PROJECT 2: Exploring & Developing cutting edge Web Tech!

JUNE, 2019

WEB701

Dean Stanley-Hunt

2019

Contents

[1. Introducing The Technology 2](#_Toc12013254)

[1.1 EasyRTC – Easy Real-Time Communication 2](#_Toc12013255)

[1.2 Node.JS – Runs EasyRTC 3](#_Toc12013256)

[1.3 WebRTC – Built Into EasyRTC 3](#_Toc12013257)

[2. Implementing The Technology 4](#_Toc12013258)

[2.1 Get EasyRTC Server Running On Windows 5](#_Toc12013259)

[2.2 Set Up Xampp & Host Web Files 7](#_Toc12013260)

[2.3 Coding the Live Chat Support 8](#_Toc12013261)

[2.4 Testing Live Chat Support 12](#_Toc12013262)

[3. Problems With The Technology 14](#_Toc12013263)

[3.1 EasyRTC 14](#_Toc12013264)

[3.2 Node.js 14](#_Toc12013265)

[3.3 WebRTC 15](#_Toc12013266)

[4. Impact Of The Technology 16](#_Toc12013267)

[5. References 17](#_Toc12013268)

# 1. Introducing The Technology

The technology chosen to investigate, and implement is EasyRTC where RTC stands for Real-Time Communication, this technology takes an advanced technology called WebRTC made by Google and wraps it all up into an easy to use web development framework, that otherwise would make using WebRTC technology a large and complicated task to tackle even for experienced developers.

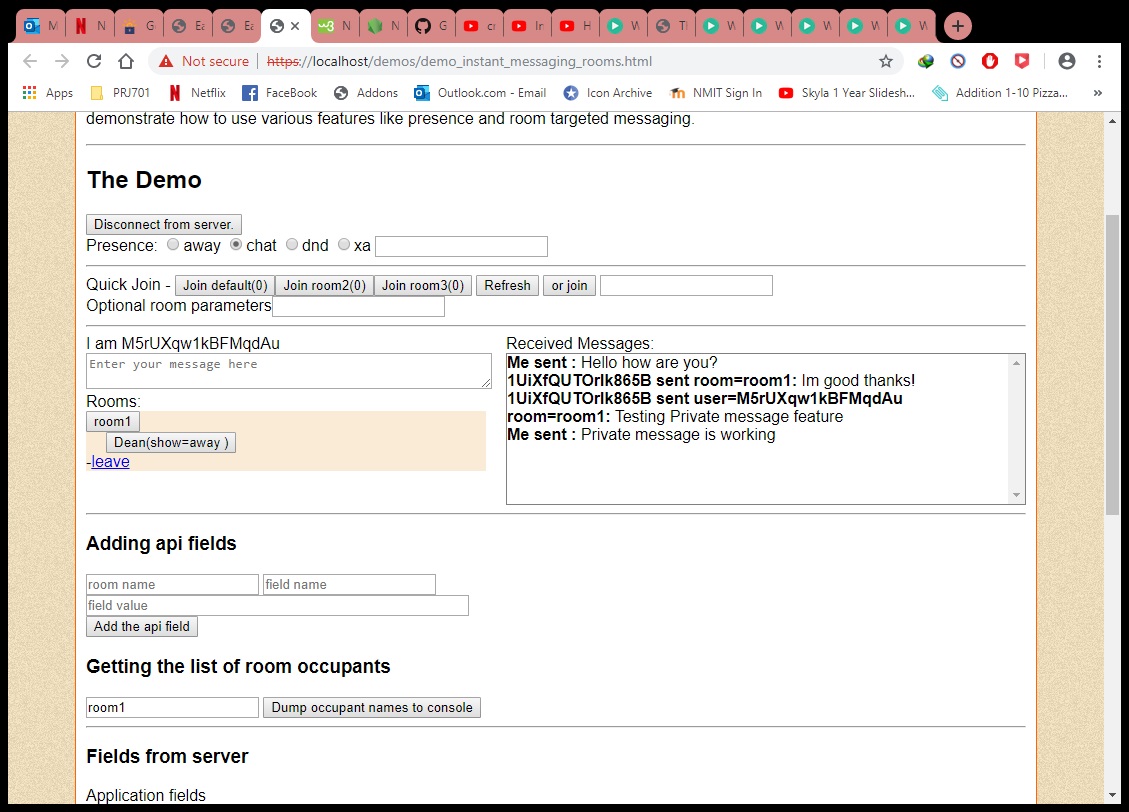
While this technology supports live audio and video streaming, for this project only the data channel feature will be implemented to create a live real-time chat support solution that can make websites capable of supporting live chat with visitors and/or potential clients, consumers, etc.

## 1.1 EasyRTC – Easy Real-Time Communication

Official Website: <https://www.easyrtc.com/>

GitHub Repo: <https://github.com/priologic/easyrtc>

“EasyRTC is a full-stack open source WebRTC toolkit suitable for building highly secure, WebRTC applications. It is a bundle of web applications, code snippets, client libraries and server components meticulously written and documented to work right out of the box.” (EasyRTC, 2019)



This image shows one of the examples that come with EasyRTC which demonstrates how to create default rooms and user created rooms for live chatting, how to display the users inside a given room and how to send private messages to any single user in the room. Using this example should provide everything needed to develop the live chat service for this project.

EasyRTC has been built using Node.JS as a run-time environment to host a web server which handles the signaling between peers and serves up the front-end web pages. It is essentially a WebRTC supporting Framework that developers can use to build web applications that take advantage of all the features of WebRTC in a matter of hours or days compared to weeks if building with WebRTC from scratch yourself. It can be used to integrate live text, audio or video chat to any website which could be as a simple live chat service for customers to contact the company instantly or for a full-scale chat network with chatrooms, private messaging and most importantly real-time audio and video chat.

## 1.2 Node.JS – Runs EasyRTC

Official Website (Required to run EasyRTC): <https://nodejs.org/en/>

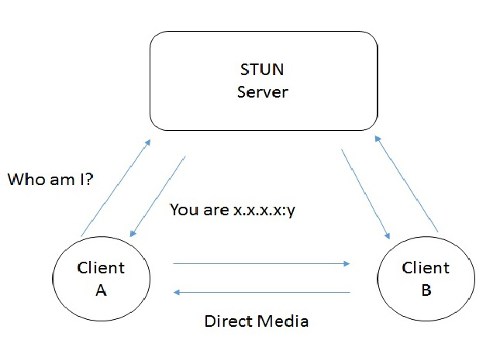
“Node.js is an open source server environment, which allows you to run JavaScript on the server and has a set of built-in modules.” (w3schools, 2019)

Node.JS installed on windows will be the server environment used to develop this project while EasyRTC will provide the framework to use where real-time communications implementation will be made easy with the built-in WebRTC features it provides.

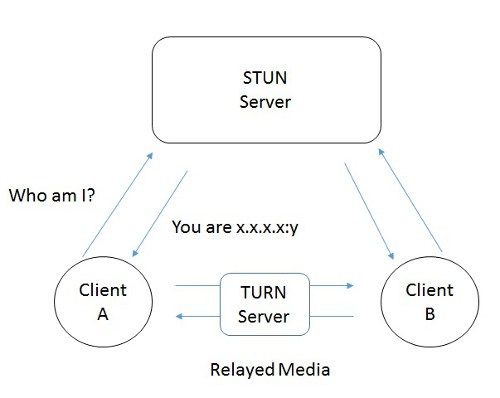
## 1.3 WebRTC – Built Into EasyRTC

Official Website: <https://webrtc.org/>

According to the Google Chrome Team (2019) at the official website, WebRTC is an open source project that is free to use for integrating Real-Time Communications (RTC) into web, mobile or desktop applications using a single set of APIs to support real-time events like text-based chat or file transfers, but most notably its successful cross-platform voice chat, video streaming or both.



The image above depicts either peer wanting to connect getting their respective connection details from the STUN server, then directly being able to connect and share Real-Time Communication.



This image above is similar to the previous one, but shows how if direct connections are not possible that a TURN server is introduced to make everything possible no matter what, it is a failsafe really and definitely not preferred because connections that run through a TURN server are constant and all traffic flows through, using server resources and bandwidth, compared to direct connections between peers which use no server resources after the initial handshake.

It is important to know that there is also a Signaling server involved which is used to initially let request a connection to another peer, and if accepted this signal server is then used to pass the connection information between the users, allowing them to then either directly connect or connect through TURN depending on the configuration passed. In EasyRTC it uses Node.js to handle this signaling and uses free servers provided by Google to support STUN and TURN services.

# 2. Implementing The Technology

Here we will go through the steps of implementing an EasyRTC server locally on EasyRTC’s default port of 8443 for SSL support, once it is installed and running in Node.js we don’t need to touch or modify it.

Then instead of using the built-in Web Server of EasyRTC for serving up Web files, we will host them elsewhere using Xampp on the default Web ports 80 & 443, which then simulates implementation for any existing Websites to store the Web files for our Live Chat Support on their current hosting.

Lastly, to make it work we should have the client side socket.io and EasyRTC API stored and pointed to locally with our Web files and then modify connections for EasyRTC inside the JavaScript Web files to point at our local EasyRTC server instance running on port 8443.

## 2.1 Get EasyRTC Server Running On Windows

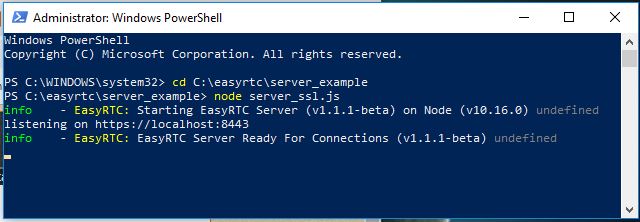
GitHub Repo: <https://github.com/priologic/easyrtc>

Node.js is required and has been downloaded and installed prior to beginning this implementation, so simply Download EasyRTC files from the GitHub repository as a zip file, extract all files and rename the folder to “easyrtc”, then cut & pasted to “C:\”.

Now opening Command Prompt or Windows PowerShell as Administrator do the following:

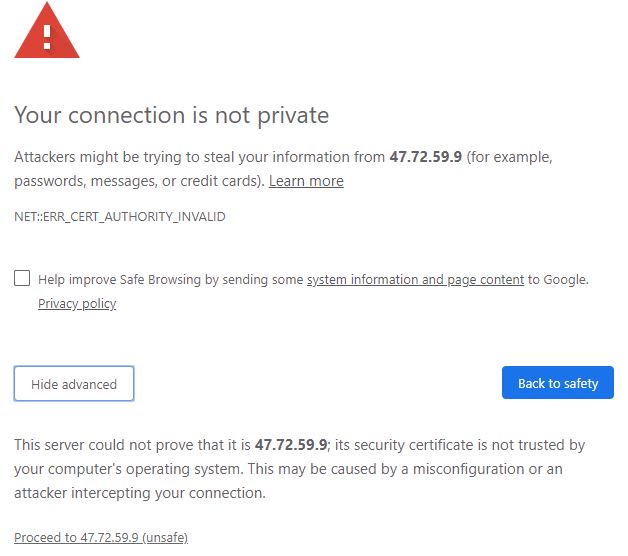
1. Enter the easyrtc directory by executing “cd C:\easyrtc”.
2. Run “npm install” and wait for it to finish installing.
3. Enter the easyrtc\server\_example directory by executing “cd server\_example”.
4. Run “npm install” and wait for it to finish installing.

Now EasyRTC is installed and ready to run anytime you want, to run EasyRTC you need to be in the easyrtc\server\_example directory by executing “cd C:\easyrtc\server\_example” like the image seen below, followed by “node server\_ssl.js” to start the EasyRTC server using SSL with Node.js, minimize and leave it running, push CTRL+C in the terminal to stop it when desired, EasyRTC also use the terminal to shows errors when they occur server-side.



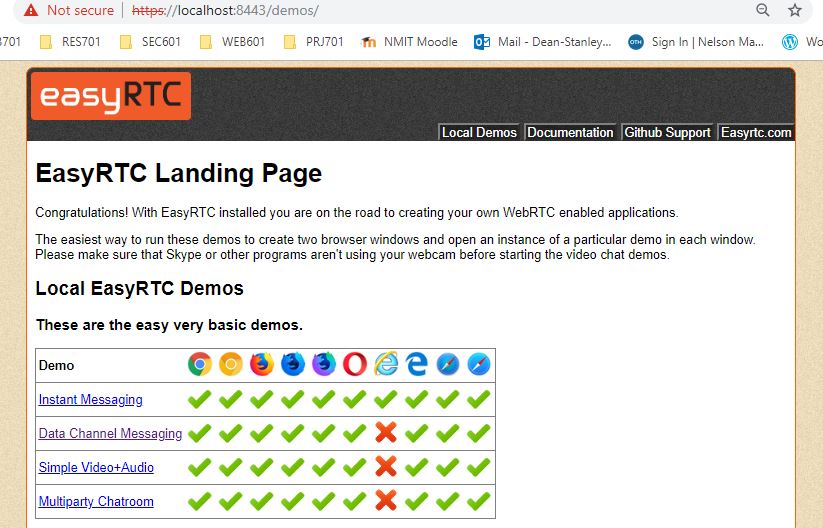
While the Live Chat Support is only using the Data Channel feature of WebRTC inside EasyRTC, it is IMPORTANT TO NOTE:

Chrome will not grant access to your local microphones or cameras for a page served using HTTP port 80 except for the localhost case, hence I am showing the use of “server\_ssl.js” and not “server.js” to support SSL secure encryption and access to these elements for the EasyRTC audio/video demo’s. But it actually is not required for Data Channeling to work, however secure encryption is desired for all connections and traffic when communicating with the EasyRTC server.

By default, EasyRTC uses a self-signed SSL certificate to serve up SSL, but normally you would get a real certificate and a domain name for pointing traffic to your EasyRTC server.

In this case, if at any point you see the warning about the SSL certificate not being trusted like in the image below seen, you can simply click the advanced button then on the bottom part which shows click Proceed.

Now you should be able to test the EasyRTC server in your browser and view or play with some of the demo’s provided as examples of various possibilities for the application of EasyRTC. These are located at “C:\easyrtc\demos\” where the EasyRTC folder itself acts like a public HTML directory, so to navigate to these demos in our browser we should now put <https://localhost:8443/demos/>.

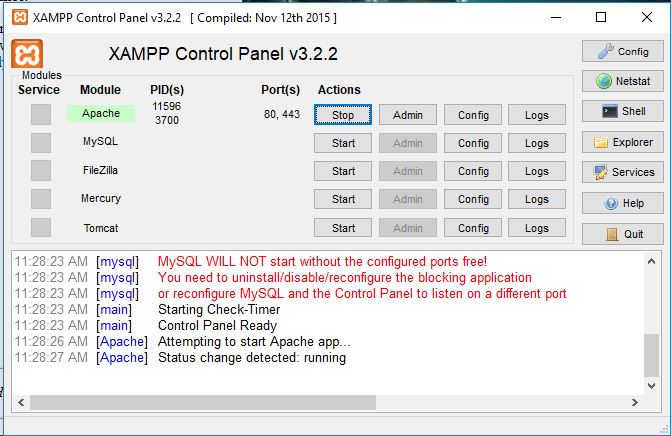
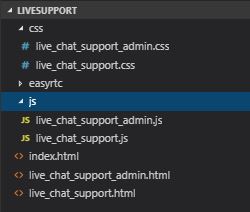


If desired you can use EasyRTC to host a Website, modify demo’s, play and learn EasyRTC. But for this implementation of Live Chat Support, we will host Web files else-where using Xampp so let’s move on.

## 2.2 Set Up Xampp & Host Web Files

Xampp Download: <https://www.apachefriends.org/download.html>

After downloading and installing Xampp for Windows, launch Xampp and you come to an interface like in the image below to the left, for this implementation you only require the Apache module for hosting a local Web server that will serve up the Web files for the Live Chat Support system. Simply click Start!

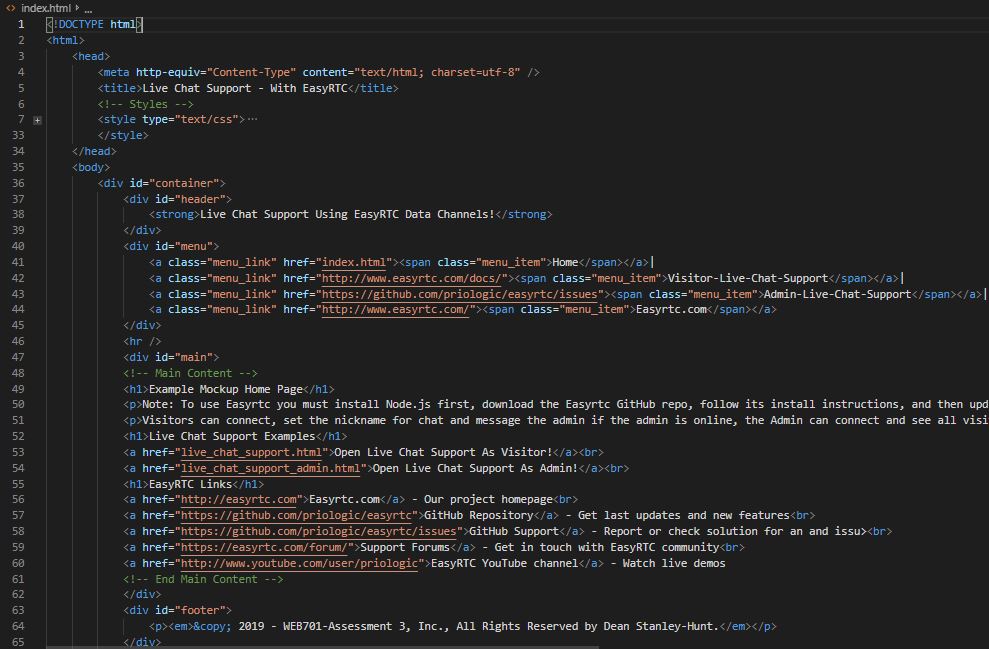
Next click the Explorer button on Xampp, it opens the folder location of Xampp. There open the “htdocs” folder which is where Xampp’s public HTML Web files are stored, for this implementation a folder called “LiveSupport” was created inside htdocs to place all the Web Files from the Live Chat Support system. A copy of the ”LiveSupport” folder and all these Web files seen in the image above to the right should be found along-side this report compressed in a zip file.

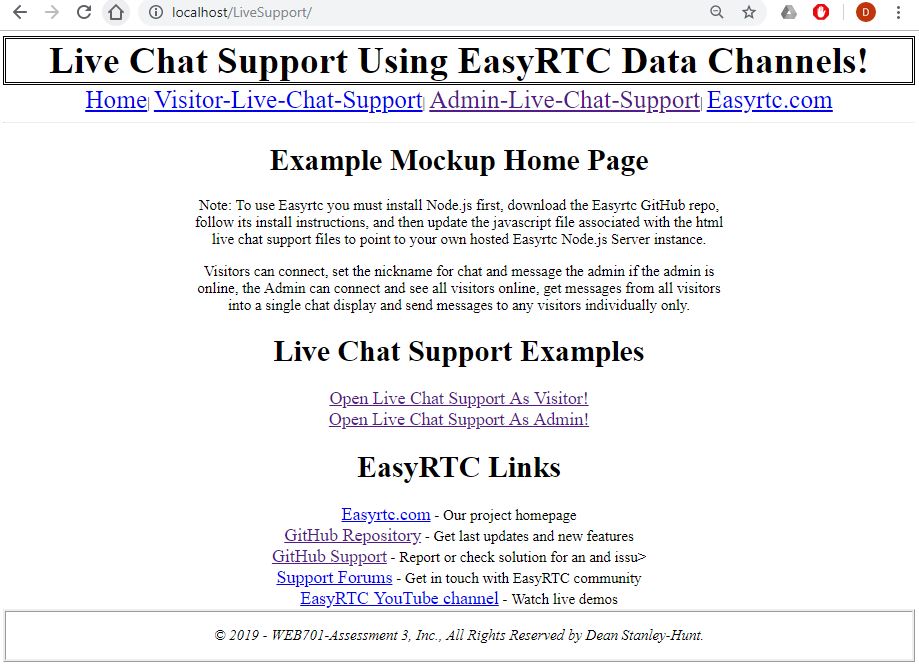
The file structure seen in the image above to the right consist of a very simple home page in the index HTML, for the Live Chat Support there is 2 HTML view files, 1 for use by a single administrator at a time only to communicate to the visitors of the Website and should never be provided to the public. The other HTML view is shown to visitors and allows communication from and to the administrator, it also indicates if there is currently an administrator online now or not.

There is 3 folders, js, css, and easyrtc. Inside the JavaScript (JS) folder and the CSS folder you can see 1 JS and CSS file named identically to correspond with their respective HTML view files, so basically for each HTML Live Chat Support view there is 1 JS and CSS associated. A CSS file styles the view while the JS file controls the behaviour of the view, and connection, events or interaction with an EasyRTC server.

The easyrtc folder should not be touched or modified, it is the home for all client-side EasyRTC API and the module for socket.io which is referenced in the head of HTML views and used by the JS file linked. These are required when deciding not to host your Website files outside of the EasyRTC server.

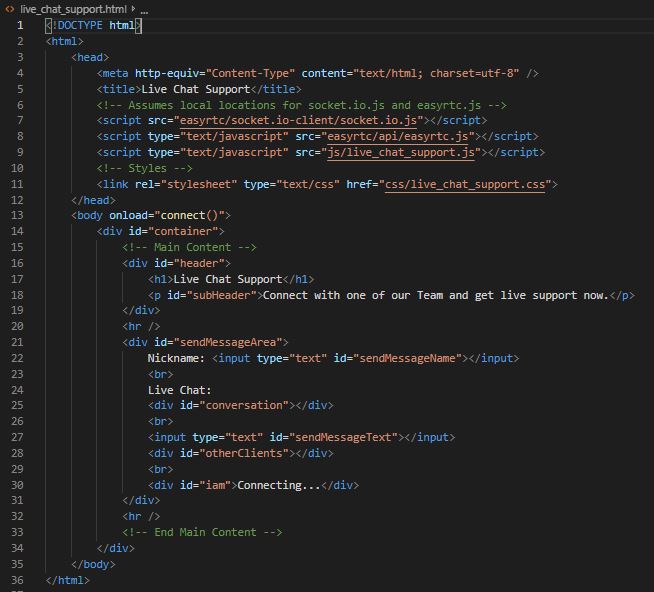
## 2.3 Coding the Live Chat Support

This image shows the code for the **Home page view** of this mock-up, it is very basic and nothing fancy just providing a bit of information and pointing to some EasyRTC resources while providing links to open the 2 Live Chat Support views, 1 for visitors and the other for administration demonstration.

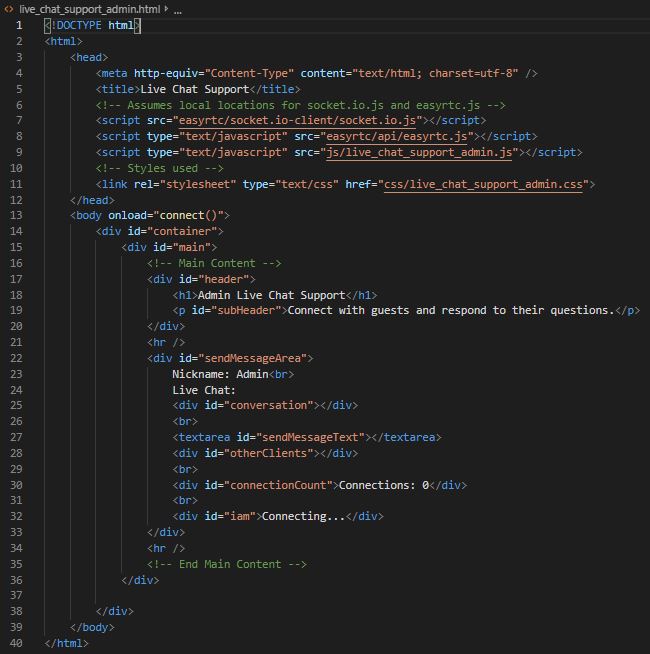
The result looks like this when visiting [localhost/LiveSupport/](http://localhost/LiveSupport/)

Home Page Mock Up:

The next image shows the code for the **Visitors Live Chat Support** HTML view, which you can clearly see has been set up to load the EasyRTC API and socket.io module locally with Web files and then imports the JS and CSS files associated with this view. When the Body starts to load it calls the connect function from the JS file, trying to establish a connection before even loading the body content. Here visitors can see if an administrator is online to ask help from, can see and send message to and from the administrator but not from other visitors, and must input the desired nickname for sending chat with.



Next up is code for the **Admin Live Chat Support HTML** view which can be used by an administrator of the Website to see all online visitors, it auto connects with them for communications, allows a single administrator at any 1 time to be able to send messages to visitors individually or send a message to all visitors connected. Being a simple mock-up it shows all chat from the multiple visitors in 1 conversation window but makes it clear who messages are from and which messages are sent to whom. Like the visitors view it also has been set up to load the EasyRTC API and socket.io module locally with Web files and then imports the JS and CSS files associated with this view.



NOTE: This Live Chat Support system has not been designed for more than 1 administrator to access this view at any 1 time, it will show the 2 admin’s online and visitors can message either individually but message from an admin simply say they came from admin so it would be impossible to know which admin responded and potentially cause confusion.

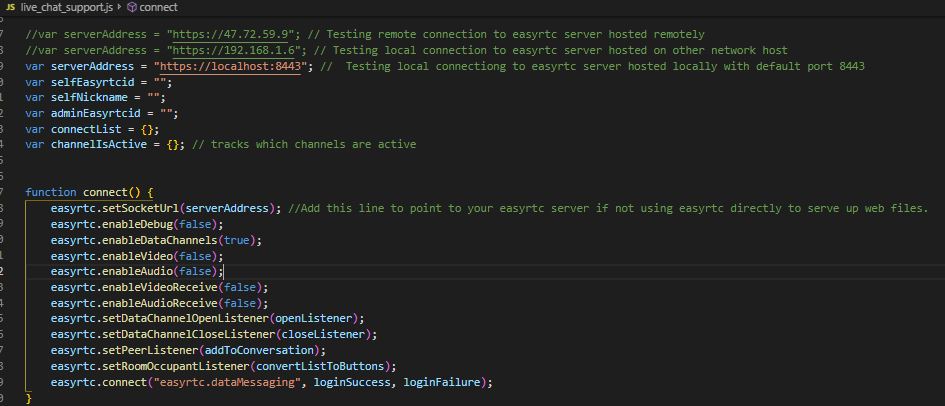
While the HTML files are still similar enough to the original data channel messaging EasyRTC demo this implementation originates from, they have been modified for their hosting, file importing, CSS exported externally and had extra fields added. The CSS files have all the style coding implemented to get everything positioned and sized to make the Live Chat Support usable and somewhat presentable in terms of layout.

The JS files have all the mechanics to make everything tick and turn over, they are the engine that drives this real-time Live Chat Support system and have been extensively modified from the original messaging demo used which simply showed all users connected to the EasyRTC server and let you individually connect to any 1 or more to then be able to send a message to any 1 user at a time, and was all running from a single view.

But now there are two JS files both behaving very differently and supporting the two different views for making this Live Chat Support system work as it now does.

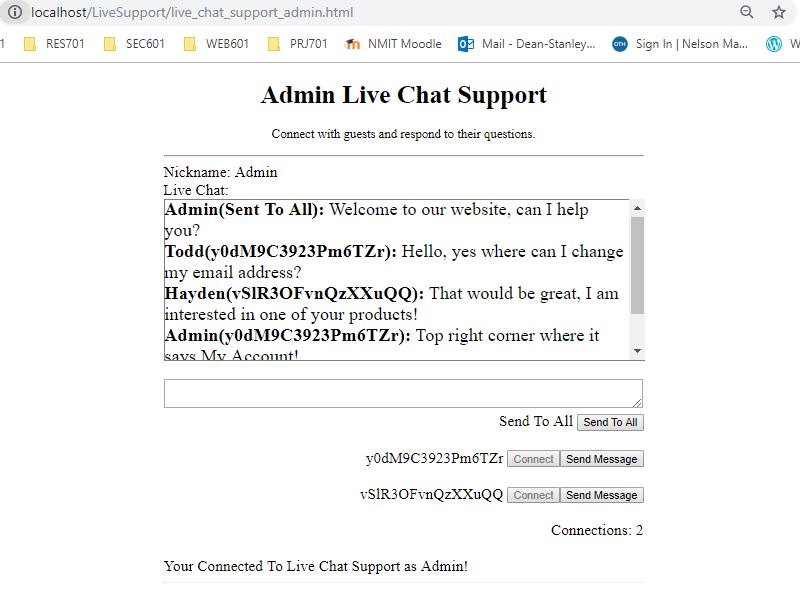
**IMPORTANT JS FILE NOTE:**

It is very important to know and understand the modification made to the connect function located in the JS files, and the variable added called “serverAddress”. By default EasyRTC demo’s don’t have the first line seen in the below image for our connect function which is “easyrtc.setSocketUrl”, this is very important to add when hosting the Web files outside the EasyRTC server, it points our connections to the EasyRTC signaling we set up and hosted locally on port 8443. It can also be set to point to a different local machine on your network or a remote machine, hosting an EasyRTC server.

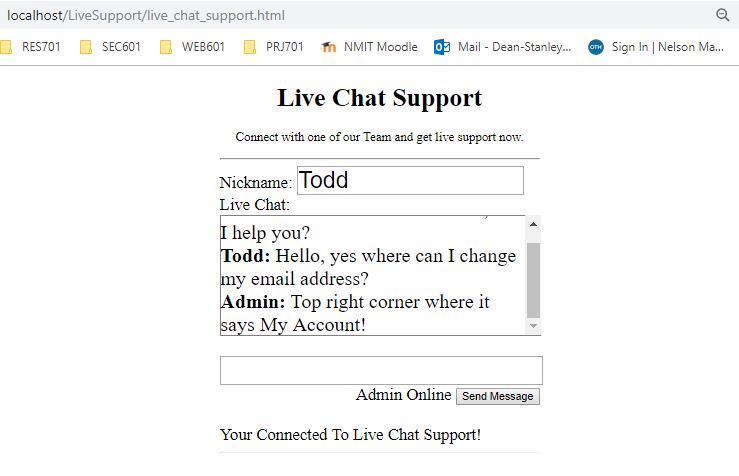


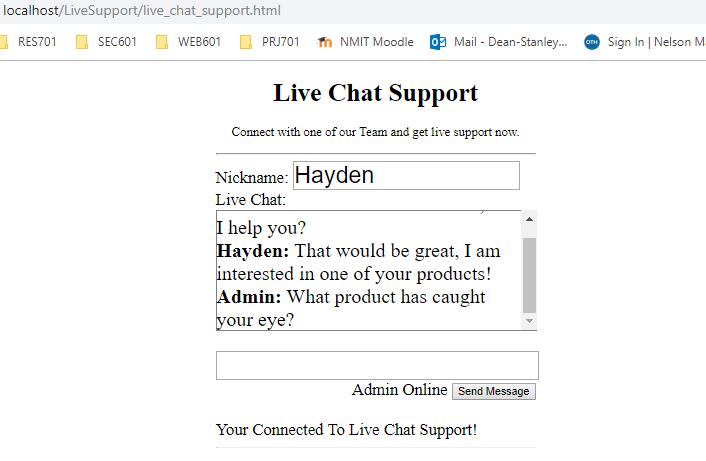
## 2.4 Testing Live Chat Support

The following 3 images show the browser address for the accessing the HTML views, and show that the Live Chat Support system is fully working, the first you can see the administrator view, with 2 visitors connected as seen in the next 2 images. We can see the number of connections, and buttons for sending a message to individuals or to all the visitors at once. Chat shows messages sent as Admin to all or to an individual, and response from visitors where the strange looking code in brackets is the identification for the user inside the EasyRTC server and is used to know who you will send a message to individually as an administrator.



For visitors, messages to and from the admin can be seen, while messages from other visitors cannot be seen or sent providing total Live Chat Support privacy between you and your visitors. See the following 2 images on the next page which show chat from the aspect of those 2 visitors seen in the administrator image above. Todd and Hayden, need help with different things and are communicating with the administrator. To wrap up implementation, EasyRTC is working and seems to suit the name, it was not to difficult to work with or code for and is easy to implement which has been done here successfully without issues. Also, these views can easily be embedded on any Web pages.





# 3. Problems With The Technology

## 3.1 EasyRTC

**Scalability:**

“The default behavior of EasyRTC server can cause scaling issues. But good news is you can tweak server-side events to handle huge traffic.” (Khattar, 2019)

To do signaling, EasyRTC uses the socket.io module on node.js. Khattar (2019) done load testing of the EasyRTC server implementation and discovered interesting results which any developer who will build with EasyRTC should read up on and become familiar with, and try to avoid the same problems.

Full Article: <https://www.hackerearth.com/practice/notes/lalitkhattar/load-testing-easyrtc-1/>

Current & Past Issues:

The (Priologic, 2019) repository has a place for developers using EasyRTC to post a thread about any issues they encounter with EasyRTC where the creators and contributors can assist with resolving issues regardless of if they are EasyRTC related, Node.js issues or the most common WebRTC issues the team there are happy to help, and make updates to EasyRTC to resolve issues as needed which leads to regular updates and improvements, there has been to many issues to start listing them here, but many have been resolved, fixed or had solutions mentioned where it has been a mistake from the developer.

I highly recommend looking at the latest issues to get an idea of what they are, any solutions and take note of the dates for when the issues where raised.

EasyRTC Issues: <https://github.com/priologic/easyrtc/issues>

## 3.2 Node.js

**Server-Side Web Application with a Relational Database:**

“Using Node.js with a relational database is still quite a pain. Do yourself a favour and pick up any other environment like Rails, Django, or ASP.Net MVC if you’re trying to perform relational operations.” (Capan, 2017)

Relational DB tools for Node.js still require further development.

**Heavy Server-Side Computation/Processing:**

“Any CPU intensive computation will block Node.js responsiveness, so a threaded platform is a better approach. Alternatively, you could try scaling out the computation” (Capan, 2017)

Node.js is single-threaded using just one CPU core to run unless distributed across multiple servers.

**Other Node.js Problems according to (Chamberlain, 2018) are:**

* Common Node.js deployment problems
* Uncaught exceptions or error events in JavaScript code
* Excessive memory usage, which may result in an out-of-memory error
* Unresponsive application, possibly looping or hanging
* Poor performance
* Crash or abort in native code

Full article with solutions: <https://developer.ibm.com/articles/6-reasons-your-node-js-apps-are-failing/>

## 3.3 WebRTC

According to (Levent-Levi, 2018), there are 5 common issues that occur with WebRTC to avoid:

* Failing to Configure STUN/TURN
* Selecting the WRONG Signaling Framework
* Not Using Media Servers When You Should
* Thinking Short-Term
* Failing to Understand WebRTC

“Code older than a year is stale or dead already. WebRTC is still too new and too dynamic. That said, it isn’t as if you have a choice anymore. Flash is dying, and there’s no other serious alternative to WebRTC.” (Levent-Levi, 2018)

Full article & solutions: <https://bloggeek.me/mistakes-developing-webrtc-applications/>

The EasyRTC website also has a page up which outlines clearly the problems and solutions with WebRTC which any serious developer using EasyRTC should read over and become familiar with.

“Many of these issues are general WebRTC or browser issues and not specific to EasyRTC… these will become less frequent as… browsers are updated.” (EasyRTC - WebRTC Problems, 2019)

* Permission Denied Error when creating a Local Media Stream
* No Self Video
* No Video sent to Peer
* Lag and slow connections
* Delays in Video or Audio
* Picture Quality
* Sound Quality

Full article & solutions: <https://easyrtc.com/docs/easyrtc_webrtc_problems.php>

# 4. Impact Of The Technology

There are three main developments which have powered EasyRTC's impact over the last few years, most of which relate to its main core technology, WebRTC:

* WebRTC evolving into a rich technology stack.
* Social apps rising in popularity and use, causing new consumer expectations.
* A wide array of disruptive apps that have emerged across various industries.

According to (Rajasekar, 2018), WebRTC is not just for browsers, it is a technology stack that has seen nearly a billion dollars of investment from the various companies involved, and in addition to being built-in to modern browsers, WebRTC is available for native mobile apps, desktop apps, and embedded Internet of Things (IoT) apps.

“WebRTC delivers benefits to users, integrators, and developers that have been less available in previous communications technologies. It’s open-source, for starters, so it’s free for all users and voice and video codecs are license-free. It also supports negotiation of media pipes and endpoints, offering efficient use of bandwidth and the best possible use of voice and video communication” (Ramsey, 2019)

Rejasekar (2018) points out that the availability of this rich open source technology stack has impacted a sea of change for enabling disruptive software development, and the essential benefit of WebRTC is enabling tightly integrated contextual communications whereby WebRTC is a building block within whatever disruptive workflows and innovative customer experience models developers can dream up.

Considering that most consumers are using messenger style applications to keep in touch from all over the world like Facebook, Snapchat and the good old Skype, social networks have brought real-time communications into the palms of well over 5 billion users revolutionizing the way we use to communicate via telephone lines or mobile calls and just dialing phone numbers. Each of these new applications supports one push audio and video calling technology that is using WebRTC directly or some derivative of the WebRTC stack to implement this feature, so it is clear to see all around us that WebRTC is impacting the way we keep in touch with friends, family, employees, and consumers.

“Within healthcare, … live video is essential to all kinds of use cases, including remote doctor/patient appointments, online therapy, group therapies, healthy lifestyle counseling, clinical reviews, and even remote experts in operating room theaters… Within education, for academic, commercial training and certification, there are again many hundreds of WebRTC-based apps delivering online classrooms, tutoring sessions, community learning groups, and more.” (Rajasekar, 2018)

So EasyRTC is really a WebRTC enabler that allows developers to start building web-based real-time communication, and can be used for just about anything you can imagine totally changing the way that communications are happening today and will undoubtedly impact the future of communication since it is the only technology that is providing cross-platform support, is totally free to use, has no licensing fees for the codecs, and providing a communication solution, via a common set of protocols.

# 5. References

Capan, T. (2017). *Why the hell would you use Node.js*. Retrieved from Medium.com: https://medium.com/the-node-js-collection/why-the-hell-would-you-use-node-js-4b053b94ab8e

Chamberlain, R. (2018). *6 reasons your node.js apps are failing*. Retrieved from IBM: https://developer.ibm.com/articles/6-reasons-your-node-js-apps-are-failing/

EasyRTC - WebRTC Problems. (2019). *WebRTC Problems*. Retrieved from EasyRTC: https://easyrtc.com/docs/easyrtc\_webrtc\_problems.php

EasyRTC. (2019). *EasyRTC*. Retrieved from EasyRTC: https://www.easyrtc.com/

Khattar, L. (2019). *Load testing EasyRTC*. Retrieved from HackerEarth: https://www.hackerearth.com/practice/notes/lalitkhattar/load-testing-easyrtc-1/

Levent-Levi, T. (2018). *Mistakes developing WebRTC applications*. Retrieved from BlogGeek: https://bloggeek.me/mistakes-developing-webrtc-applications/

Priologic. (2019). *Priologic - EasyRTC*. Retrieved from GItHub: https://github.com/priologic/easyrtc/issues

Rajasekar, B. (2018, March 07). *WebRTC Wave Now Unstoppable*. Retrieved from NoJitter.com: https://www.nojitter.com/webrtc-wave-now-unstoppable

Ramsey, R. (2019). *Exploring the impact of webrtc communications*. Retrieved from WebRTCworld.com: http://www.webrtcworld.com/topics/webrtc-world/articles/376527-exploring-impact-webrtc-communications-collaboration.htm

The Google Chrome Team. (2019). *WebRTC*. Retrieved from WebRTC: https://webrtc.org/

Thétiot, H. (2019, March). *Priologic - EasyRTC*. Retrieved from GitHub: https://github.com/priologic/easyrtc

w3schools. (2019). *Node.JS*. Retrieved from w3schools: https://www.w3schools.com/nodejs/default.asp