|  |
| --- |
| **CO3015 Computer Science Project** |
| Online Health Care Application |
| **Melvin Joseph** |

|  |
| --- |
|  |

Table of Contents

[Abstract 1](#_Toc7727239)

[Introduction 2](#_Toc7727240)

[Survey of Literature/Information Sources 2](#_Toc7727241)

[Node.js - Express 3](#_Toc7727242)

[Socket.io 3](#_Toc7727243)

[JavaScript 3](#_Toc7727244)

[Infermedica API 3](#_Toc7727245)

[WebRTC 3](#_Toc7727246)

[Requirements 4](#_Toc7727247)

[Functional 4](#_Toc7727248)

[Quality Attributes 4](#_Toc7727249)

[Non-Functional 5](#_Toc7727250)

[Specification, Planning and Algorithms 5](#_Toc7727251)

[Overall Design Specification 5](#_Toc7727252)

[Programming Language 5](#_Toc7727253)

[Web Framework 6](#_Toc7727254)

[Front end 6](#_Toc7727255)

[Sessions and Sockets 7](#_Toc7727256)

[Symptoms checker AI (Artificial Intelligence) 7](#_Toc7727257)

[WebRTC Video connector 7](#_Toc7727258)

[Database 7](#_Toc7727259)

[Authentication and Security 8](#_Toc7727260)

[Planning 8](#_Toc7727261)

[Algorithms 9](#_Toc7727262)

[Implementation 9](#_Toc7727263)

[The Architecture 9](#_Toc7727264)

[Database 10](#_Toc7727265)

[User login and Sign Up 11](#_Toc7727266)

[Registering 11](#_Toc7727267)

[Dashboard 11](#_Toc7727268)

[Symptoms Checker 12](#_Toc7727269)

[Video Connector 13](#_Toc7727270)

[Signalling 14](#_Toc7727271)

[Testing 15](#_Toc7727272)

[Mocha, Chai and Supertest 15](#_Toc7727273)

[Critical Appraisal 16](#_Toc7727274)

[Were the requirements met? 16](#_Toc7727275)

[What went wrong? 17](#_Toc7727276)

[Conclusion 18](#_Toc7727277)

[Bibliography 18](#_Toc7727278)

[Appendix 19](#_Toc7727279)

# Abstract

The National Health Service is one of the most important public sectors in the UK. The service provides care for all Citizens in the UK and is free at the point of use. The vast majority of NHS income directly comes from taxations and National Insurance contributions from citizens. One of the biggest issues facing the NHS is the large amount time patients have to wait in the A&E. One cause of this issue is due to the unnecessary attendance of patients to A&E. I have decided to create my project regarding the NHS as it was a sector I was passionate about.

For my project, I have developed an application where people who believe they have a condition but not sure if it’s serious enough to go to the hospital, can use to deduct their symptoms and have a diagnosis done online. This would be used as a filter to hopefully reduce A&E visitors, lowering the waiting times, strain on staff as well by the NHS budget.

One way in which I decided to help the user deduct their symptoms is through the use of an AI symptoms checker which conducts an interview with the user, examines the scope of their symptoms and is able to give a prediction on what they are potentially suffering from. The second way was through a direct video and audio conference with a medical professional conducted with peer to peer streaming with WebRTC.

# Introduction

*Aim*

The aim of my project is to create an application where a person would be able to find out what their illness is without having to directly visit the hospital. This would be achieved in two ways. The first one would be by using an AI to interview the patient regarding their symptoms to diagnose their condition. The second method would be through direct contact with a qualified medical professional which would be achieved with virtual interaction through video and audio. This would hopefully reduce the number of people physically going to the A&E for unnecessary reasons hence reducing the strain on NHS staff and their resources.

*Objectives*

1. Do market research to gain information into the viability of my project.
2. Learn how to build a web application from scratch.
3. Acquire the right resources to build a web application.
4. Use an agile methodology to map out my application

* Break down all functionalities into stories which can be completed in sprints
* Test my application regularly during development

# Survey of Literature/Information Sources

My initial plan was to create a Spring MVC servlet web project and integrate video API into it. However after research I found that it would be more clear and simple for me to create the application on JavaScript with WebRTC and Node.js to execute the code. I did the entire research using the web as I felt it had enough information.

## Node.js - Express

I chose to learn and implement Node.js for the backend of my application. I chose this because after researching online, there were a lot of recommendations to use Nodejs . Express is the web development package for NodeJs and this had all the libraries to build my application. I installed Nodejs from the official site (NodeJs). The site also gave me documentation on node methods. Pluralsight gave a beginners lesson on Nodejs and helped me to set up the ***npm*** environment. (Pluralsight Home)

I also used w3schools’s tutorials to give further understanding of Nodejs . (w3schools -nodejs)

## Socket.io

While researching, (Signalling protocols)I found out that I needed to use web sockets for signalling and getting a Peer-to-Peer connection. I decided to use socket.io as it was very popular and a lot of resources were available to learn me grasp it’s concept (Socket.io). It also consists of a Node.js server which made it compatible with my backend.

## JavaScript

I haven’t developed in JavaScript for a long time and I found my skills inadequate while working on the project. I used pluralsight and w3schools to start learning. (Pluralsight Home) (w3schools - JavaScript) I still have a lot more to learn.

## Infermedica API (Infermedica API documentation)

This was the API used for interviewing patients using their symptoms. They had a very confusing/ difficult to understand website regarding how to use the API. After spending several weeks I was able to grasp a good understanding of how the API worked.

## WebRTC

This was the first thing I did research on. Initially I decided to use an API from a 3rd party company called vidyo.io . However when I researched further into the API I realised that there was a better alternative called WebRTC (w3.org - WebRTC) which stands for web real time communication. “WebRTC stands for web real time communication. It is an open source project which enabled communication of data including audio and video in Web and native apps. It has several JavaScript APIs which helps you to capture record and stream audio and video between users. The main reason I changed my initial plan to a WebRTC JavaScript project was it offered me to write a rich, real-time application without requiring additional plugins (webrtc). I planned on using a 3rd party API which would have additional costs after I made a certain amount of requests. If I am using the JavaScript APIs instead, this would be free of charge and I have more control in writing the application.

These websites gave me a tutorial on how to create a basic WebRTC video app. (webrtc - codelabs) (Pluralsight Home) I didn’t get to finish the tutorial as I was having issues with my laptop. However I will be going back to it once I resolve them.

# Requirements

## Functional

* **An easy to use AI**

Since the target audience for my application is anyone above the age of 16 the application would need to be very easy to use and navigate.

* **Have a secure form for both the patient and medical professionals to login or signup.**

Each user would need to have an account in order to login. This would need to be safe and secure.

* **Have a database to store and retrieve their login and individual information.**

I would need to store all the user details securely in a database which will be used to retrieve the same information when they login.

* **The patient can choose whether they want to chat to a medical professional or use the AI system**
* **The patient will be able to commence a video and audio conference with a medical professional.**

When the patient is ready, they will be able to press a button which you commence a video conference between them and a medical professional

* **The patient will be able to see the qualifications and other information about the medical professionals online.**

When the user has logged and commenced a video conference with a medical professional, they would be able to see their qualifications.

* **If the symptoms are clear then the medical professional is able to send the patient a prescription for medicine they need which they would need to print off and take to the pharmacy**.

## Quality Attributes

* **Simplicity**

The application needs to be simple and easy to use. The target audience for my application is anyone over the age of 16. This means that I need to make the usability and navigation of the application simple enough for anyone to use as certain demographics will find it hard to navigate through otherwise.

* **Securable**

My application has to be secure as it will be dealing with personal and private information. The database needs to be secure as it would hold user login credentials. The conferencing needs to be secure as the patient is trusting that they are speaking to a medical professional and no one else.

Each of the medical professionals’ accounts will have to be authorised by the NHS to certify their qualifications. This will mean the patients are speaking to people who will actually be able to help them.

* **Reliability**

My application would need to be reliable as people will be using it when they are in need of help. It needs to be able to help the patient with understanding their problems. This means they should be able to talk to a medical professional as pretty much as soon as they log in. Also, the information they’re given should be correct. To make sure the patients receive the right advice, the medical professionals will have to be authenticated by the NHS. We also have a symptoms checker to help the professional with the diagnosis.

## Non-Functional

Resources

* IDE Software – NodeJS Express
  + I would need an integrated development environment capable of developing with NodeJS Express module.
* Tortoise SVN and GitHub
  + For version control
* Web Browser – Either Chrome or Firefox
* Databases-
  + I have to have a server to store all user login credentials.
* WebRTC JavaScript APIs – for video conferencing
  + I will be using an API from Vidyo.io to integrate the video conferencing aspect.
* Infermedica (Medical symptoms) API – for initial symptoms querying
  + This will be used once the user has logged in to query them so we can find out the general condition of the patient.

# Specification, Planning and Algorithms

## Overall Design Specification

### Programming Language

I initially decided to write it in JAVA Spring however after several weeks of researching I discovered that it would be difficult for me to implement all the functionalities. This was due to the lack of information available regarding spring and the particular APIs that I wanted to implement. After researching further, I decided to move my project onto NodeJS and JavaScript using NPM which is a package manager for NodeJs that hosts thousands of free packages to download and use. Below is how the framework of my application turned out.

### Web Framework

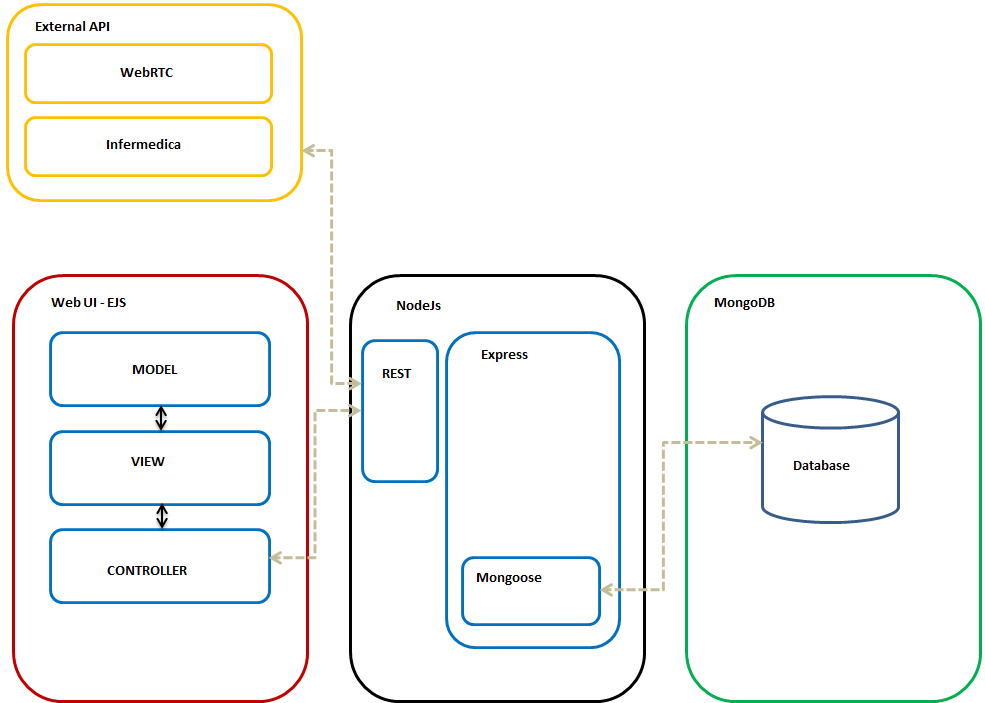


Figure 1 – *A simple visualisation of my project framework*

Figure 1 shows the architecture of my project. It consisted for 4 mains parts. The cores of the application is the NodeJs server which connects the MongoDB database, EJS web UI and external API together.

The framework that I decided to choose for the project was called ***express***. Express is a minimal and flexible framework which provided me with a robust set of features to develop within my application. One of the main reasons I chose this was due to the overwhelming support and guidance available for Express online.

### Front end

When deciding what front end framework to choose I came across EJS. This stands for Embedded JavaScript and is a simple templating language which lets me generate HTML mark-up with plain JavaScript. The reason I chose this rather than frameworks such as React or Angular was because I wouldn’t have to learn any new syntax or methodologies as ***EJS*** just cuts out the middle man and is just plain JavaScript which emits the HTML I wanted.

**Bootstrap –** This is a CSS framework which helps to develop a responsive front end. I have used Bootstrap in some of my webpages to help with the design and aesthetics.

**JQuery –** This is a JavaScript library which helps to simplify the traversal and manipulation of HTML pages. I have used JQuery to help me complete certain functions

### Sessions and Sockets

Each time a user logs in they create a session. This means when multiple users have logged in, their data won’t get mixed together and the application can detect which user is doing what. Also when they login a socket connection is initialised which helps connect users together. Having sockets also helps me to differentiate between my users. I can split the clients and the medical professionals into two groups.

### Symptoms checker AI (Artificial Intelligence)

One of the main features of my application is a symptoms checker API. This API is able to conduct patient triage and preliminary medical diagnosis. By querying the user about their symptoms, this API enabled me to diagnose their condition . The AI inference engine analyses the data and provides a list of likely conditions. This is possible thanks to the sophisticated algorithms the API uses to perform diagnostic reasoning. The request and response between the application and the API is done through JSON objects. These objects are manipulated in several ways to retrieve information to display to the user. I have explained this in more detail in my implementation section.

### WebRTC Video connector

The specification of the video connector was to gather user information through WebRTC API and pass it across two user browsers. A normal convention of passing information between users on different machines is through a socket connection however WebRTC allows the users to connect over their web browsers using peer to peer streaming. This makes the connection incredibly fast and you are able to not only transfer larger amounts of data but data of different media. . I have explained this in more detail in my implementation section.

### Database

The database that I decided to use for storing all the data for the project was MongoDB. The main reason I chose this was due to the great affinity MongoDB has with NodeJs. Through research I discovered that NodeJS had a great wrapper package for MongoDB called mongoose which was optimised for fast single queries which was ideal for my application. Also there was a lot of research material available to help with the integrations. It is through the mongoose module that the connection between NodeJs and MongoDB is made. Appendix 3 shows examples of the schemas I used in the database. 3.1 shows the schema for a patient and 3.2 shows the schema for a medical professional. For the patient, the data I stored was name, email and password. The latter which is hashed. These were the only information I asked for when they registered. The id and date values are auto-generated by MongoDB and "*isMP"* is a Boolean which lets the application know if the user is a medical professional or not. BY default all users will be set to false for this field and only the admin can change it after authentication.

### Authentication and Security

Firstly the most vulnerable part of my project is my database. MongoDB has built-in end to end security and since it’s a noSQL it isn’t susceptible to certain attacks like SQL injections.

At the current moment I don’t have a way to authenticate between a patient and medical professional upon registration. The way in which medical professionals are authenticated is by firstly registering to the application. Then they or the NHS would need to contact the admin with evidence of their qualifications. Once the admin has confirmed that they are a medical professional they can input their qualification and other professional details into the database and change the "*isMP"* function to true (see appendix 3.1)

The Symptoms checker API does not allow any form of submitting the user’s personal information and it only processes statistical and de-identified information which isn’t merged with any personal text. This makes the API very secure for anyone to use.

To help with the security of my application I used several ***NPM*** packages with NodeJS. These were:

* Passport – This is an authentication middleware. The main function of passport is to authenticate requests using mechanisms called ‘strategies’ . I’ve created my own authentication strategy and implemented it for my application. Each time a user calls a request such as login, the requested is parsed through my strategy and checks their credential details with what’s stored in the database. Only if the credentials match the strategy will the user be able to carry on.
* BCryptjs – this is a password hashing function and I’ve used it to encrypt the user’s password when storing it in a database. This is very important as if an intruder gains access to my database, he will not be able to gain access to the user password. Bcrypt was designed to be slow. This means that it reduces the number of passwords by second an attacker could has when crafting an attack.
* Session authentication- To make sure that users can just bypass the login stage of the application I’ve made sure to check at each point as the user is navigating through the website, they are authenticated and if not they will be sent back to login page.

ensureAuthetincation() – This is a function which I created to authenticate a user at each step of the application. This method is run each time a user is routed to a different page. Before redirecting them, it checks their current information to see if they have logged in. If so it allows the redirecting to happen, otherwise it sends the user to the login page. This method prevents a user and directly acting a page by just typing in the page URL.

## Planning

The planning for my project has changed a lot since the start of the year. This was due to time constrictions and not being able to complete task on-time, the research not being sufficient hence having to migrate the whole project into a different framework, not managing my time well between the project, other modules and my personal life and finally not being in regular contact with my supervisor to ask for guidance . Therefore overall I would say the planning for my project has been poor. Since I changed my framework, I pretty much had to start from scratch and there were few things I was able to transfer over. This happened at the end of January hence making a lot of the work I did in semester 1 pretty much obsolete. This was the plan after I change my framework in January 20/01/2019.

Table 1 – Project Deadline Plan

|  |  |
| --- | --- |
| **Objective** | **Deadline** |
| Create login and signup | 20/02/2019 |
| Symptoms checker API | 20/03/2019 |
| Video Connector | 15/04/2019 |
| Prescription generator | 30/04/2019 |

The plan for semester 2 didn’t go as anticipated. The 2nd objective of creating the symptoms checker took longer than the deadline. This was due to the complexity of the API documentation. Therefore it pushed back the deadline for all the other objectives.

## Algorithms

In my project I have used certain algorithms to apply certain functions. These are:

* Bcrypt hashing algorithm- I used this algorithm to has my passwords before storing them in the database. It incorporates a salt mechanism and adds it to the hashed password creating a long and unique string which is stored in the database. This string would be incredibly hard for someone to decrypt.
* Infermedica API algorithms – The symptoms checker API uses many algorithms to help with diagnostic reasoning such as when the AI has sufficient information to be able to stop and give diagnosis of a condition.

# Implementation

## The Architecture

Below shows all the files which make up my application.

* **config -** In this folder is where I have put the authentication code for relating a user to the database. In the passport.js file I have created the authentication strategy and the auth.js file is called each time a user navigates though the views to make sure they have been properly authenticated and isn’t using an old session.
* **models-** This file contains the model I use.
* **node\_modules-** This is the default file in a NodeJs application that stores all the relevant modules I need to run the application.
* **package.json** – This is also part of the default application as it refrences details about my application. These include information such as all the modules I have referenced in the application, scripts I can use to run the application etc.
* **public-** This directory contains all the JavaScript, CSS and images I used in the application.
* **routes -**This is where I store all my routing that I done in the application. It acts as the controller.
* **views** –The folder that contains all the views
* **app**.js – This is the very core of my project. This is a JavaScript file and is the focal point of the application as it is where I initialise my application and require all the needed modules and libraries.
* **test-** This contained my files for testing my application.

The way my application starts of is through the command NPM start. This runs my app.js file and this is what it does.

1. Requires all the necessary modules of the application and stores in in a variable ready to be used.
2. Initialises the app
3. Creates a connection to the MongoDB database. If there is an error in the connection the console will print an error
4. Sets up the static folders in my directories.
5. Sets the view engine. This is where I say what type of front end engine I want to use for the application.
6. Configures all the relevant models and adds their middleware. These including sessions, body parser etc.
7. Sets up the initial routing of the application which is mostly done in the /routes folder.
8. Sets up socket connection
9. Configures a port and starts the server, listening for any incoming requests.

Once step 9 is completed I can visit the port (<http://localhost:4000/>) and view the index view of the application which was set up in step 7. The /routes folder acts as the controller for the rest of the application and all the mapping is done there.

## Database

In order for me to create a user environment on the application the first thing I did was configure a database to store all their information. I chose MongoDB as the database and I found the necessary connecters to connect my application to it. I then created a schema for the user with fields such as “name” , “email” and “password” . This meant that on my user sign up form, each time a user registers it takes their information, maps it to the schema and adds the data as a collection into the database. Appendix 3 shows the format in which I have saved my users

## User login and Sign Up

As described before, the front end of the application is written in EJS which is what I used to create the login and sign up forms.

### Registering

When a user wants to register, they simply visit the register page and enter the details in the form. The form then posts that request to the controller which does various check with the data before successfully register the user. These checks are:

* All fields must be filled in and no are left blank
* The two attempts at the password must match
* The password must be at least 6 characters long
* The email must be unique and not have registered before

If any of these checks have failed then an error message will be shown to the user using flash. Once a user has successfully registered, they will be directed to the login page with an appropriate message and their details will be saved into the database including their password which is also encrypted before being stored. This is an example of a failed and a successful attempt at registering.

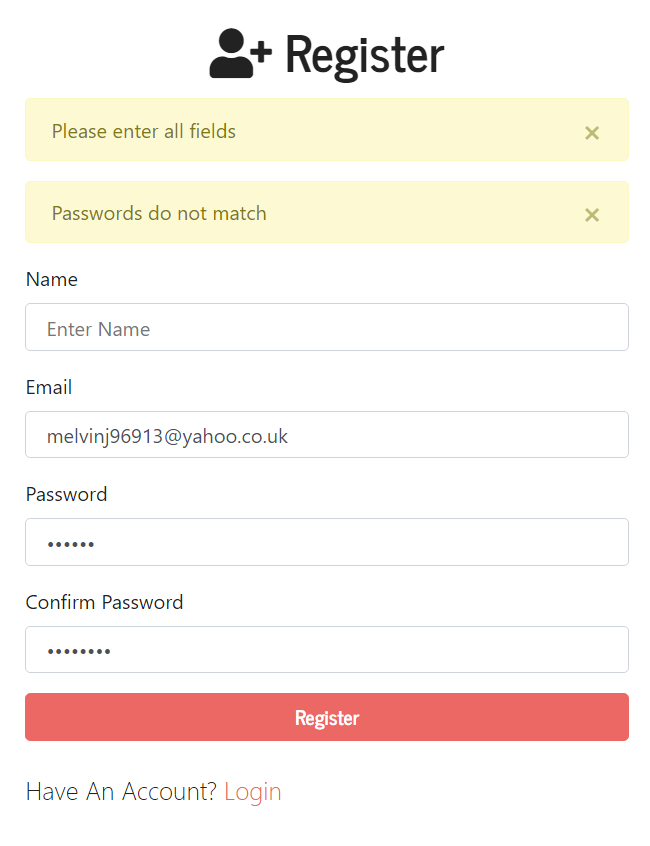
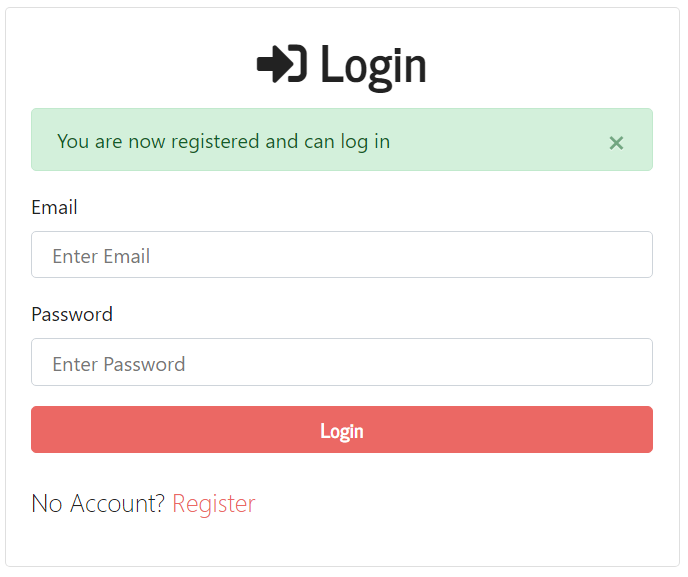


Figure 2 –*Two examples of flash errors returned in the login and register forms*

**Logging In**

When a user wants to log in they simply input their email and password they used to register. Similarly to register their details would be check with the database and appropriate error messages would be shown. Unfortunately I have not created a mechanism for users to retrieve their lost details and this could be something I implement later on.

## Dashboard

Once a user has logged in successfully, they would be sent to the dashboard. From here, they can choose to either use the symptom checker AI or talk to a medical professional. They are able to see how many medical professionals are online at that current moment.

## Symptoms Checker

Once the user has chosen to check their symptoms, they are sent to the ***symchecker.ejs*** page.

This is the initial page of their diagnosis and where the user enters a lot of initial information about them. These include:

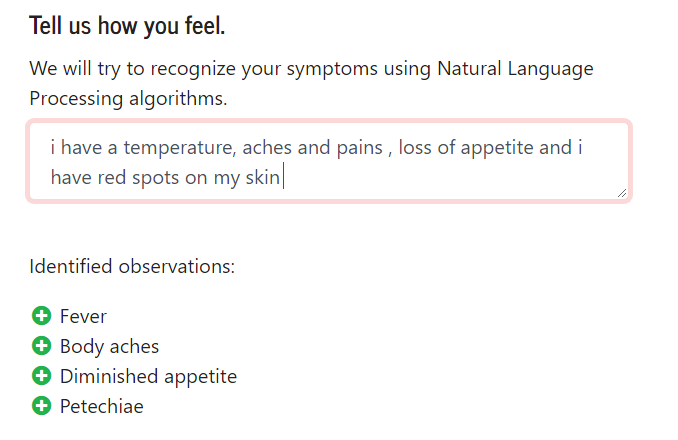
* *Age and gender*- This is so that the AI can automatically instantiate corresponding risk factors that may alter the prevalence of medical conditions. For example conditions such as arthritis are suffered less by younger people so this helps the API to deduce its probability in the user. The age attribute is expressed as a positive integer and the gender attribute can have either *‘male’* or *‘female’* values.
* *Demographics and History* – These are common risk factors people have such as *‘high cholesterol’* or that they *‘smoke’.* These also help to impact the base prevalence of various conditions.
* *Geographical location* – This is just to check whether the user has visited to again help with the initial diagnosis before the interview begins.
* *Natural Language Processing (NLP) Text area* – This is where the user can simple write how they are feeling. Whatever they have typed is then processed by the API and it will understand clinical concepts (symptoms and risk factors) mentioned by the user. *Figure 3* below shows how the natural language is processed for e.g. “I have a temperature” was mapped to “Fever”. The way this works is the API has stored thousands of symptoms with their name and common name along with other information as JSON object. It then tries to match any part of the text with either the name of common name. If there is a match, the symptoms name is returned. I will explain further about the way I implemented NLP in more detail below.

Figure 3 -*An example of the Natural language processing*

Once the user has entered all the relevant information on the page they can press next. This is when the interview will begin and the user is directed to ***interview.ejs*** . They will be asked a bunch of different questions. The questions would either be single where the options are either *yes*, *no*, or *skip,* group single which are radio buttons with one option able to be selected and finally group multiple which are checkboxes where multiple options are able to be selected. Appendix 1.4 shows the different types of questions. The user iterates through these questions until the API decides it has relevant information to give its diagnosis after which the user would be shown a summary page with what illnesses the user is most likely to suffer from.

I will now explain how I implemented the Infermedica API to diagnose the user. The way the API works is by sending a request to the API URL *https://api.infermedica.com/v2* with particular parameters, headers and body and then getting the response and picking out relevant information from it to with display it to the user or other means. The two man endpoint of the url that I used were /parse and /diagnose.

<https://api.infermedica.com/v2/parse> - This was the endpoint I used to help with the natural language processing. In the text area I implemented an *oninput* function called *nlp().* This meant that as soon as the user starts typing, it would run.

***nlp()*** *-* The purpose of the function was to retrieve the input from the user, send it as a request to the API, retrieve the response and populate the observations field like shown in Figure 3. When I got the information of the text area, I create a JSON object with it. Appendix 1.1 shows the object. The string in blue would be populated with the text the user has written. I would then use a built in JavaScript API called fetch() to send the object to the API and retrieve the response. Appendix 1.2 shows a typical response from the API as JSON object. I would parse the object and populate the observations list with the information.

<https://api.infermedica.com/v2/diagnosis> - This was the main endpoint of the API. It is what I used to conduct the interview and complete the diagnosis. I call it when the user has clicked *next* after inputting the relevant initial information. This runs my ***diagnose()*** function.

***diagnose()*** *–* This function retrieves all the initial information the user has inputted including the data collected from running ***nlp()*** and creates a big JSON object array with it. Appendix 1.3 shows an example of the object it creates. The “id” attribute represents each individual piece of information the user provided initially. For example the *id* for “BMI of 30” is *p\_7* and if the user selected it ,the *choice\_id* would be *present* . Next, a large JSON array is sent to the API using fetch(). The function then stores the JSON object array and the response in session storage and changes the view to *interview.ejs* . Once the view loads it runs the function *load().*

***Load()*** *–* This function firstly retrieves the JSON array and initial response from the API. It then populates the view with the question and the answers for the user to choose from and starts the interview. As the user presses next after selecting their answer, the method creates a single JSON object of the question. It then appends this object to the JSON array that was retrieved and using fetch() posts the information. The response it retrieved and the view is again populated with the new question that came with the response. The process is looped until a certain condition called *should\_stop* in the response is met. Each time this attribute is checked and if it’s false the view is populated with another question. If it’s true, then the view would be populated with the summary which is also retrieved from the response.

## Video Connector

Unfortunately due to time constraints I was unable to fully complete the implementation of the video connector between a patient and a medical professional however most of the complicated connections have been implemented and the patient is able to hold a video and audio conference with a medic. The steps involve are shown in Appendix 4. The way in which my video connection works as mentioned before is by using WebRTC APIs and socket.io for signalling. What this allowed me to do is connect from browser to browser between the patient and the medic.

### Signalling

Signalling is a process that I had to use to let a patient make an initial contact with a medic to let them know that they wanted to start a conference. WebRTC didn’t specify any methods in which to do this so I decided to use Socket.io. This was already implemented in my application as I was using it to check how many and what type of users were logging in to the application. In order for WebRTC to work, I had to use sockets to send and receive “ice-candidates” and “video-Offers” .

Video Offers

This was the first offer I had to pass from the patient to the medic. Initially I used a fetch() method to find the socket id of the medic the patient wanted to talk to. Then from the patient socket, I sent an initial offer to the medic socket with the patient web details. Once the medic socket received it, it returned an acceptance offer.

ICE Candidates

ICE stands for interactive connectivity establishment and it is a protocol used to traverse around network addresses by using another protocol called STUN. ICE candidates are sent from one browser to another to find the right path to connect to each other. Once the medic’s browser had received the acceptance offer, it started to send ice candidates to the RTCPeerConnection. This is a WebRTC API which is able to stream audio and video between users. It is able to detect when an ICE candidate is sent. Once detection is found, I would have to use sockets to send this ice candidate to the patient browser to check if it’s compatible. If not this step is repeated until a suitable ICE candidate is found.

#### WebRTC APIs

These were the APIs that I was able to use get the connection.

**getUserMedia():** This API captures the audio and video directly from the browser. When called, the user will be prompted to agree to the usage of the camera and microphone. The data that is streamed through them is separated into tracks and I was able to add these tracks onto the *RTCPeerConnection* API to send them away to display on the other browser as well as display the users’ media on their own page.

**RTCPeerConnection**: As explained above this API handles the actual traversal of information and was one of the most important APIs as it was the glue that held everything together. After I gathered the media from the user, I would add each track to RTCPeerConnection instance. This instance’s description would then be sent via socket to start the initial vide offers. Another instance of the same type is then created on the medic side. When the patient instance received an answer from the medic instance, it would start to receive ICE candidates from the browser. Once these are detected, they would be sent to the medic browser to find a match. When a match is found both instances send over the media tracks across the browser which are received and reproduced on the other browser starting a video conference.

I was able to complete the transfer of video between the users however the implementation is still incomplete and are missing many features to make it user friendly and acceptable to release.

These are the steps that I needed to have done in-order to get the video connector complete:

1. Firstly, I feel the layout of the video portal page is in adequate. The video are displays clearly however it would be better to move the smaller own user video out somewhere better on the page and maybe have a border.
2. The video is missing an end call button. This is an essential item as most video conference software have this and is pretty much standard. The way in which the user ends the call right now is my pressing the dashboard button or by the back button on the browser. This isn’t ideal and confuses the user.
3. Other video controls- These are things like the mute button and the volume button. Again to help the users in with adjusting the video to their liking.

In order to complete step 1, I would need to design and plan the page in a better way to make it aesthetically pleasing and also going with the theme of the rest of the application . In order to complete step 2 and 3 I would need to do more research on the video element and find out how to manipulate the video tag and apply user input function on the video itself.

# Testing

## Mocha, Chai and Supertest

For the testing of my application, I have used libraries called ***mocha, chai*** and ***supertest***. These libraries let me run asynchronous tests on NodeJs and in the browser. It allows flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Appendix 1 shows some of the test cases I implemented.

One thing I tested was to see if my routing was correct. Figure 2 shows my login and register pages and I tested to make sure if my application was successfully reaching these views. Appendix 2.1 shows the cases used to test the routing of the views. In this part I am testing my login page at /users/login and my register page at users/register. The module tests the REST API at the particular mapping and expects at certain string to be shown that would be on the corresponding view. In the first test, I am trying to get to my login page. This view was the text Login written which is what I am expecting to search for. In the second test it is the same except I’m searching for the text register.

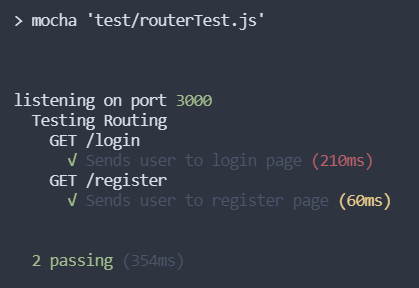
Once the cases are run the results are shown in terminal. Figure 7 shows the results of the test cases made in Appendix 2.1 . Both the cases have successfully passed. If a case was to fail the results would show that as well. Appendix 2.2 shows an instance of failure. If the mapping was incorrect, the results show what was expected and what was received which helps to quickly debug the issue.

Figure 7-The results from testing

# Critical Appraisal

## Were the requirements met?

Overall, I met most of the requirements I set for myself at the start of the project however I was unable to complete a few of the requirements which were key aspects to the project. I will explain the requirements that I met and the ones I didn’t

**Build an easy to use UI**- I believe I have met this requirement. The application is very easy to use right from the initial page. The font and theme are kept the same throughout the different pages which gives the application an identity. All the routing have been thoroughly tested and have been assured to not crash or take the user to a location they didn’t intend to go to.

**Have a secure form for both the patient and medical professionals to login or signup –** This requirement was also met both the patient and medical professional can log in and signup.

**Have a database to store and retrieve their login and individual information.** – This requirement was met as I fully implemented a MongoDB database and used it to retrieve user details. I believe this was a good choice as it had god compatibility with the framework and helped in achieving the requirement.

**The patient can choose whether they want to chat to a medical professional or use the AI system –** This requirement was met as when the patient has successfully logged in, they are given two options. Once is to check their symptoms and their other is to talk to a medical professional. Appendix

**The patient will be able to commence a video and audio conference with a medical professional.** – This requirement was mostly met and since I implemented the main functionality of the requirement and a patient is able to video chat with a medic without any issues. However it isn’t complete as it’s missing a lot of the common inputs which I was aiming for in my application. This very disappointing I couldn’t finish it on time. The reason I was unable to finish this was due to leaving this functionality towards the end as well as other reasons. I thought I would be able to get the other functionalities done fairly quickly and spend the most time on this one but due to missing deadlines, the time I had left to complete this functionality became smaller and smaller. In order to have achieved this requirement one thing I could have done was to re-evaluate my plan and changed it so that I prioritised on getting this requirement complete before the lesser important ones.

**The patient will be able to see the information about the medical professionals online.** – This is achieved as before the user commences the call, they are able to see information about the medics.

**If the symptoms are clear then the medical professional is able to send the patient a prescription for medicine they need which they would need to print off and take to the pharmacy.** – I was unable to set up the WebRTC **RTCDataChannel** method to connect between the patient and the medic and stream other form of data between them. Therefore I didn’t achieve this requirement. The reason I was unable to was similar to the last one as I left this requirement towards the end and ran out of time.

## What went wrong?

**Planning**

The main thing that went wrong for me was my planning and judgement. I initially thought that I would do the project on Spring MVC without realising that it would be much tougher to implement some of the functions and that there was very little information available about certain APIs and their integration with Spring MVC. I was really frustrated throughout semester 1 and struggled to produce a prototype. I also was too ambitious with my planning as I tried to complete certain functionalities of the project to early. My contingency planning was also very poor as once I missed a deadline I was too confused to know what part to focus on and to re-organise the tasks accordingly.

**Time-management**

Another aspect that went wrong for me was my time-management. I really struggled to balance my project around my other modules , work and social life. This really impacted on the amount of hours I actually spent coding on the project. This really affected the tight deadlines I had already set causing many setbacks.

**Getting support**

Throughout the entire project I didn’t manage to utilise the support from my supervisor. I missed many meetings and didn’t make contact when I was struggling. I also feel I was lacklustre in my research. It took me too long to solve certain issues and bugs I was having which were a main cause to not meeting deadlines on time.

### Context

**Economic**

My application can have a potential impact on the economy. If more people were using the application rather than visiting the NHS, it would reduce the amount of money and resources the government have to spend in maintaining the service. This could have a positive impact as if the government can spend the money elsewhere to improve the economy further. On the other hand, one way in which I could potentially see the application having a negative effect is by the application increasing the number of people visiting hospitals. This could happen by incorrect analysis of patient symptoms. If a patient uses the AI to check their condition and if it diagnoses something serious, then the user is likely to get it check out by a real medic.

**Social**

The application could potential help the social context of people as it makes it easier for someone to gain a medical examination at the comfort of their home. This could help the user identify any illnesses they thought were not serious enough to go to the hospital for but if the application diagnosed them with potentially having something then they would be able to discover it sooner than if they had just left it. This would potentially help people to stay healthy.

This could also work the other way as it could get more people stressed out over conditions which they were wrongfully diagnosed with.

The concept of my application isn’t used in the wider economy yet as people would still fell much better if they had a physical medic diagnosing them rather than someone over the internet. However in the recent past there has been increasing online medical support for patients

**Sustainability**

In order to sustain my application, it would need to be taken over by a large company such as the NHS. To have to incentivise medical professionals to work to give advice to people would be through actual pay. Another way I could take it is by making it a non-profit application where medical professionals who have free time can sign up and give advice to people for free. This method seems more logical for me to start however it would be difficult to gather the medical workforce needed, especially ones who would do it for free.

### Personal Development

In terms of how I have developed, I feel I have gained a lot of knowledge on web development especially handling RESTful APIs as well as using and implement external APIs. I have improved my JavaScript programming and know how to fully develop and run a NodeJS application from scratch. As the first half of my project didn’t go well it has really taught me a lesson on management has a whole. Managing my time, resources and support have all been improved during the second half of the project.

# Conclusion

For this project, I have developed a web application with a variety of features for two kinds of users. The first type is patients who want to check if they really need to visit the hospital or not. The second are medical professionals who are there to give advice to the patients . The application features a very easy to use user interface making sure almost anyone is able to use it.

The first functionality is login and register system where user information is saved and retrieved from a database. This has been implemented very reliably and securely authenticating the user at each step of the application. Another feature of the application can then be used to check the symptoms of the patients and give a diagnosis of conditions they might me suffering from using Artificial Intelligence. The patient is also given a probability of how likely they are to have the diagnosed condition. The second main feature of the application is for the patient to hold a video conference with a medical professional. This was mostly implemented, however due to time constraints this is still in development and not ready for use.

The application is able to be extended with further functionalities in the future. You can add further details to the schema of the patient and the medic where I can add things like medical history to the patient which can be passed onto the medic when doing the diagnosis. Also I can add integrate the database I have with another one such as an official NHS database. This would help me to auto-register the medics if their information is available as well as gather much more information.

# 

# Bibliography

*Infermedica API documentation*. (n.d.). Retrieved 2019, from Infermedica: https://developer.infermedica.com/docs/api

*NodeJs*. (n.d.). Retrieved from NodeJs: https://nodejs.org/en

*Pluralsight Home*. (n.d.). Retrieved November 21, 2018, from Pluralsight : https://app.pluralsight.com/

*Signalling protocols*. (n.d.). Retrieved from bloggeek.com: https://bloggeek.me/siganling-protocol-webrtc/

*Socket.io*. (n.d.). Retrieved November 21, 2018, from socket.io: https://socket.io/docs/

*w3.org - WebRTC*. (n.d.). Retrieved November 10, 2018, from w3.org: https://www.w3.org/TR/webrtc/

*w3schools - JavaScript*. (n.d.). Retrieved from w3schools: https://www.w3schools.com/js/

*w3schools -nodejs*. (n.d.). Retrieved November 21, 2018, from w3schools: https://www.w3schools.com/nodejs/default.asp -

*webrtc*. (n.d.). Retrieved November 16, 2018, from webrtc,org: https://webrtc.org/

*webrtc - APIs*. (n.d.). Retrieved November 20, 2018, from w3c.github.io: http://w3c.github.io/webrtc-pc/

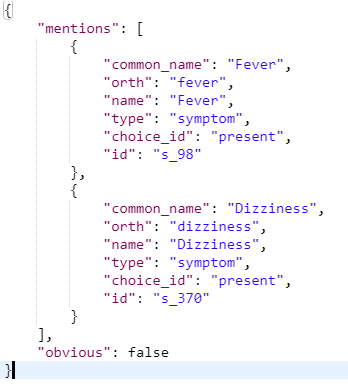
*webrtc - codelabs*. (n.d.). Retrieved from codelabs.developers.google.com: https://codelabs.developers.google.com/codelabs/

*webrtc-infrastructure*. (n.d.). Retrieved from html5rocks.com: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

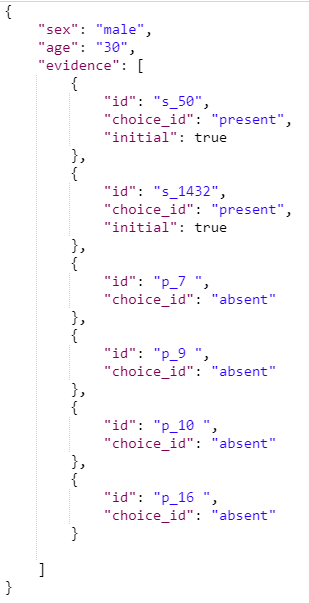
**APPENDIX**

**1 - Sending and retrieving data from Infermedica API**

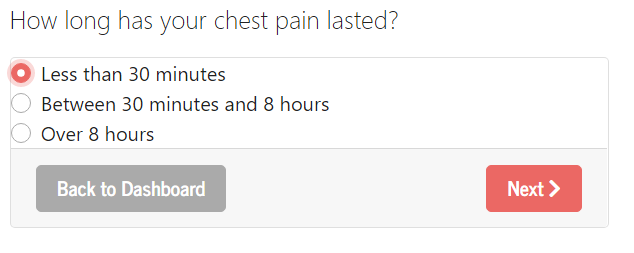
* 1. **s**

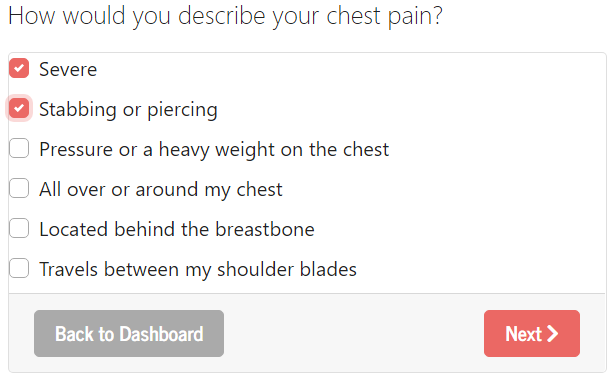


**1.2**

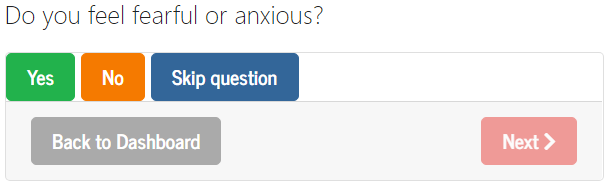


**1.3**

**1.4**



Group Multiple

****

single

Group single

**2- Testing**

**2.1**





**3– Database**

**3.1**



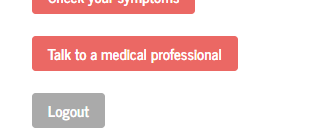
3.1

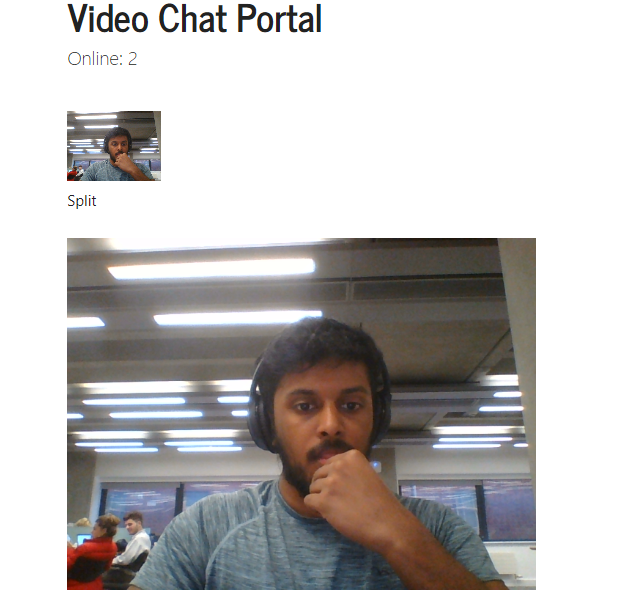
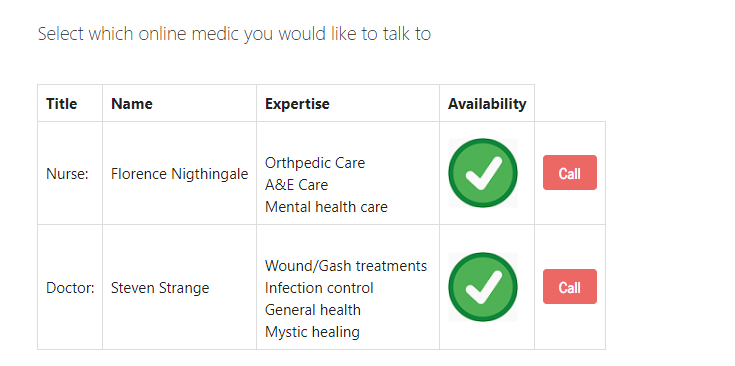
**3.2**

**4- Video connector**

**Steps to connect to a medic from patient side**

**Step 1** – Click on the “Talk to a medical professional” button which takes the patient to the video portal





**Step 3** – After pressing the button the user will be able to see two videos one of them and another of the medic.

**Step 2** – from the portal choose the medic whom they wish to call and press the call button.