AngularJS routing provider we can configure how AngularJS tries to map URLs to Controllers and Views. We have used the AngularJS route provider to configure the routing within our application. Based on the URL a user accesses, the AngularJS routing provider will direct the application to a particular view and controller combination.

Reasons for choosing AngularJS:

* Extensibility: This allows us to extend and/or enhance the capabilities of AngularJS.
* 2-way data binding: AngularJS automatically updates the view whenever the model changes – as well as visa-versa.
* Controllers: AngularJS allows us to write logic for the behaviour of the application without having to worry about updating DOM elements.
* Server Communication: AngularJS provides in-built support for XHR. It also handles asynchronous server communication with simple constructs.
* Bootstrapping: This can automatically initialise an AngularJS application. We have employed AngularJS automatic bootstrapping to compile the DOM using the ng-app directive.
* Dependency Injection: AngularJS allows us to declaratively describe our application logic. This means we declare components as and when required and replace components as necessary.
* Directives: Allows us to create custom HTML syntax with custom functionality behind it.
* Testability: AngularJS was designed to be testable. This means we can use AngularJS constructs to implement truly testable code.

### JQuery

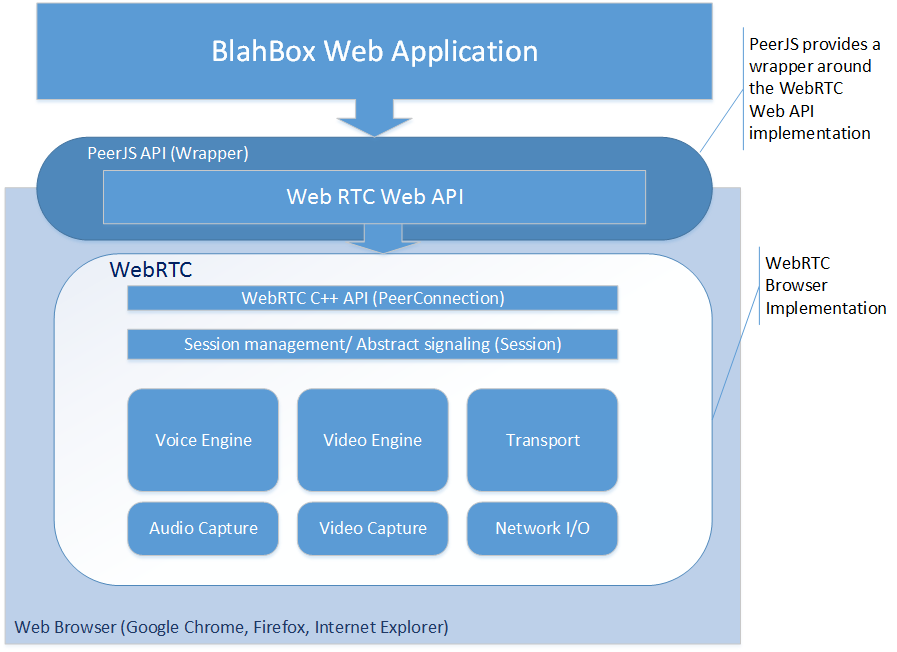
This is a client side library that is primarily used to manipulate the DOM. We do not use JQuery for DOM manipulation as we have used AngularJS to manage rendering HTML to the browser. JQuery has been included in this project as it is required by the SignalR client side library.

### PeerJS

PeerJS handles the complexities of working with the WebRTC API including handshaking, binary string encoding and server brokering of connections. PeerJS.com also provides a free server to broker connections between peers.

The browser WebRTC implementation handles the actual networking complexities including NAT traversal, UDP and it also maintains the peer-to-peer connection tunnel. WebRTC was drafted by the W3C (World Wide Web Consortium) to provide plugin-less VoIP (Voice over Internet Protocol) and P2P (Peer to Peer) communication in web browsers. WebRTC provides the following features:

* True peer-to-peer
* Handles NAT traversals
* UDP
* Configurable data reliability
* Supports raw binary data



Features of PeerJS:

* Handles the WebRTC handshaking.
* Provides the ability to connect to peers by id.
* Optimises the connection brokering latency using a combination of hybrid XHR and WS.
* Provides binary data support with BinaryPack serialization
* Provides fast connection establishment by pre-connecting to clients before actual connection is initiated.
* PeerJS has server-side service that provides authentication and initiates connections between peers.

### Bootstrap

Bootstrap (getbootstrap.com) has been employed to provide a responsive UI design to enable rendering of the application on numerous devices. Bootstrap is an open-source front-end UI framework for building fluid web page layouts. Bootstrap provides is a set of CSS and JavaScript files that have enabled us to implement a responsive UI design layout based on a grid system.

### Jasmine

We have used Jasmine (jasmine.github.io) to implement our JavaScript Unit Tests. Jasmine is a Behaviour Driven Development (BDD) testing framework for JavaScript. Jasmine allows us to write tests that are independent of the DOM (Document Object Model) and tests that are independent of other JavaScript frameworks. Jasmine includes the following features:

* Behaviour Driven Development (BDD) syntax for creating test specifications.
* Supports asynchronous testing.
* Support for Mocking and Spying.
* Jasmine allows us to write our own custom matchers.
* Jasmine allows us to share or isolate behaviours between tests within a spec.
* Jasmine provides support for continuous integration.

### JQuery SignalR Library

The application uses SignalR to notify the client that another user is online. SignalR is Microsoft’s web sockets implementation for the .Net Framework. The SignalR client side components require JQuery.

## Data Transfer Strategies

All of our data transfer from the client side to the server side is handled by AngularJS. On the server side we have implemented REST APIs which transfer data via JSON. On the client side, our application makes asynchronous XHR requests using AngularJS. AngularJS provides us with the $http service, this service facilitates server communication and data transfer. The AngularJS $http service provides a promise/deferred implementation for making XHR requests and retrieving data from those requests.

A promise is an interface for interacting with an object that represents the results of an action that is performed asynchronously. A promise is a mechanism for handling the results of an operation that may or may not be finished at any given point in time. One way of looking at promises is that they are to asynchronous programming what ‘try, catch and throw’ are to synchronous programming.

## Evaluation and Testing

We have used the Jasmine Behaviour Driven Development (BDD) framework for testing all of our JavaScript code. Behaviour Driven Development (BDD) is an evolution of the concept of Test Driven Development (TDD). With Behaviour Driven Development (BDD) test are designed to test features or behaviours of our application. BDD allows us to we write tests in a natural language style which can be read by non-technical individuals. Writing tests in this style also means that the tests provide excellent documentation of our code.

The main reasons for using Jasmine are:

* There are no dependencies on any other JavaScript frameworks.
* Jasmine does not depend on the DOM (Document Object Model).
* Jasmine uses a clean and understandable syntax.
* Jasmine borrows from the best parts of ScrewUnit, JSSpec, JSpec and RSpec.
* Jasmine allows us to test our AngularJS bits.

Jasmine was also used because it provides support for Stubs, Spies, Fakes and Mocks. Stubs are objects that simulate other objects, they provide the same interface but are configured to return pre-canned results. Spies allow us to write tests that check what was functions were called and what parameters were used to call these functions. A fake is used to simplify a dependency so that a test can pass easily. For example, a fake can be used to provide a simplified model instead of one from a database. Mocks are like stubs but they also provide behaviour verification, they are used to verify that the correct behaviours occurred during an interaction.

To test AngularJS XHR requests we have used the $httpBackend service. This service provides a fake HTTP backend implementation for the $http service. The $http service handles our server communication, we use it to perform all our AngularJS XHR requests. $httpBackend allows us to test our JavaScript code in the absence of the server side REST APIs. We have used the $httpBackend service because we want our tests to run without any external dependencies. The $httpBackend service allows us to perform tests without sending test data to our real REST services. We have used the $httpBackend service to respond with pre-configured responses so that we can test the client side JavaScript logic without requiring an external server side dependency.

#### Debugging Tools

##### We have used Visual Studio to perform debugging. Visual Studio allows us to insert break points into JavaScript code and step through it. We have also used Visual Studio to debug server side code.

We have used AngularJS Batarang Chrome Extension to debug and profile the AngularJS bits. AngularJS Batarang sits as an addition tab within Chrome Developer Tools. It includes the ability to view dependencies, view the data in the Models that are loaded in our application. The AngularJS Batarang tool also allows us to view performance metrics of the AngularJS bits of our application. The AngularJS Batarang also extends the Chrome Developer Tools Console by adding an area where we can look at the properties of AngularJS objects.

# Summary

The *BlahBox* development is complete and even though it's not completely ready for launching it live, it's a rather advanced working prototype. The application works well in all browsers supporting WebRTC therefore it's not supported by IE and native Android browsers, however this limitation will be gone in near future, in addition mobile version of Chrome can be used on mobiles, which implements WebRTC.

Before the start of the development the team gathered together to discuss the requirements of the application and frameworks to be used. Once the requirements were established the team members started the developing proofs of concept of the critical parts of the application, such as SignalR, PeerJS, AngularJS. Once the concepts were proven, the work was divided between team members based on their abilities.

The code base is tracked by Git; before everyone started committing, we had to agree how everyone is going to commit without constantly causing conflicts between different members of the team. The approach we took was to create a branch for each of the members or parts of the development and to work strictly off the branch. Once a particular section of the requirements is complete, the changes were pushed to GitHub and a pull request was created. We also had to agree to allow 2 people to merge pull requests, this is to ensure that the code is reviewed before pushing to master branch.

The issues that were arising were logged using Issue Tracker in GitHub and were dealt with by the team members that created them or in cases where it made sense to assign the issue to another member, it was done so.

The main tools for the development of BlahBox were Visual Studio and GitHub. Also a number of different libraries and frameworks were used on both, server and client side.

The application was deployed to a live server on AWS under the following URL http://blah-box.tk. The web server hosting the application is a free tier VPS running Windows Server 2012. The application's master branch is checked out on the server, therefore in case new changes are added to the application, the server administrator just has to pull the new changes.

The application core functionality is developed and tested, however it lacks usability testing which could point out to issues that would require application re-structuring. In addition to usability, the application lacks alert messages when a user adds another user to the friends list and although there are notifications appearing when there's an incoming call, the application should have sound effects for incoming calls and messages, as well as when the call is ended or a user was added to the list.

Currently the application only implements Google OAuth authentication, however ideally it should provide the option to login with other providers such as Facebook, Twitter or LinkedIn.

# Bibliography

Basu, S., 2013. *Real-World Techniques.* [Online]   
Available at: http://link.springer.com/chapter/10.1007/978-1-4302-5026-5\_15#page-1  
[Accessed 29 04 2014].

Bishop, T., 2011. *Flash, Silverlight and the end of the line for browser plug-ins.* [Online]   
Available at: http://www.geekwire.com/2011/adobe-flash-microsoft-silverlight-sunset-browser-plugin/  
[Accessed 29 04 2014].

Chouhan, H., 2013. *6 Reasons to Choose the Bootstrap CSS Framework.* [Online]   
Available at: http://www.ostraining.com/blog/coding/bootstrap/.   
[Accessed 29 04 2014].

Cohen, K., 2014. *Basics of WebRTC Peer Connections.* [Online]   
Available at: https://medium.com/programming-ideas-tutorial-and-experience/c1ed743de2f6  
[Accessed 29 04 2014].

Dennis, Z., 2013. *Choosing an SSO Strategy: SAML vs OAuth2.* [Online]   
Available at: http://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/  
[Accessed 29 04 2014].

Designer, D., 2013. *5 responsive frameworks for more productive web design.* [Online]   
Available at: http://www.designer-daily.com/5-responsive-frameworks-for-more-productive-web-design-34969  
[Accessed 29 04 2014].

D'Oliveira, R., 2011. *Pros And Cons Of Using Frameworks.* [Online]   
Available at: http://www.1stwebdesigner.com/design/pros-cons-frameworks/  
[Accessed 29 04 2014].

GitHub, n.d. *github.com.* [Online]   
Available at: https://github.com/search?q=stars%3A%3E1&s=stars&type=Repositories  
[Accessed 29 04 2014].

Govett, D., 2013. *PEERJS: A PEER TO PEER NETWORKING LIBRARY IN JAVASCRIPT USING WEBRTC.* [Online]   
Available at: http://badassjs.com/post/43090030238/peerjs-a-peer-to-peer-networking-library-in-javascript  
[Accessed 29 04 2014].

Gube, J., 2013. *10 Best Responsive HTML5 Frameworks and Tools.* [Online]   
Available at: http://designinstruct.com/roundups/html5-frameworks/  
[Accessed 29 04 2014].

Hanselman, S., 2011. *Asynchronous scalable web applications with real-time persistent long-running connections with SignalR.* [Online]   
Available at: http://www.hanselman.com/blog/AsynchronousScalableWebApplicationsWithRealtimePersistentLongrunningConnectionsWithSignalR.aspx  
[Accessed 29 04 2014].

Hickson, 2014. *WebRTC 1.0: Real-time Communication Between Browsers.* [Online]   
Available at: http://dev.w3.org/2011/webrtc/editor/webrtc.html  
[Accessed 29 04 2014].

Jasmine, n.d. [Online]   
Available at: http://jasmine.github.io/2.0/introduction.html  
[Accessed 29 04 2014].

Klingenstein, N., 2012. *OASIS Security Services (SAML) TC.* [Online]   
Available at: https://www.oasis-open.org/committees/tc\_home.php?wg\_abbrev=security  
[Accessed 29 04 2014].

Knight, K., 2011. *Responsive Web Design: What It Is and How To Use It.* [Online]   
Available at: http://www.smashingmagazine.com/2011/01/12/guidelines-for-responsive-web-design/  
[Accessed 29 04 2014].

Koussa, S., 2013. *Federated Identities: OpenID vs SAML vs OAuth.* [Online]   
Available at: http://www.softwaresecured.com/2013/07/16/federated-identities-openid-vs-saml-vs-oauth/#  
[Accessed 29 04 2014].

Lero, C. S., 2011. *A Study of the Characteristics of Behaviour Driven.* [Online]   
Available at: http://ulir.ul.ie/bitstream/handle/10344/1256/Solis\_2011\_behaviour.pdf  
[Accessed 29 04 2014].

Mardanov, A., n.d. *Rapid Prototyping with JS.* [Online]   
Available at: http://books.google.ie/books?hl=en&lr=&id=afPPAgAAQBAJ&oi=fnd&pg=PP5&dq=test+driven+development+javascript&ots=qCtdUE\_kwF&sig=GUksjY\_8Hepkk86vjtMznPUSdmA&redir\_esc=y#v=onepage&q=test%20driven%20development%20javascript&f=false  
[Accessed 29 04 2014].

Michels, D., 2013. *Is Skype a WebRTC Killer?.* [Online]   
Available at: http://www.nojitter.com/post/240158376/is-skype-a-webrtc-killer  
[Accessed 29 04 2014].

North, D., 2006. *Introducing BDD | Dan North & Associates.* [Online]   
Available at: http://dannorth.net/introducing-bdd/  
[Accessed 29 04 2014].

OAuth.NET, n.d. *OAuth Community Site.* [Online]   
Available at: http://oauth.net/  
[Accessed 29 04 2014].

OpenID.NET, n.d. *OpenID Connect FAQ and Q&As.* [Online]   
Available at: http://openid.net/connect/faq/  
[Accessed 29 04 2014].

PeerJS, n.d. *PeerJS.* [Online]   
Available at: http://peerjs.com/  
[Accessed 29 04 2014].

Petereit, D., 2013. *Peer.js Enhances WebRTC, Allows For Peer-To-Peer Connections In The Browser.* [Online]   
Available at: http://www.noupe.com/javascript/peer-js-enhances-webrtc-allows-for-peer-to-peer-connections-in-the-browser-75120.html  
[Accessed 29 04 2014].

Pretorius, J., 2013. *AngularJS Pain Points.* [Online]   
Available at: http://www.jacopretorius.net/2013/07/angularjs-pain-points.html  
[Accessed 29 04 2014].

QUnit, n.d. *QUnit.* [Online]   
Available at: https://qunitjs.com/  
[Accessed 29 04 2014].

Skeleton, n.d. *A Beautiful Boilerplate for Responsive, Mobile-Friendly Development.* [Online]   
Available at: http://www.getskeleton.com/#whatAndWhy  
[Accessed 29 04 2014].

Smus, B., 2012. *BACKBONE AND EMBER.* [Online]   
Available at: http://smus.com/backbone-and-ember/  
[Accessed 30 04 2014].

Somorovsky, J., 2011. [Online]   
Available at: https://www.owasp.org/images/5/5a/07A\_Breaking\_XML\_Signature\_and\_Encryption\_-\_Juraj\_Somorovsky.pdf  
[Accessed 29 04 2014].

Stovall, G., 2013. *AngularJS: An Overview.* [Online]   
Available at: http://glennstovall.com/blog/2013/06/27/angularjs-an-overview/  
[Accessed 29 04 2014].

webrtc.org, n.d. *Web RTC.* [Online]   
Available at: http://www.webrtc.org/  
[Accessed 29 04 2014].

Yuzhuo, S., 2013. *Design and realization of chatting tool based on web.* [Online]   
Available at: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=6703312  
[Accessed 29 04 2014].

Zaefferer, J., 2012. *Introduction To JavaScript Unit Testing.* [Online]   
Available at: http://www.smashingmagazine.com/2012/06/27/introduction-to-javascript-unit-testing-2/  
[Accessed 29 04 2014].

Zaparka, P., 2011. *Which JS Unit Testing Framework?.* [Online]   
Available at: http://www.unboxedconsulting.com/blog/which-js-unit-testing-framework  
[Accessed 29 04 2014].