



Covid Digital Doctor

Project Engineering

Year 4

Brian Sharkey

Bachelor of Engineering (Honours) in Software and
Electronic Engineering

Galway-Mayo Institute of Technology

2020/2021



Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Brian Sharkey

Acknowledgements

I would like to thank the lecturers in GMIT for helping me complete this project.

Table of Contents

1	Summary.....	7
2	Poster.....	8
3	Introduction.....	9
4	Background.....	10
4.1	Background – AWS EC2.....	10
4.2	Background – Oximeter.....	10
4.3	Background – IoT Core.....	10
4.4	Background - Bluetooth Low Energy.....	11
4.5	Background - Node.js.....	11
4.6	Background – MongoDB.....	11
4.7	Background - ESP32.....	11
5	Project Architecture.....	12
6	Project Plan.....	13
7	ESP32 Code.....	14
7.1	Connecting ESP32 to Oximeter.....	14
7.2	Connecting ESP32 to Wi-Fi.....	15
7.3	Publishing to AWS IoT Core.....	15
8	Covid Digital Doctor Web App.....	17
8.1	Register.....	17
8.2	Creating New Patients.....	19
8.3	Sending Emails.....	20
8.4	Receiving data from AWS IoT Core.....	22
9	Ethics.....	24

9.1	Confidentiality:.....	24
9.2	Security:.....	24
9.3	Data Inaccuracies:	24
10	Conclusion	25
11	References	26

1 Summary

As part of the project module of fourth year in the Software and Electronic course in GMIT each student was required to complete a software and/or electronic engineering application area.

The project is expected to provide the students with valuable experience in project management. The students in the process will create ideas, research, and plan out their projects accordingly. They will use their knowledge, skills, and competence, to deliver a project of significant technical and academic challenge.

The students also have the opportunity to improve their soft skills as there are presentations, reports, videos, and a poster that are mandatory deliverables throughout the year. 25% of the final grade is assessed at Christmas where students are required to present the work completed to that point. The remaining 75% will be examined in the summer with the emphasis being on functionality and demonstration.

I opted to create a web application as my final year project. The main function of my project is to allow doctors to view their patients' heart rate and oxygen saturation levels remotely. The important features of my project are:

- User Registration.
- User login.
- Patient creation.
- Patient emails.
- Logging Oximeter readings.
- Displaying Patient Information.

The main technologies that were used were:

- Node.JS
- MongoDB.
- MQTT Iot Core.
- AWS.
- BLE

2 Poster

Covid Digital Doctor

Brian Sharkey
Bachelor of Engineering (Honours) in Software and Electronic Engineering
Galway-Mayo Institute of Technology

I propose a device that can be sent home with a patient that can monitor the heart rate and blood oxygen saturation as this is the main indicator for the need of ventilation. The devices sensor data will be available to the patients' doctor via the 'Covid Digital Doctor' web application. The web application can keep both the patient and the doctor in contact by storing patient contact information and will allow the doctor to quickly send emails notifying the patient of the need to return to the hospital.


Introduction:

The Covid – 19 pandemic is causing major disruptions all over the world. One of the major fears during this time is that our healthcare systems will be overwhelmed by the numbers of people requiring care. This is of great concern as there is only a finite number of ICU beds. Preserving the number of ICU beds available for people that really need them is the aim of my project. After consulting a healthcare professional working in a local hospital, she highlighted to me what she thought was a flaw in the way in which Covid- positive patients were processed in the hospital.

She offered me an example:

A man walks into A&E and he tests positive for Covid-19. The man is middle-aged. The staff in the hospital are left with a dilemma. Do they let the man go home to self-isolate or do they admit him to a hospital bed? The problem is the man may appear healthy enough to go home (not requiring ventilation), but he may deteriorate at home. This leads to most cases as such being kept in hospital and the patient is often occupying a bed they do not need.

Application Scenarios:




Doctor

Retrieve Data from sensors:

- SPO2,
- BPM

- Notify Patients to return to hospital.
- Create patients.
- Log Patient details
- Sign up/ Sign in

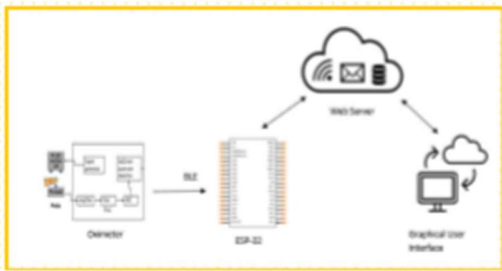


Patient


- Wear oximeter.
- Send data to Application.
- BPM, SPO2 etc.
- Provide details to doctor

- Return to hospital when notified.
- Rest at home.

Architecture Diagram:



Result Example:




The picture above is an example of the main feature of the application. The application is displaying heart rate and blood oxygen saturation levels of patient 'Brian Sharkey'.


Technologies:

- Node.js
- AWS Hosting Services.
- AWS IoT Core.
- ESP32
- Oximeter.


- Low-Energy Bluetooth.
- C Programming Language.
- JavaScript.
- HTML.
- MongoDB



Department of
Electronic and
Electrical Engineering



COVID DIGITAL DOCTOR



3 Introduction

The Covid – 19 pandemic is causing major disruptions all over the world. One of the major fears during this time is that our healthcare systems will be overwhelmed by the numbers of people requiring care. This is of great concern as there is only a finite number of ICU beds. Preserving the number of ICU beds available for people that really need them is the aim of my project. After consulting a healthcare professional working in a local hospital, she highlighted to me what she thought was a flaw in the way in which Covid- positive patients were processed in the hospital.

She offered me an example:

A man walks into A&E and he tests positive for Covid-19. The man is middle-aged. The staff in the hospital are left with a dilemma. Do they let the man go home to self-isolate or do they admit him to a hospital bed? The problem is the man may appear healthy enough to go home (not requiring ventilation), but he may deteriorate at home. This leads to most cases as such being kept in hospital and the patient is often occupying a bed they do not need.

My solution to this problem is to create a device that can be sent home with a patient that can monitor the heart rate and oxygen saturation levels in their blood as this is the main indicator for the need of ventilation. The devices sensor data will be available to the patients' doctor via the 'Covid Digital Doctor' web application. The web application can keep both the patient and the doctor in contact by storing patient contact information and will allow the doctor to quickly send emails notifying the patient that they need to return to the hospital.

4 Background

I will use this section to describe the technology featured in my project.

4.1 Background – AWS EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web-based service that allows users to run their applications in the AWS public cloud. The service operates by creating virtual machines where applications are executed. AWS users can easily manage instances of each virtual machine via the AWS web interface. The service offers an enormous amount of control over the instances which is one of its foremost benefits, which makes the operation as simple as if it was being run locally.

4.2 Background – Oximeter

Pulse oximetry is a very effective and non-invasive method of monitoring a person's blood oxygen saturation levels. The sensor is commonly placed on the fingertip as it is an area of the body that is thin and has adequate blood flow. The oximeter produces its readings using an electronic processor and a pair of small light emitting diodes (LED's) facing a photodiode. One of the LED's is red and the other is infrared. The light is passed through the fingertip and light absorption in the blood is recorded. If more infrared light passes through the fingertip that is an indication of deoxygenated haemoglobin and if more red light is passed through the fingertip that is an indication of oxygenated blood. A numeric value can then be derived by a mathematical equation. Oximeters are very easy to operate and are often equipped with Bluetooth.

4.3 Background – IoT Core

AWS IoT Core is one of the many services provided by Amazon. It is a cloud service that allows developer to connect, control and manage IoT devices. AWS IoT Core supports device connections that use the MQTT protocol. To get the IoT core connected to your devices you must create a 'thing' and attach connection policies and certificates. When done correctly you should now be able to publish and subscribe to topics on your created 'thing'. This can generally be accomplished with the use of an SDK.

4.4 Background - Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless personal area technology designed to be incorporated in applications in the sport, medical, security industries among others. The key difference between Bluetooth and BLE is that BLE is much less power consuming. This is ideal for small devices with limited battery power. BLE will remain in sleep mode until a device attempts a connection and each connection will only last a few milliseconds. These features make it ideal for applications and devices that exchange small amounts of data.

4.5 Background - Node.js

Node.js is an open source back-end JavaScript runtime environment. It allows developers to use event-driven programming on web servers, which allows developers to create fast web servers in JavaScript. Node.js is very useful when creating your own applications as you can create backend applications which is ideal for any application that deals with data. Node.js makes it easy to interface with databases.

4.6 Background – MongoDB

MongoDB is an open-source NoSQL database. MongoDB is described as document-orientated which means the data is stored like a JSON document. Each document in the database is defined by its schema which is essentially is a set of rules which defines the kind of data that is accepted. This makes MongoDB extremely flexible in terms of storage and ideal for developers dealing with many different types of data such as integers and strings etc.

4.7 Background - ESP32

The ESP32 is a low-power microcontroller with Wi-Fi and Bluetooth integrated. The ESP32 can be programmed in the Arduino IDE making it very user friendly. Many of the Arduino libraries can be used with the ESP32. It is relatively easy to connect the ESP32 to many other electronic devices given its Wi-Fi and Bluetooth capabilities which makes it well suited to an IoT project.

5 Project Architecture

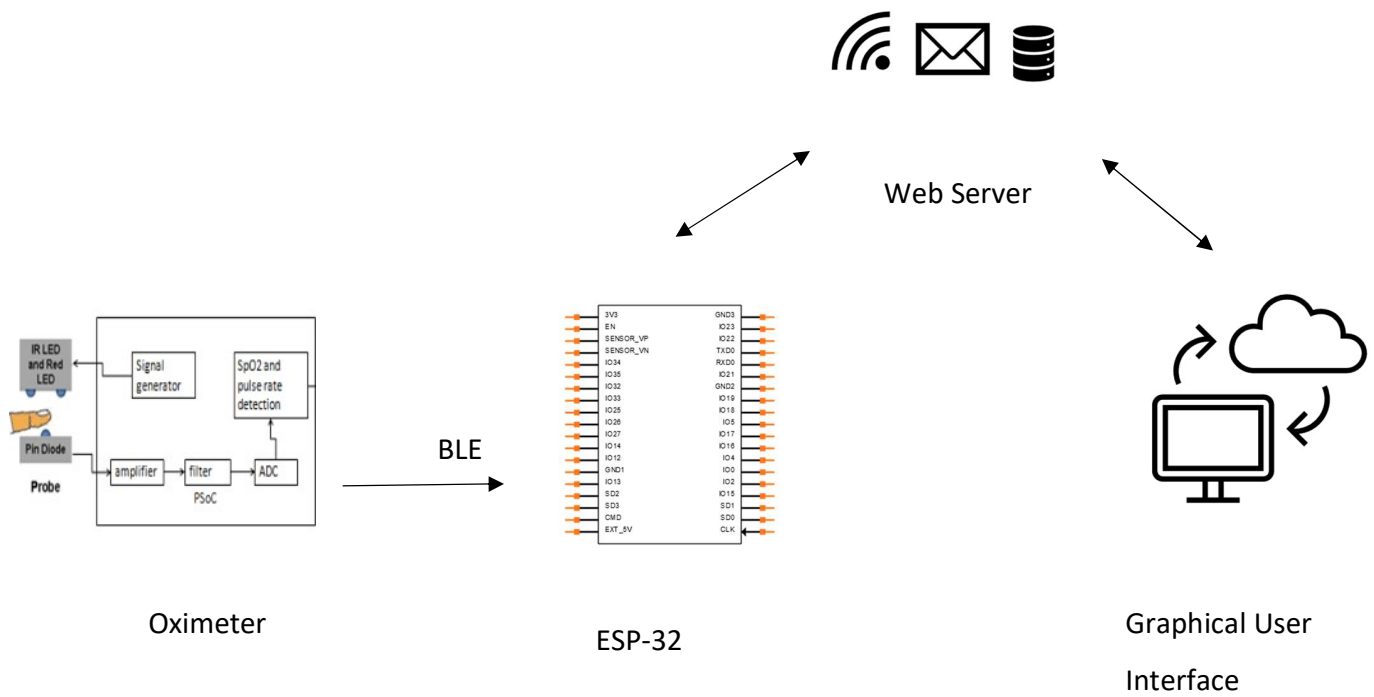
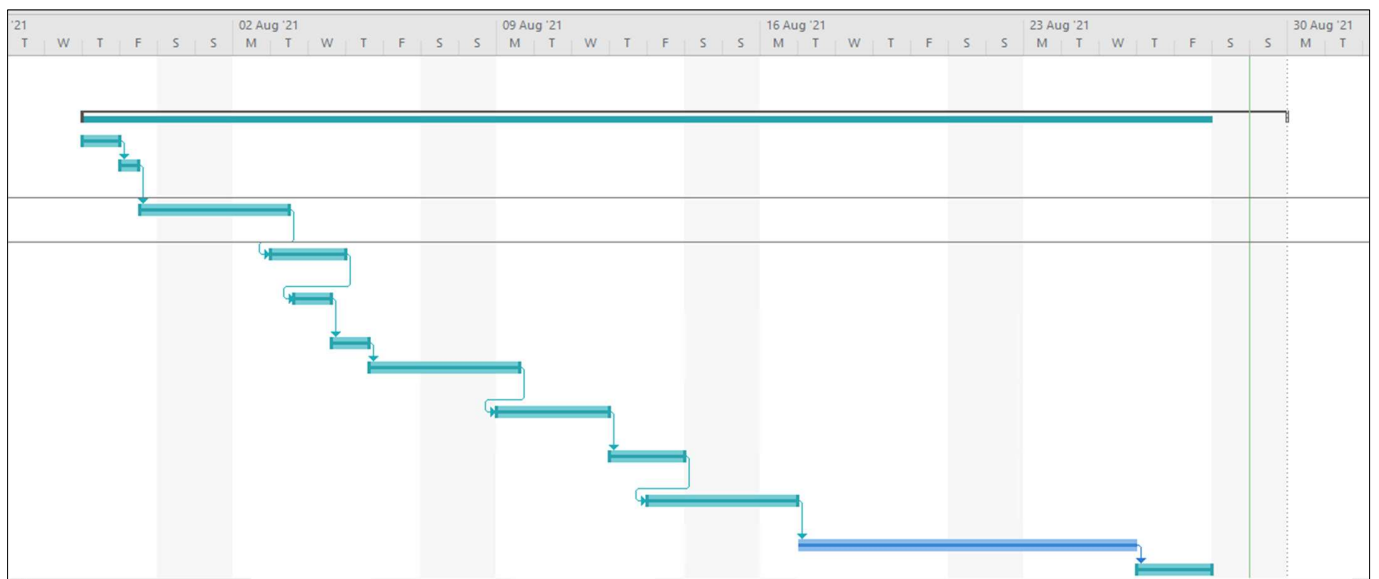
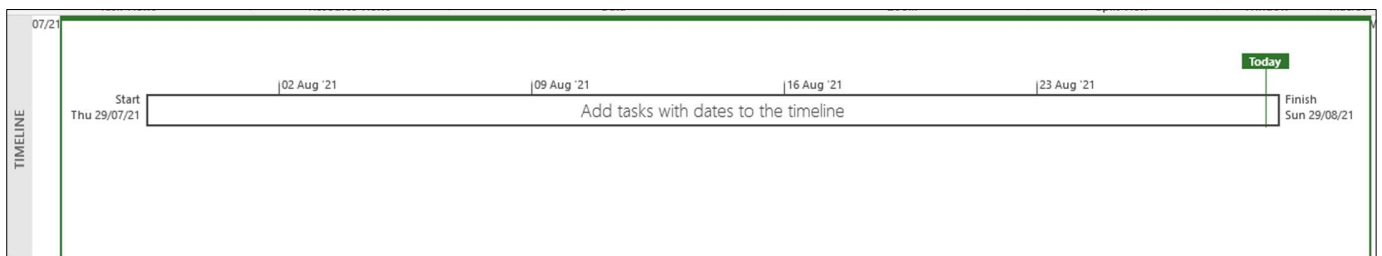


Figure 5-1 Architecture Diagram

6 Project Plan

✓	✈	Covid Digital Doctor	23 days	Thu 29/07/21	Sun 29/08/21	
✓	✈	Create Login feature	1 day	Thu 29/07/21	Thu 29/07/21	
✓	✈	Add patient badge to navbar	0.5 days	Fri 30/07/21	Fri 30/07/21	4
✓	✈	Creating routes between pages	2 days	Fri 30/07/21	Tue 03/08/21	5
✓	✈	Source and add all needed images for	2 days	Tue 03/08/21	Wed 04/08/21	6
✓	✈	Create 'Information' page and 'Create	1 day	Tue 03/08/21	Wed 04/08/21	7
✓	✈	Add nodemailer to project	1 day	Wed 04/08/21	Thu 05/08/21	8
✓	✈	Get webpage hosted on AWS	2 days	Thu 05/08/21	Mon 09/08/21	9
✓	✈	Get AWS IoT Core publishing data to my	3 days	Mon 09/08/21	Wed 11/08/21	10
✓	✈	store data from aws in mongo db	2 days	Thu 12/08/21	Fri 13/08/21	11
✓	✈	add chart to patient profile display	2 days	Fri 13/08/21	Mon 16/08/21	12
✓	✈	Create Video	7 days	Tue 17/08/21	Wed 25/08/21	13
✓	✈	Create Video	2 days	Thu 26/08/21	Fri 27/08/21	14



7 ESP32 Code

The ESP32 plays an important role in Covid Digital Doctor. It is the link between the oximeter and the web application. I found that the ESP32 is a very powerful piece of hardware. It possesses the ability to connect to multiple devices via Wi-Fi and BLE at the same time. I used the ESP32's BLE capability to connect to my oximeter and retrieve the BPM and SPO2 values and send them my web app using the AWS IoT core using its Wi-Fi capability.

7.1 Connecting ESP32 to Oximeter

My first step in programming my ESP32 was to connect to the oximeter. I used the 'BLE-client' example from the ESP32 BLE Arduino library as a starting point. This example code allowed me to connect to a device and read its values which are then displayed in the serial terminal. To get this example to work I had to first find the service UUID and the characteristic UUID of the sensor. These are unique identifiers that indicate which device and service you require. I did this by downloading the 'BLE Scanner' app and connecting it to my oximeter. This displays an array of information about the oximeter. You can find the service you need by reading from the characteristic and if the return value changes this is likely your sensor characteristic. Once I had found the correct UUID's I now had the values displaying in my terminal. I then parsed the data and stored them in appropriately named variables.

```
|  
// The remote service we wish to connect to.  
static BLEUUID serviceUUID("cdeacb80-5235-4c07-8846-93a37ee6b86d");  
// The characteristic of the remote service we are interested in.  
static BLEUUID charUUID("CDEACB81-5235-4C07-8846-93A37EE6B86D");
```

7.2 Connecting ESP32 to Wi-Fi

Wi-Fi is relatively simple to connect to from the ESP32. I used the 'Wi-Fi.h' library and there are many examples that show how to connect to your Wi-Fi. All that is needed is the SSID and password of your internet connection.

```
WiFi.mode(WIFI_STA);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

Serial.println("Connecting to Wi-Fi");

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
```

The code here prints a "." Every half a second until it establishes a connection to the network.

7.3 Publishing to AWS IoT Core

To get my ESP32 to publish its data to the AWS IoT core I first had to configure the IoT core on the AWS management console. Once you have done that correctly, you will have a 'thing' to communicate with. Your 'thing' will have policies and certificates attached. These certificates are very important for your ESP32 code as they provide AWS IoT with the ability to authenticate client and device connections. I started my code with a template called 'ESP32_AWSIoTCore_Template'. This template required me to add the AWS certificates I had created. Once that was added I was able to connect to the AWS IoT core

```
// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGMGQWlhem9uMRkwFwYDVQQDExBBbWp6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTE1MDUyNjAwMDAwMFoOTEL
MAkGA1UEBhMCVVMwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKcXj
b3QgQ0EgMTCCASiWdQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKcXj
ca9HgFB0fW7Y14h29Jl09lghYF10hAEvzAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
906II8c+6zfltRn4S9Wiw3te5djdY26k/oI2peVKVuRF4fn9tBb6dNqcmsUSL/qw
IFAGbHrQgLFm+a/sRcmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/OegwLX0tdHA114gk9S7EWW67c4cX8jJGKLhD+rcdqsq08p8kDilL
92FcXmn/6pUCYsiKrlA4b9v7LWibxcccVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAaANCMCAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFiQYsIU07LwMLJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCj8jdaQ2ChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
USFMCCjJmCXPI6T53iHTfIUJrU6adTrCC2qJeH2ERxh1bI1Ejjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8vIT096LXFvKW1JbYK8U90vv
o/ufQJVtMVT8QpPHRh8jrdkPSHCa2XV4cdFyQsR1bld2wgJcJmApsyM2Fo6IQ6XU
5MsI+yMRQhDKFJioaldXgJUKK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTF4688QvvG5
-----END CERTIFICATE-----
)EOF";
```

Here is an example of one the certificates that needed to be included to authenticate a connection

Once these three programs working independently. The only remaining task was to combine them to retrieve data from the oximeter and send it to AWS IoT core. I used my AWS code and added the BLE and Wi-Fi code and ensured they were all working separately within the one sketch. The major change that I made with the sketch at this point was I altered the `publishMessage()` function to accept two variables. This allowed me to pass the bpm and spo2 values to the function where it could then be sent to IoT core. The sensor data was now showing in the AWS test client.

```
void publishMessage(int data1, int data2)
{
    StaticJsonDocument<200> doc;

    doc["BPM:"] = data2;
    doc["SPo2:"] = data1;
    doc["time"] = millis();
    char jsonBuffer[512];
    serializeJson(doc, jsonBuffer); // print to client

    client.publish(Oximeter, jsonBuffer);
}
```

`PublishMessage()` accepts two variables. Which are then added to a document and converted to a JSON so it can be sent to IoT core

8 Covid Digital Doctor Web App.

The Covid Digital Doctor Web app was created using Node.js and express. Express was used to create a template to work with. The app serves many functions in my project, through the app the user can:

- Login/Logout.
- Register
- Create new patients.
- View patient details.
- Store patient details.
- Notify patients via email.

The application of course provides more function than these four topics but as these are the most complex features, I will describe these in more detail.

8.1 Register

Registering a user refers to the sign-up functionality of the application. The user can select the option from the navbar and are presented with a form. The form contains two fields, an email field, and a password field. Once the user enters valid information the email and associated password is stored in the user's collection in Mongo dB and can be used to login in future.

To authenticate the username and password I used a package called 'Passport.js'. Passport works with what are referred to as strategies and there are many pre-defined strategies available in the documentation such as a Facebook login. But for this project I created my own strategy. The main aim of my strategy is to ensure that a valid email and password have been entered and that the email entered has not been used before. Before I add the valid information to the database, I first encrypt the password for added security using the b-crypt package available in Node.js.

```

var userSchema = new Schema({
  email: {type: String, required: true},
  password: {type: String, required: true}
});
userSchema.methods.encryptPassword = function(password) {
  return bcrypt.hashSync(password, bcrypt.genSaltSync(5), null);
};
userSchema.methods.validPassword = function(password){
  return bcrypt.compareSync(password, this.password);
};

module.exports = mongoose.model('User', userSchema);

```

In the userSchema where I define the information accepted in the user document, I define a method using b-crypt to encrypt the password.

```

passport.use('local.signup', new LocalStrategy({=
}, function(req, email, password, done) {
  req.checkBody('email', 'Invalid Email').notEmpty().isEmail();
  req.checkBody('password', 'Invalid password').notEmpty().isLength({min:4});
  var errors = req.validationErrors();
  var messages = [];
  if(errors){
    errors.forEach(function(error){
      messages.push(error.msg);
    });
    return done(null, false, req.flash('error', messages));
  }
  User.findOne({'email':email}, function(err, user){
    if(err){
      return done(err);
    }
    if(user) {
      return done(null, false, {message: 'Email is already in use.'});
    }

    var newUser = new User();
    newUser.email = email;
    newUser.password = newUser.encryptPassword(password);
    newUser.save(function(err, result){
      if(err){
        return done(err);
      }
      return done(null, newUser);
    });
  });
});
});

```

This is where I define my strategy using passport. It checks to make sure both input fields are not empty. It checks to ensure that the email is an email address.

It makes sure that there is no user with the same email and then adds the user to the database and encrypts the password.

8.2 Creating New Patients

The application allows the user to create a new patient. They can access this feature by clicking the link from the navbar if they are logged in. There are 5 input fields displayed to the user:

- First Name
- Last Name
- Email
- PPS
- Age

A post method is then used to pass the data to the database. A 'router. Post' is used on the server side and that is where the data from the form is extracted and used to create a patient.

```
router.post('/newpatient', isLoggedIn, function(req,res,next){  
  var patient = new Patient({  
    firstName: req.body.first,  
    lastName: req.body.last,  
    email: req.body.email,  
    ppsNumber: req.body.pps,  
    Age: req.body.age,  
    imgPath: 'https://cdn3.iconfinder.com/data/icons/vector-icons-6/96/256-512.png'  
  });  
  
  patient.save(function(err,result){  
    exit();  
  });  
  
  function exit()  
  {  
  }  
  res.redirect('/covidapp/index');  
});
```

I set the variables defined in the patient schema to the corresponding values in the form on the html page. I also pass a generic image to be displayed on the patients' thumbnail. The BPM and SPO2 values are not required here because I set them to not be required in the schema.

8.3 Sending Emails

A crucial part of the project was enabling the user to view patient details and quickly send an email to notify the patient of the need to return to hospital. Node provides a fantastic package called 'nodemailer' that can send emails from an application [2].

On the patients' profile page that displays charts of the patients BPM and SPO2 levels I added a button directly underneath that sends an email to the patient the user is currently looking at.

The patient is found by passing it's unique Id to in the POST method. Using the Id I can extrapolate which patient the user requires using the mongo function findById(). New variables were then created to store the attributes of the patient. The patient's email could then be passed to nodemailer. Nodemailer requires an email and password of the email address sending the email and the email address of the destination.

```
<form style = "max-width:480px; margin:auto;"action="/user/profile/{{this.id}}" method = "post">
```

The Id of the current patient is passed from the POST method.

```
router.post('/profile/:id', isLoggedIn, function(req,res,next){
  console.log(req.params.id);
  var patient_id = req.params.id;
  var first, last, pps,age,spo2,bpm, email;
  Patient.findById(patient_id, function(err, patient) {
    if(err){
      return res.redirect('/')
    }
    first = patient.firstName;
    last = patient.lastName;
    pps = patient.ppsNumber;
    age = patient.Age;
    spo2 = patient.spo2;
    bpm = patient.bpm;
    email = patient.email;
```

The Id is then used server side and the findById() function is called. Once the patient is found their details are stored.

```
const output = `
<p>Hello,</p>
<br>
<p>The Covid Digital doctor system has detected that your
  oxygen saturation levels are low. It is advised you return to hospital.</p>
<br>
<p>Thank you,</p>
<br>
<p>The Covid Digital Doctor Team.</p>
`;
```

This is the output written in html and will be sent as the body of the email

```
let transporter = nodemailer.createTransport({
  service: 'gmail',

  auth: {
    user: 'CovidDigitalDoctor@gmail.com', // generated ethereal user
    pass: 'GMIT2020/2021' // generated ethereal password
  }
});

// setup email data with unicode symbols
let mailOptions = {
  from: '"Covid Digital Doctor" <CovidDigitalDoctor@gmail.com>', // sender name
  to: email, // list of receivers
  subject: 'Oxygen saturation is low!', // subject line
  text: 'Hello world?', // plain text body
  html: output // html body
};

// send mail with defined transport object
transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    return console.log(error);
  }
  console.log('Message sent: %s', info.messageId);
  console.log('Preview URL: %s', nodemailer.getTestMessageUrl(info));
  // res.redirect('/covidapp/index',{msg: 'Email sent!'});
});
```

The senders email details are entered first.

Here, details about the email itself are entered. Set the email to the email of the patient and the output created earlier is also added

The email is sent, and confirmation is printed in the console.

8.4 Receiving data from AWS IoT Core

Heart rate and oxygen saturation levels are needed for a doctor to make a determination on a patient's health. The AWS IoT Core was used to transmit this data from ESP32 to the Covid Digital Doctor application. This can be used in a Node.js project by using the 'aws-iot-device-sdk' package.

The first step was creating the device. This required the certificates created and downloaded from the AWS management console.

```
var device = awsIot.device({
  keyPath: "C:\\newAWSkeys\\ac04a7f5c78064d94dab1164b35b32e0dad5d5d4b915a318aea98caa6893c899-private.pem.key",
  certPath: "C:\\newAWSkeys\\ac04a7f5c78064d94dab1164b35b32e0dad5d5d4b915a318aea98caa6893c899-certificate.pem.crt",
  caPath: "C:\\newAWSkeys\\AmazonRootCA1.pem",
  clientId: "node",
  host: "a31tdkejw9fn0n-ats.iot.us-east-1.amazonaws.com"
});
```

The file paths of the certificates are added.

A function was then created connect to the AWS IoT Core and subscribe to the topic.

```
async function getData(topic){
  device.on('connect', function() {
    console.log('connect');
    device.subscribe(topic);
  });
}
```

Once the application is connected and subscribed to the correct topic, it is now waiting for messages that are published to that topic. AWS sends the message as one document called a payload. To extract the information needed from the payload, it first must be converted to a string. Once it is in string format it can now be parsed, and the information required can be isolated.

```
device.on('message', function(topic, payload) {
  console.log('message', topic, payload.toString());
  var BPM = payload.toString().substring(
    payload.toString().lastIndexOf("BPM:") + 7,
    payload.toString().lastIndexOf(',')
  );

  var SPO2 = payload.toString().substring(
    payload.toString().lastIndexOf("SPO2:") + 8,
    payload.toString().lastIndexOf(",")
  );
  console.log(BPM);
  console.log('/');
  console.log(SPO2);
  console.log('/');
  updateBPMandSPO2(BPM, SPO2);
});
```

The program is waiting for a message from the topic.

Once a payload arrives the toString()

Method is used to convert it to a string. substring() is then used to parse the data.

The data extracted is then passed to a function.

The only remaining part of the process is to store the data. Since this project only has one source of data (one oximeter). I decided to update just one patient. There are many ways to do this, but I decided that I would update the one that has my own email just for demonstration purposes.

A function was created where the extracted BPM and SPO2 values were passed. The Mongo function findOneAndUpdate() was used to find the user with the correct email and add the oximeter readings.

```
async function updateBPMandSPO2(BPM, SPO2){
  await Patient.findOneAndUpdate({
    email: 'briansharkey07@gmail.com'
  }, {
    $push: {
      bpm: BPM,
      spo2: SPO2,
    }
  });
}
```

Patient is found by email.

Push is used to push the new values into the arrays on the patient records

9 Ethics

While researching the feasibility of this project it forced me to consider its ethical implications. Any system that operates with user information and particularly with medical information carries a great deal of responsibility. I found there are three main ethical considerations associated with a system like this: [3]

9.1 Confidentiality:

This system will hold very sensitive data about each patient including name, age, PPS, heart rate, and SPO2. This information should only be shared with others with the participants permission or if required by law. There may be situations where a patient is unable to grant permission themselves due to age or mental incapacity. In these cases, the decision should be made by legal representation or a legal guardian.

9.2 Security:

Unfortunately, it is all too common in the health industry for medical records to be breached. Therefore, security measures should be put in place. User data should be encrypted and password protected to prevent unauthorized parties from accessing the data. Policies and procedures should be implemented to maintain the security of patient records. For example, users must not share login details with anyone and must log out whenever finished using the system.

9.3 Data Inaccuracies:

Data accuracy is inherently the most important aspect of the project as it is the data that determines whether the patient should return to hospital. Therefore, incorrect data could result in the rapid deterioration of a patient's health. There are multiple factors that should be considered as a possible threat to data accuracy. Storage issues and data transfer issues could be a potential cause for concern. Users may also make mistakes and will require training on how to correctly use the Covid Digital Doctor.

10 Conclusion

In conclusion, I am extremely satisfied with how the Covid Digital Doctor has been developed. In the beginning stages of the project, I set out specific outline of what my project should be able to do. I feel that each of those main functions have been accomplished. The main features of the project that I felt were vital were:

- ✓ Create Log in / Log out / Register feature.
- ✓ Connect ESP32 to Oximeter.
- ✓ Transmit Oximeter data to application.
- ✓ Display data in user friendly format.
- ✓ Ability to notify patient via email.

I regard the final result of the project to be a strong prototype. Although I accomplished all the main features, I recognise that there will of course be areas of the application that I feel could be improved on. For example, I would add a feature that would automatically send an email when the patients' SPO2 level falls below 95 in future versions. However, I feel I have accomplished a great deal and furthered my knowledge in the area of Nodejs and AWS in particular.

11 References

- [1] Node.js, "About Node.js," [Online]. Available: <https://nodejs.org/en/about/>.
- [2] Traversy Media, "Nodemailer - Send Emails From Your Node.js App," 8 09 2017. [Online]. Available: <https://www.youtube.com/watch?v=nF9g1825mwk&t=1347s>.
- [3] "Ethical issues in electronic health records: A general overview," 6 04 2015. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4394583/>.
- [4] H. Kinsley, "Reinforcement Learning," PythonProgramming, [Online]. Available: <https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/>. [Accessed 02 02 2021].
- [5] MakeSigns, "Scientific Posters Tutorial," [Online]. Available: <https://www.makesigns.com/tutorials/scientific-poster-parts.aspx>. [Accessed 09 02 2021].
- [6] Arduino. [Online]. Available: <https://www.arduino.cc/>. [Accessed 09 02 2021].
- [7] D. Carty, "techtarget," [Online]. Available: <https://searchaws.techtarget.com/definition/Amazon-Elastic-Compute-Cloud-Amazon-EC2>.
- [8] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Pulse_oximetry.
- [9] Wikipedia, "Pulse Oximetry," [Online]. Available: https://en.wikipedia.org/wiki/Pulse_oximetry.
- [10] D. Francis, "Get started with this AWS IoT tutorial for beginners," 07 04 2020. [Online]. Available: https://searchcloudcomputing.techtarget.com/video/Get-started-with-this-AWS-IoT-tutorial-for-beginners?_gl=1*lwjfmj*_ga*MTgwMTYxMDAyMi4xNjI5OTcwNTM4*_ga_TQKE4GS5P9*

MTYyOTk3NjkyMi4zLjEuMTYyOTk3NjkzMi4w&_ga=2.107797506.1490321989.1629970538-1801610022.16299.

- [11] Wikipedia, "Bluetooth Low Energy," [Online]. Available: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy .
- [12] D. Saha, "What is MongoDB and Why to use it?," 25 05 2021. [Online]. Available: <https://www.dotnettricks.com/learn/mongodb/what-is-mongodb-and-why-to-use-it> .
- [13] Wikipedia, "ESP32," [Online]. Available: <https://en.wikipedia.org/wiki/ESP32>.
- [14] K. McCallum, "Can an Oximeter Help Detect COVID-19 at Home?," 12 08 2020. [Online]. Available: <https://www.houstonmethodist.org/blog/articles/2020/aug/can-an-oximeter-help-detect-covid-19-at-home/>.
- [15] fda.gov, "Non-contact Temperature Assessment Devices During the COVID-19 Pandemic," [Online]. Available: <https://www.fda.gov/medical-devices/coronavirus-covid-19-and-medical-devices/non-contact-temperature-assessment-devices-during-covid-19-pandemic>.
- [16] W3schools, "HTML Forms," [Online]. Available: https://www.w3schools.com/html/html_forms.asp.
- [17] mongodb, "What Is MongoDB?," [Online]. Available: <https://www.mongodb.com/what-is-mongodb>.
- [18] S. Jaffry, "Best practices for securing sensitive data in AWS data stores," 24 12 2018. [Online]. Available: <https://aws.amazon.com/blogs/database/best-practices-for-securing-sensitive-data-in-aws-data-stores/>.
- [19] Academind, "NodeJS / Express / MongoDB - Build a Shopping Cart - #1 Intro & Setup," 16 05 2016. [Online]. Available: <https://www.youtube.com/watch?v=-3vwxn78MH4>.

- [20] Things School, "Connecting ESP32 to AWS IoT Core," 08 09 2020. [Online]. Available: <https://www.youtube.com/watch?v=eZazePshMig&t=1622s>.
- [21] K. Adcock, "How to make an academic poster in powerpoint," 28 11 2018. [Online]. Available: https://www.youtube.com/watch?v=_WnholbfcoM.
- [22] AWS, "What is Amazon EC2?," [Online]. Available: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>.
- [23] AWS, "What is AWS IoT?," [Online]. Available: <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>.