Team 25 Final Report
Home Sensor Dashboard

Members :
Fin Cottle – 1938561
Craig Thompson – 2046039
David Priehoda - 2048076
William Beazley - 21011673
Yanqiao – 21115418
Wing - 2042783

Client: Alia Abdelmoty - abdelmotyai@cardiff.ac.uk
***Supervisor*:** Xin Zhao - ZhaoX30@cardiff.ac.uk

Contents

Introduction

Potential Persona A

Name: Celine Potter

Age: 29

Sex: Female

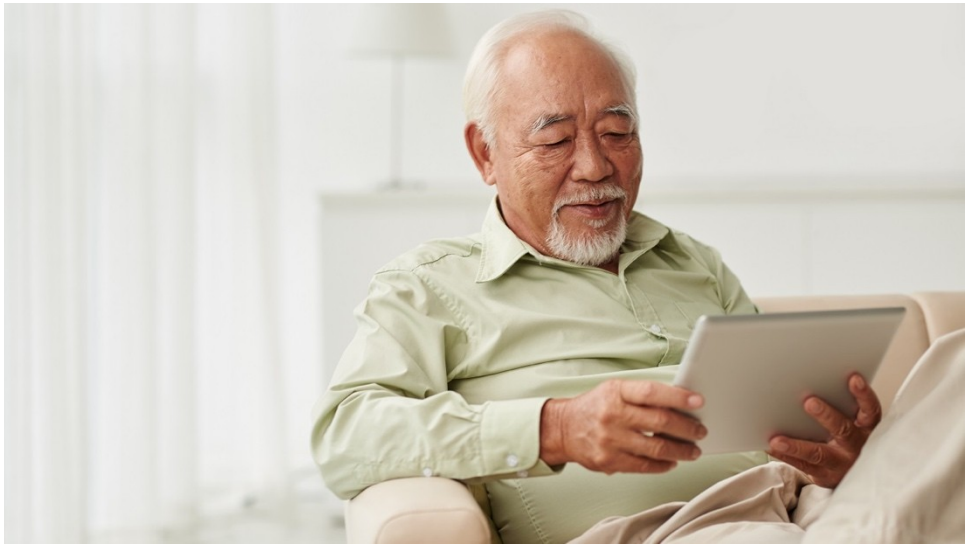
Occupation: Professional carer

Celine loves her job so much, but it is too busy for her to manage so many clients at a time. Due to carer shortage, one carer needs to take care of at least 10 elderly people, but it is hard to visit each elderly every day. She is trying her best to visit as many elderly people as possible in a day to ensure the elderly's living conditions are comfortable. Each time she visits an elderly person, she will make sure the light bulb is working normally, the house is not hot and stuffy, and needs to remind the elderly to open the windows to ensure the circulation of the air.

Goals:

- A system to monitor temperature in the house
- A system can monitor every corner of the house
- A system that can connect with the clients
- View the state of clients homes remotely

Potential Persona B



Name: Alex Champion

Age: 64

Sex: Male

Occupation: Retired

Alex has hired a carer to take care of him, but his carer only visits him three times a week. Most of the time he lives alone and needs to look after himself. As he ages, he becomes sensitive to cold temperatures because of a decrease in the metabolic rate. It would be great if he could get alert when the temperature drops so he can get ready to wear more clothes and it will be much better if the heater turns on automatically so Alex can avoid getting mild hypothermia at night. Although he is open-minded to modern technologies, he takes a long time to learn to use an application now.

Goals:

- A system that sends an alert to the user
- A system that is easy to use
- A system that can monitor the temperature of the house

Changes we made from the interim report

The majority of our implementation follows along closely with our requirements in the interim report. However, due to some technical limitations, we have made 4 changes to the implementation plan during our development process.

The first one is functional requirement 8, which is ‘System will pre-process data on input to remove erroneous data’. After some study and research, we found that the probability of a short time error in the sensor is very small. Either the sensor is faulty and all the data is wrong, then we should inform the user and suggest them to replace the sensor instead of automatically correcting the wrong data, or the environment is abnormal and we should inform the user immediately instead of hiding the error. In addition, it is difficult to tell whether there is an error in the sensor data or an anomaly in the environment, so we should leave it to the user to tell, rather than letting the system choose automatically. After careful consideration, we have removed this feature from the system.

The second one is functional requirement 9, which is ‘System could be able to interact with third party systems. Since our system is only a test version and has no actual sensor data input. Due to practical and technical constraints, we are not able to find a third-party system suitable for data interaction with our system at this time. As a result, we are giving up this feature in our test version system. However, we will continue to develop this once we have successfully deployed the system and received real sensor data input.

The third one is functional requirement 10, which is ‘System could predict when to turn on and off appliances depending on the time. ‘Due to time constraints, we are not able to generate large amounts of data that match the real environment in a short period of time and predict user habits based on the data. In addition, as we didn't interact with real sensors, it’s hard for us to test the functionality of the turn sensors on/off. Predicting user habits involves machine learning techniques, which we don't have yet, but we will continue to dig deeper in this area in the future.

The last one is functional requirement 2, which is ‘Uses could view different summaries of the data provided by the sensor’. Instead of just providing data summaries such as minimum and maximum, we have made some improvements, providing the exact time when these data occur, giving the user a more detailed view of the environmental data.

How did these changes impact our work?

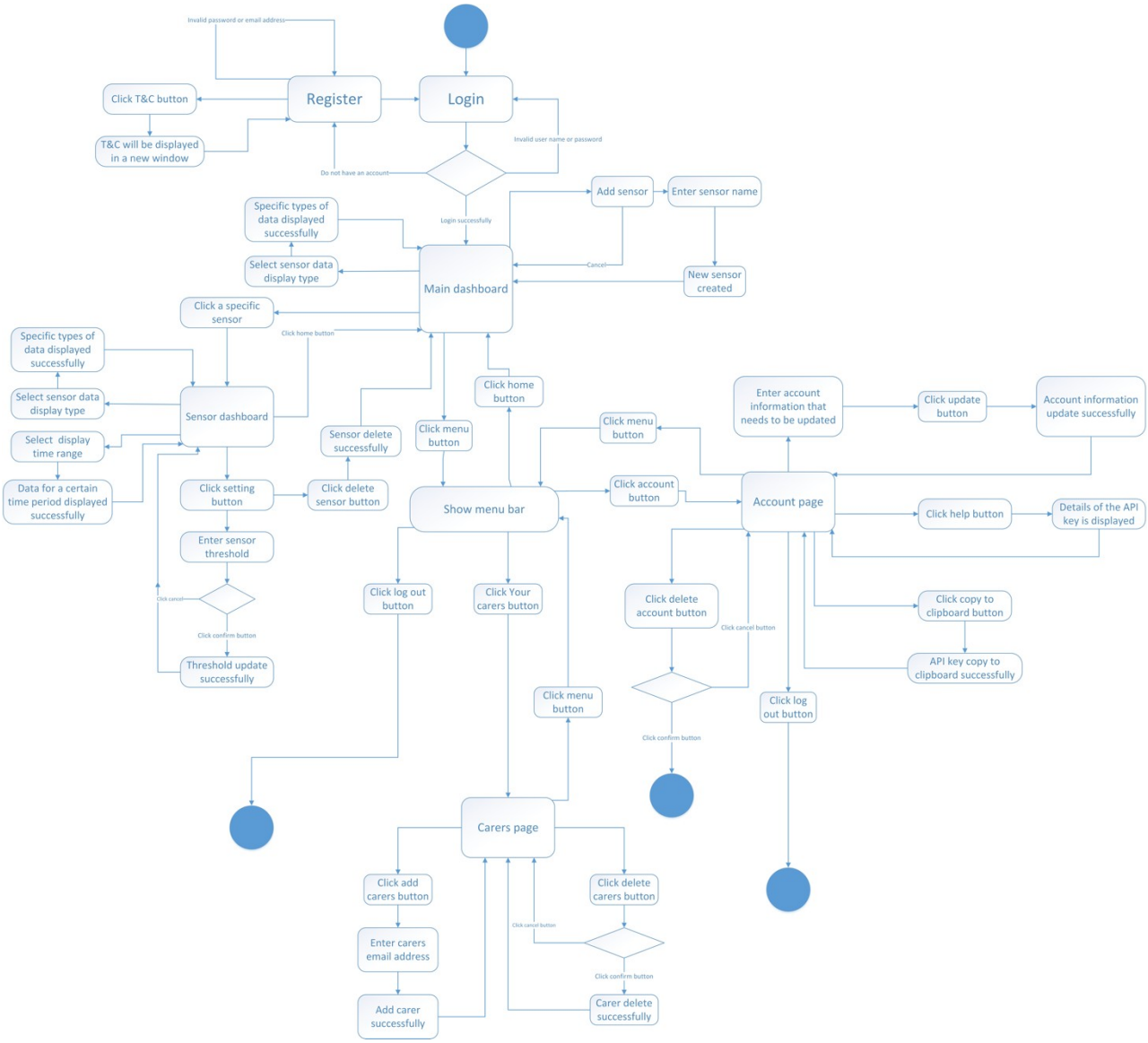
The changes we made do not impact the final system much as they are not major features. These changes have made us aware that the real development process is subject to a variety of realities and technical constraints and that we must constantly revise and improve our development plans. We also had a follow-up meeting with our client to demonstrate our demo system. Our client suggested that we should focus more on information, how it's presented, and what kind of information we should present to our users. To achieve this, we have improved the data summary module by putting in more detailed information (e.g., The highest and lowest point of the data and when they happen), we have also enlarged the buttons to make it easier for users to operate them.

Explaining the solution

The initial task we were given by the client was to develop a web-based application that takes the data that is streamed from sensors in a property as input and provides an interface for the users to visualize and analyse the data from the different sensors in different rooms in a property. The solution we made was a web-based sensor dashboard that is responsive to both desktops and mobile devices. Since we are given sample data instead of real-time input data, we wrote sensor.py to read data from the sample and then commit them to our database to simulate real-time input data. To use the system, users first need to register an account and log in to it, each account has a unique API key, which can be seen on the account page. Then users can use the 'add' button in the top right corner to create new sensors, each sensor has a unique Sensor ID, which can be seen by clicking a specific sensor. After running sensor.py and filling in the account API key, Sensor ID and data file name, the sample data will be committed to the database and retrieved by the sensor, then the data of the sensor as well as the data summaries (maximum, minimum, average, latest) will be shown. This may seem a bit complicated, but it won't be a problem because, in the real world, users can get the data directly from the sensor without having to enter it manually. After passing in the data, users can define a threshold for each sensor so that the system will remind users when data values are out of the threshold. Furthermore, users can also add a carer in the carers page to share their sensor data with the carer. To protect user privacy, users can delete all sensor and account data at any time.

Work on the project since the interim report

Activity diagram for the system



Description of final system architecture and design

The main system architecture contains four main subsystems.

The first of which is the account system, which enables users to log in, register, modify their account details and save sensor data. The system has three main front-end .html pages (login.html, register.html, account.html). Back-end logic code for the system (e.g., User authentication, database operations, etc.) is in routes.py. A database model User is used to save users' data to the database, the model includes user id, first name, last name, email, password (hashed for security), carer id, and API key. Flask forms (Login form, register form, Account update form, Add carer form and Sensor limit form) are used to connect front-end .html pages to back-end code logic.

The second subsystem is the sensor system. This system includes the sensor addition and deletion function, importing sensor data, saving, displaying and making a summary of the data. The system has two main front-end .html pages (home.html and sensor.html). As we do not have access to the sensor data for

the implementation input at the moment, we use sensor.py to import the sample data to stimulate data input. Data is sent to our flask app using HTTP POST requests by codes written in routes.py. We will first save the data into our database, then sensors can retrieve and display the graphs using Chart.js JavaScript library. A database model SensorEntry is used to save sensors' data to the database. The model includes sensors' id, data entry time, and environmental data such as light, humidity, noise, etc. Another database model Sensor is used to link sensors to users' accounts as well as save the sensor limits set by users. The model includes sensor id, user id, sensor name and other upper and lower boundaries of environment data.

The third subsystem is the notification system. This system is built based on the sensor system. It's for users to define a threshold so that if the readings of the sensor are outside the threshold, the system can alert the user. The front-end code of this system is in sensor.html. A window for setting the threshold will pop up when users click the set button in sensor.html. Back-end logic code for the system (e.g., update limit, get sensor limit) is in routes.py. A Flask form SensorLimitForm is used to connect front-end .html pages to back-end code logic. The threshold set by users will be committed and stored in the database model User. The system can retrieve the threshold set by users from the database, making graph points outside the user limits colored differently.

The fourth subsystem is the carer system. This system allows users to add a carer, who can have access to users' sensor data. The system has three main front-end .html pages (carers.html, patient.html, patients.html). In carers.html, users can add or delete a carer for their account. Patients.html is the page where the carer can see all of their patients and then select which one they want to view, patient.html shows all the graphs for a specific patient. Back-end logic code for the system (e.g., add carer, delete carer, database operations, etc.) is in routes.py. A flask form AddcarerForm is created to connect front-end .html pages to back-end code logic. A database model UserCarer is used to save carers' information, the model includes carer id, user id and carer since.

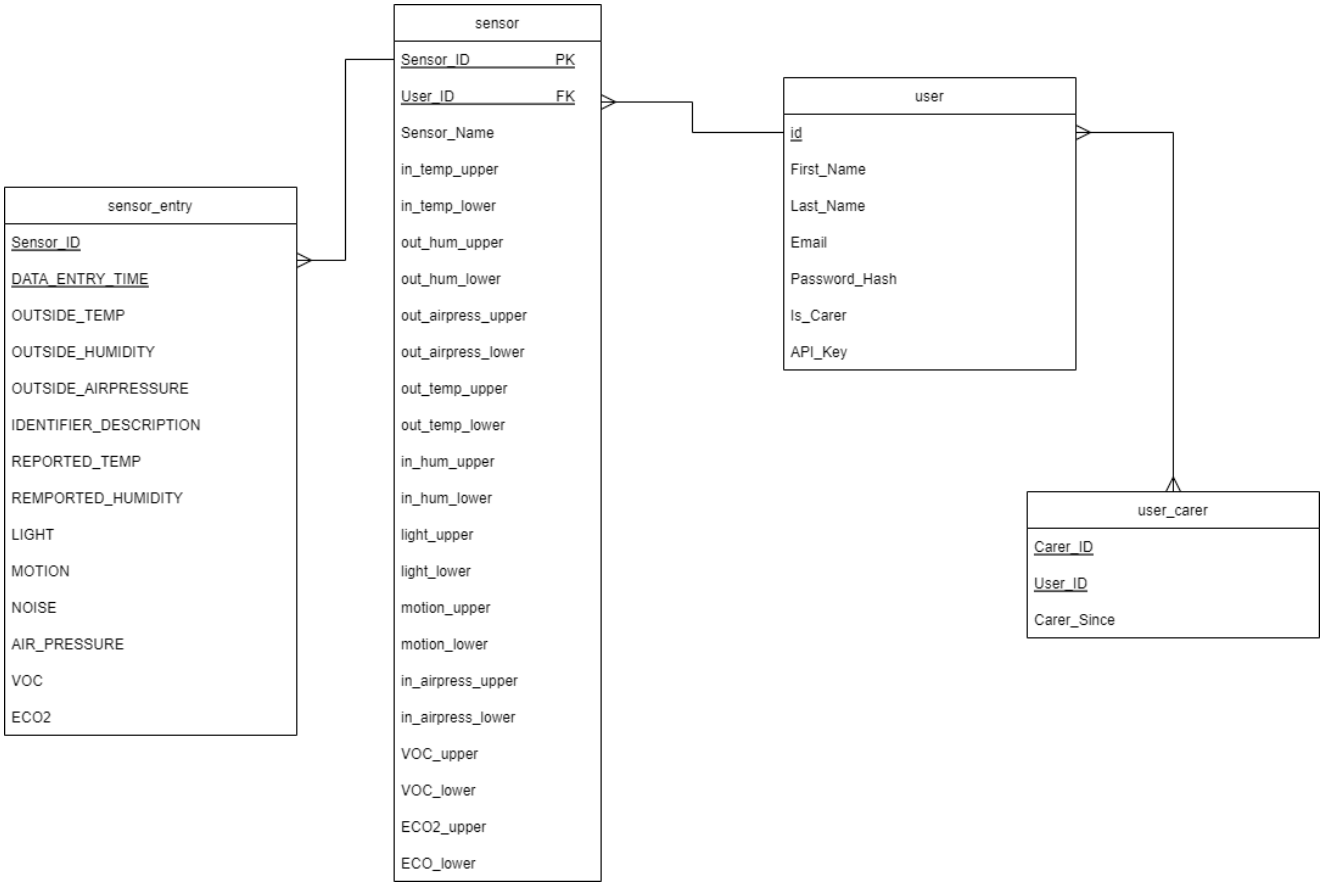
There are also other front-end .html pages in our system (e.g., navibar.html, PP.html, T&C.html and welcome.html). The navigation bar is included in home.html, account.html and carers.html, it can be displayed by clicking on the button in the top left corner, making it easy for the user to switch between the three pages. PP.html and T&C.html include our privacy policy and terms and conditions. Welcome.html, it is a transition screen from the login.html to the home.html.

As for CSS files, styles set for navbar are stored in navbar.css, while styles for all other pages are stored in style.css.

Database design (static behaviour)

Here is the design of the database, storing all the sensor data, as well as the users account.

The keys are underlined, and the relationships are denoted with lines, also referring to the number of instances of each, for example, “One user has many sensors”.



The sensor table also now (compared to previous versions of the database design) stores the user-defined limits for the system, which can be updated by the user via the sensor page.

Core implementation

Main dashboard

For our final solution, we designed a web application making use of HTML, JavaScript and python languages with the CSS stylesheets for our design. In the future we would like to expand this to a Mobile application, but as for now our solution is of high quality and works as a web application on both Mobile and Desktop.

The CSS for our solution contains is stored within two static files, styles.css and navbar.css. These two files provide the layout and design of our solution. Snippets of both can be seen below. We chose to go with background-color #729CE9 which is a light blue coloring as it provides a clean and sophisticated look that also pairs well with the sensor readings when they are displayed on the home dashboard page. As well as padding, margin and overflow control within the body. Both are then extended to every page used in the system.

```css
#menu-open-btn{
    font-size: 60px;}

#menu-close-btn{
    float: right;
    font-size: 40px;
    color: #f2fee4;
    padding-top: 0px;
    position: relative;}

#sidenav{
    width: 20vw;
    height: 100%;
    position: fixed;
    top: 0;
    left: 0;
    background-color: #293c48;
    transform: translate3d(-100vw,0,0);
    transition: transform .3s cubic-bezier(0, .52, 0, 1);
    overflow: hidden;
    z-index: 1;}

#sidenav.show{
    transform: translate3d(0vw,0,0);}

#sidenav ul{
    margin-top: 5vh;}

#sidenav ul li{
    list-style:none;
    left: 25%;}

#sidenav ul a{
    display: block;
    text-decoration: none;
    font-size: 32px;
    color: #f2fee4;}

#account-btn{
    position: absolute;
    bottom: 50px;}

#logout-btn{
    position: absolute;
    bottom: 5px;}
```

```css
body {
  background-color: #729CE9 !important;
  margin: 10px;
  padding: 0;
  color: #000;
  overflow-x: hidden;
}
```

After the user is verified and logged in, the first screen they will see is the home screen. As you can see below there are sensor readings with descriptions with possible flows on either side on the screen to navigate the system (Left) or add a new sensor (Right).

On this home screen each sensor will provide the user with a snapshot of data from the past 24 hours. With the added capability of choosing which unit of measurement they would like to see. Some examples are “Temperature", "Humidity”, “Light” and “Air pressure as seen in the screenshot below.

Below are two snippets of code from the home.html file. This is where all the code lies for displaying the graph data as well as utilising JavaScript functions in order to add a new sensor to the dashboard. When loading sensor data the system loops through every sensor that is located in the user’s sensor database and produce charts using JavaScript capable of switching between different measures of measurements as seen in the example above.

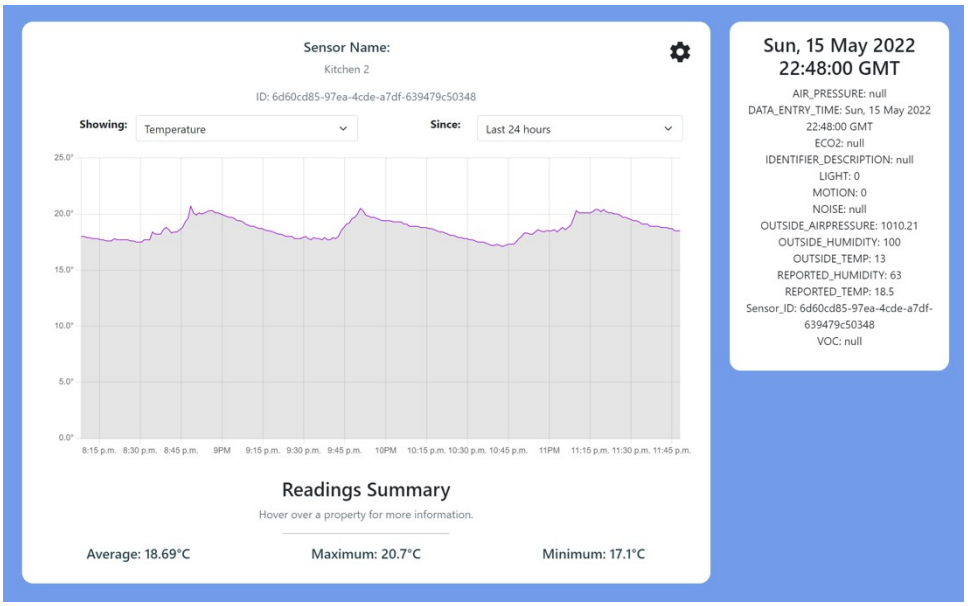
```
{% block head %}
<script type="text/javascript">
    function go_to_sensor(sensor_id){
        window.location.href = '/sensor/'+sensor_id;
    }

    $(document).on('submit','#sensor-form',function(e){
        e.preventDefault();

        if (document.getElementById("sensor-name-field").value == ""){
            alert("Please enter a sensor name");
            return;
        }

        //CSRF Protection
        var csrf_token = "{{ csrf_token() }}";
        $.ajaxSetup({
            beforeSend: function(xhr, settings) {
                if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !this.crossDomain) {
                    xhr.setRequestHeader("X-CSRFToken", csrf_token);
                }
            }
        });

        $.ajax({
            type:'POST',
            url:'/newsensor',
            data:{sensor_name:$("#sensor-name-field").val()},

            success:function(){
              alert('New sensor created');
            }
        })
    });
</script>
{% endblock head %}
```

```
<script>
    {% for sensor in sensors %}

        $.get("/get_sensor_data?sensor_id="+ '{{ sensor.get_id() }}' +"&data_type="+ "0" +"&time_option="+ "0", function(data){
                //{'Sensor_ID': 'ID', 'Summary':{'Avg': 0, 'Min': 0, 'Max': 0}, 'Data': [[timestamp, value], ...]}

                make_chart(data.Sensor_ID, data.Data, 0);
        });
        //Create drop down menu
        var options = ["Temperature", "Outside humidity", "Outside air pressure", "Outside temperature", "Reported humidity", "Light", "Motion", "Noise", "Air
        var dropdown = document.getElementById("dropdown {{ sensor.get_id()}}");

        for (var option in options) {
            var el = document.createElement("option");
            el.textContent = options[option];
            el.value = option;
            dropdown.appendChild(el);
        }

        dropdown.onchange = function() {
            $.get("/get_sensor_data?sensor_id="+ '{{ sensor.get_id() }}' +"&data_type="+ this.value +"&time_option="+ "0", function(data){
                make_chart(data.Sensor_ID, data.Data, this.value);
            });
        };
    {% endfor %}



    function make_chart(sensor_id, chart_data, data_type){
        $('#'+sensor_id).replaceWith($('<canvas id="'+sensor_id+'"></canvas>'));
        var ctx = document.getElementById(sensor_id);

        var data = [];
        for (var i = 0; i < chart_data.length; i++) {
            data.push({x: new Date(chart_data[i][0]).getTime(), y: parseFloat(chart_data[i][1])});
        }


        //Make chart
        var chart = new Chart(ctx, {
            type: "line",
            data: {
            datasets: [{
                data,
                indexAxis: "x",
                normalized: true,
                borderColor: "darkviolet",
                pointBackgroundColor: "darkviolet",
                borderWidth: 1.5,
                pointRadius: 0
                }
            ]
            },

            options: {
                layout: {padding: 20},
                resizeDelay: 250,
                responsive: true,
                animation: false,
                parsing: false,
```

Overall in the system there are 4 databases holding user data. These are the “User” database which stores all user data, “UserCarer” which stores information if the user is a carer, “SensorEntry” and “Sensor” Which holds sensor information separate from the entries.

Sensor page

The sensor page is the second most important page of the entire system, this is where the user can view the sensor data from a specific sensor. Here the user can select from two drop down boxes to change the timeframe of data that they want to be down, as well as the measurement form like “Temperature or humidity”. The user also has access to the settings feature where they can set higher and lower threshold boundaries if they wish to be notified when a sensor reading goes out of bounds.



Below is the sensor database where all data is saved to. Here is where all data gets stored and retrieved. As well as getting ranges for alert system.

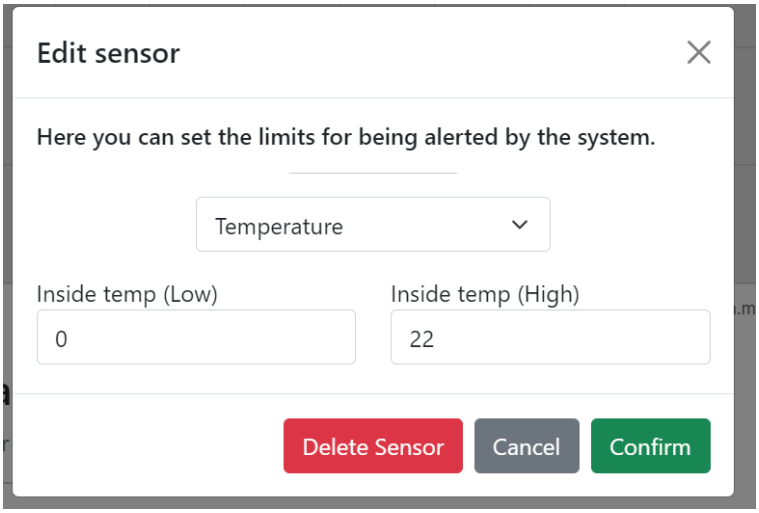
```python
class Sensor(db.Model):
    Sensor_ID =  db.Column(db.String(36), default=str(uuid4()), primary_key=True)
    User_ID = db.Column(db.String(36), nullable=False)
    Sensor_Name = db.Column(db.String(32), default="Sensor")
    #Limits
    in_temp_upper = db.Column(db.Integer, default=27, nullable=False)
    in_temp_lower = db.Column(db.Integer, default=0, nullable=False)
    out_hum_upper = db.Column(db.Integer, default=50, nullable=False)
    out_hum_lower = db.Column(db.Integer, default=30, nullable=False)
    out_airpress_upper = db.Column(db.Integer, default=1050, nullable=False)
    out_airpress_lower = db.Column(db.Integer, default=950, nullable=False)
    out_temp_upper = db.Column(db.Integer, default=27, nullable=False)
    out_temp_lower = db.Column(db.Integer, default=0, nullable=False)
    in_hum_upper = db.Column(db.Integer, default=50, nullable=False)
    in_hum_lower = db.Column(db.Integer, default=30, nullable=False)
    light_upper = db.Column(db.Integer, default=300, nullable=False)
    light_lower = db.Column(db.Integer, default=0, nullable=False)
    motion_upper = db.Column(db.Integer, default=70, nullable=False)
    motion_lower = db.Column(db.Integer, default=0, nullable=False)
    noise_upper = db.Column(db.Integer, default=85, nullable=False)
    noise_lower = db.Column(db.Integer, default=0, nullable=False)
    in_airpress_upper = db.Column(db.Integer, default=1050, nullable=False)
    in_airpress_lower = db.Column(db.Integer, default=950, nullable=False)
    VOC_upper = db.Column(db.Integer, default=3000, nullable=False)
    VOC_lower = db.Column(db.Integer, default=0, nullable=False)
    ECO2_upper = db.Column(db.Integer, default=2000, nullable=False)
    ECO2_lower = db.Column(db.Integer, default=0, nullable=False)

    def get_id(self):
        return self.Sensor_ID

    def get_name(self):
        return self.Sensor_Name

    def get_entries_between(self, start_time, end_time=datetime.now()):
        #Return all sensors entries from a given time period
        return SensorEntry.query.filter_by(Sensor_ID=self.Sensor_ID).filter(SensorEntry.DATA_ENTRY_TIME >= start_time, SensorEnt

    def get_data_between(self, data_type, start_time, end_time=datetime.now):
        entries = self.get_entries_between(start_time, end_time)
        data = []

        if len(entries) > 0 and hasattr(entries[0], data_type):
            for entry in entries:
                data.append([entry.DATA_ENTRY_TIME, str(getattr(entry,data_type))])

        return data

    def get_limit(self, id):
        #id = 'in_temp_upper'
        return getattr(self, id)

    def delete_sensor(self):
        db.session.query(SensorEntry).filter_by(Sensor_ID=self.Sensor_ID).delete()
        db.session.delete(self)
        db.session.commit()
        return True
```

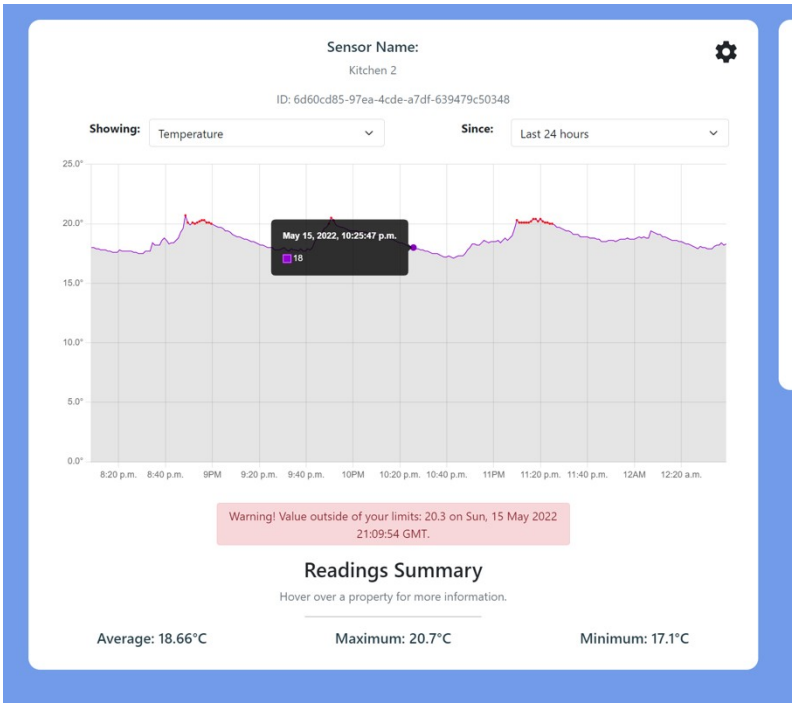
Just like the main dashboard, this screen also utilises JavaScript in the same way, except here it only produces 1 main graph where all the data can be displayed. However, in this page you can view more Indepth data like the most recent data entered into the system on the right of the page, as well as summary readings at the bottom of the page. The HTML Code outlining the summary readings can be seen below.

```javascript
        if (Object.keys(summary).length == 5){
            var max = document.getElementById('max');
            var min = document.getElementById('min');
            // If readings are temperatures add degrees C
            document.getElementById("no-data").innerHTML = "";
            if (data_type_dropdown.value == 0 || data_type_dropdown.value == 3) {
                //document.getElementById('latest').innerHTML = "Latest: " + data[data.length - 1][0] + "°C";;
                max.innerHTML = "Maximum: " + summary.Max + "°C";
                max.setAttribute("data-bs-original-title", summary.MaxDateTime);

                min.innerHTML = "Minimum: " + summary.Min + "°C";
                min.setAttribute("data-bs-original-title", summary.MinDateTime);
                document.getElementById('avg').innerHTML = "Average: " + summary.Avg + "°C";
            } else{
                max.innerHTML = "Maximum: " + summary.Max;
                max.setAttribute("data-bs-original-title", summary.MaxDateTime);

                min.innerHTML = "Minimum: " + summary.Min;
                min.setAttribute("data-bs-original-title", summary.MinDateTime);
                document.getElementById("avg").innerHTML = "Average: " + summary.Avg;
            }
        }

        else{ // if no data is available
            document.getElementById("no-data").innerHTML = "No readings found";
            document.getElementById("latest").innerHTML = "";
            document.getElementById("max").innerHTML = "";
            document.getElementById("min").innerHTML = "";
            document.getElementById("avg").innerHTML = "";
        }
    }
```

Alert system

The system also comes equipped with an alert system. This will notify the user when the set threshold boundaries go out of bound either higher or lower. The user can set these limits by clicking on the settings icon and entering the higher or lower limits. They can also select which measuring system they want to limit. Examples of the inside design can be seen below as well as an example of the warning message alerting the user





The coding for this system was done in the HTML file using a combination of JavaScript and HTML coding language. Snippets of the code can be seen below

```
<!-- Modal -->
<div class="modal fade" id="sensor_limit_modal" tabindex="-1" aria-labelledby="label" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="modal-label">Edit sensor</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <form method="POST" id="limit-form">
        <div class="modal-body">
          <h6>Here you can set the limits for being alerted by the system.</h6>
          <hr class="mx-auto w-25">
          <input type="hidden" name="csrf_token" value="{{ csrf_token() }}"/>
          <div class="row mb-3">
            <select id="limit-type" class="form-select w-50 mx-auto" name="limit-type"></select>
          </div>
          <div id="limits_parent" class="row row-cols-2">
            {% for f in form %}
            {% if f.id != 'csrf_token':%}
            <div class="col mb-1 d-none" id="{{f.id}}_div">
              {{f.label}}
              {{f(value=sensor.get_limit(f.id))}}
            </div>
            {% endif %}
            {% endfor %}
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-danger" onclick="deleteSensor();">Delete Sensor</button>
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
          <input type="submit" class="btn btn-success" value="Confirm">
        </div>
      </form>
      <div id="message_success" class="d-none alert alert-success w-50 py-1 mx-auto">
        Updated limits successfully!
      </div>
    </div>
  </div>
</div>
```

```
function notify_user(dataset, value){
  var msg = document.getElementById('message_warning')
  var date = new Date(value.x - (60*60*1000)).toUTCString();
  msg.innerHTML = "Warning! Value outside of your limits: " + value.y + " on " + date + ".";
  msg.classList.remove("d-none")
}
```

Login & Register

In order to implement the login and register function, login.html and register.html were added to the ‘template’ dictionary. These two files contain the UI shown on the running pages. The login and register system use two different forms in order for them to work:

```
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[InputRequired(), Email()])
    password = PasswordField('Password', validators=[InputRequired()])
    remember = BooleanField('Remember me')
    submit = SubmitField('Submit', render_kw={'class':'btn btn-primary btn-lg w-100'})


class RegisterForm(FlaskForm):
    first_name = StringField("First Name", validators=[InputRequired(), Length(max=30, message="First name cannot be longer than 30 characters.")])
    last_name = StringField("Last Name", validators=[InputRequired(), Length(max=30, message="Last name cannot be longer than 30 characters.")])
    email = StringField('Email', validators=[InputRequired(), Email()])
    password = PasswordField('Password', validators=[InputRequired(), Regexp('((?=.*[a-z])(?=.*[A-Z])(?=.*[\W]).{8,64})', message="Password must be between 8-64 characters and contain
    confirm = PasswordField('Confirm password', validators=[InputRequired(), EqualTo('password', message="Confirm Password value must be equal to password")])
    tick = BooleanField("I accept TOS", validators=[InputRequired()], render_kw={'id':'tos-check', 'class':'form-check-input'})
    submit = SubmitField('Submit', render_kw={'class':'btn btn-primary btn-lg w-100'})
```

These forms contain different classes users need to input when they want to log in or register. Each form can determine what inputs are acceptable (e.g., InputRequired(), Email(), length(), EqualTo,etc) and a submit button.

The code used to connect the database, verify the user's identity and detect incorrectly entered was written in routes.py as follows.

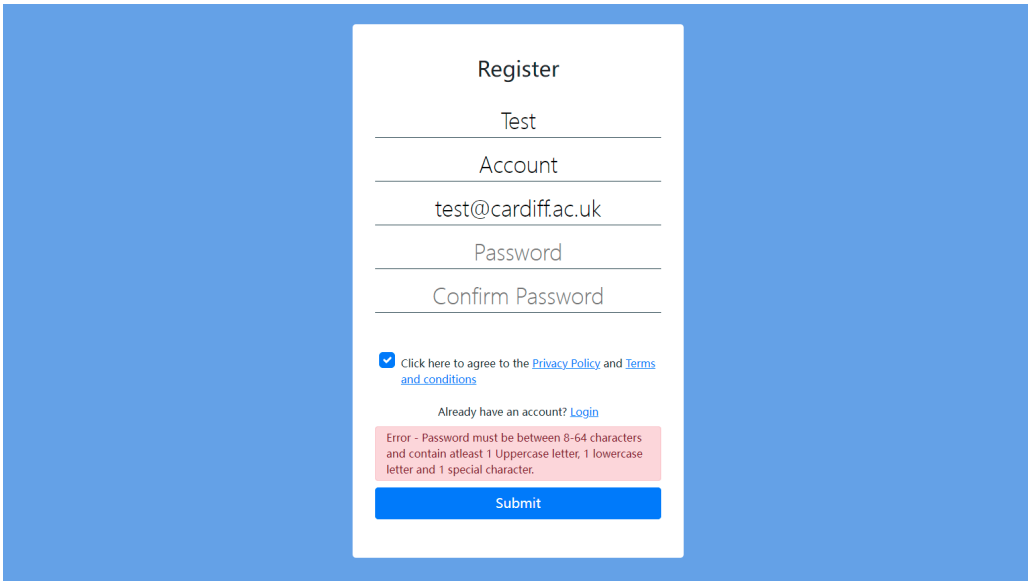
The register function

```python
@app.route("/register", methods=["GET", "POST"])
def register():
    UserRegister = RegisterForm()
    if request.method == 'POST':
        form = RegisterForm(request.form)
        if not form.validate():
            for field, error in form.errors.items():
                flash(f"Error - {error[0]}")
        else:
            first_name =  request.form.get('first_name')
            last_name =  request.form.get('last_name')
            email = request.form.get('email')
            password = request.form.get('password')
            password2 = request.form.get('confirm')
            test_str = re.search(r"\W", password)
            if User.query.filter_by(Email=email).all():
                flash('Email already in use!')
                render_template('register.html', title='Register', form=UserRegister)
            else:
                new_user = User(First_Name = first_name, Last_Name = last_name, Email = email, Password_Hash = generate_password_hash(password))
                db.session.add(new_user)
                db.session.commit()
            return render_template('login.html', form=UserRegister)
    return render_template('register.html', form=UserRegister)
```

The login function

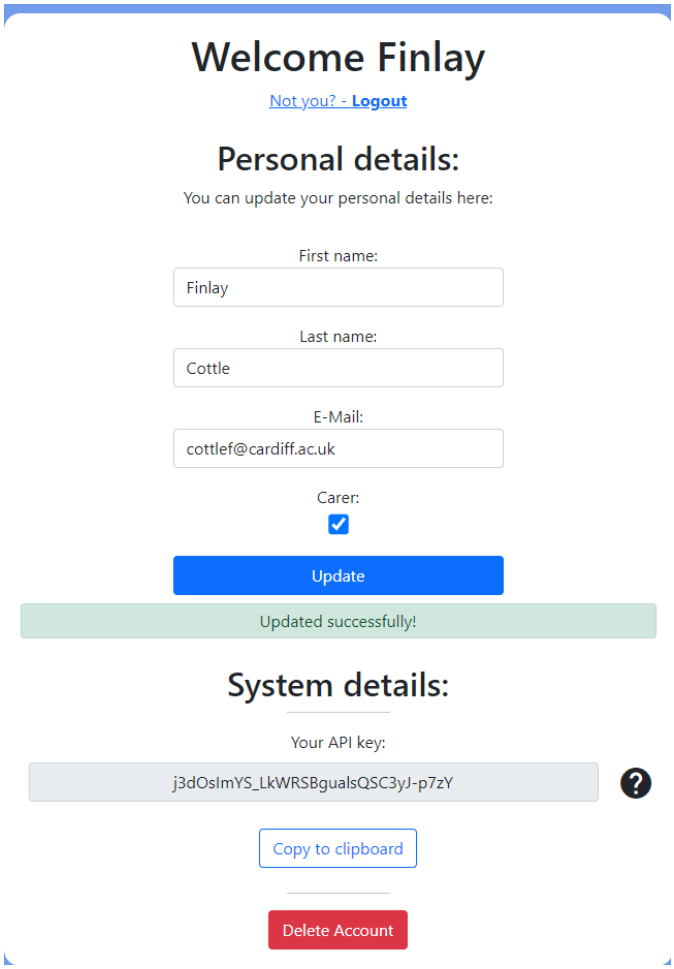
```python
@app.route("/login", methods=["GET", "POST"])
def login():
    UserLogin = LoginForm()
    if request.method == 'POST':
        UserLogin = LoginForm(request.form)
        if not UserLogin.validate():
            for field, error in UserLogin.errors.items():
                flash(f"Error - {error[0]}")
        else:
            email = request.form.get('email')
            password = request.form.get('password')
            user = User.query.filter_by(Email=email).first()
            if user is not None and user.verify_password(password):
                login_user(user)
                return render_template("welcome.html", name=user.get_first_name())

            flash('Invalid Username or Password')

    return render_template('login.html', title='Login', form=UserLogin)
```

In order to register, the password has at least eight digits and contains at least one special character and matches with the confirm password, the email has to follow a traditional format. Users should also click a button to agree to the Privacy Policy and Terms and conditions before they register. In order to test the validation function, we can input 12345678 as our password, the system will flash an error message as follows.



It’s worth mentioning that the password has been hashed so that no one can access it directly from the database.

Account page



```
    def delete_user(self):
        #Delete all of users sensors
        for sensor in self.get_sensors():
            sensor.delete_sensor()

        db.session.delete(self)
        db.session.commit()
        return True
```

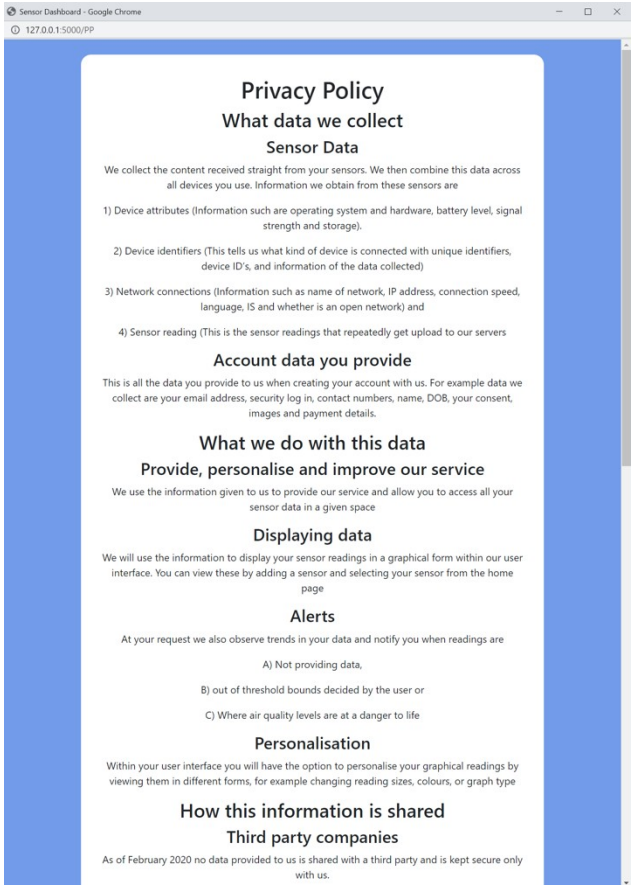
The main functionality of the account page is derived from the functional requirement “Users can delete all personal and sensor data relevant to them if they chose”. This is where the user (if they wish) can remove all traces of their personal and sensor data from the system by using the “Delete account” button. After clicking this a notification will popup asking the user to confirm like seen above that all data relevant to them will be removed from the system. This is also stated as a functionality in our privacy policy which can be seen below.

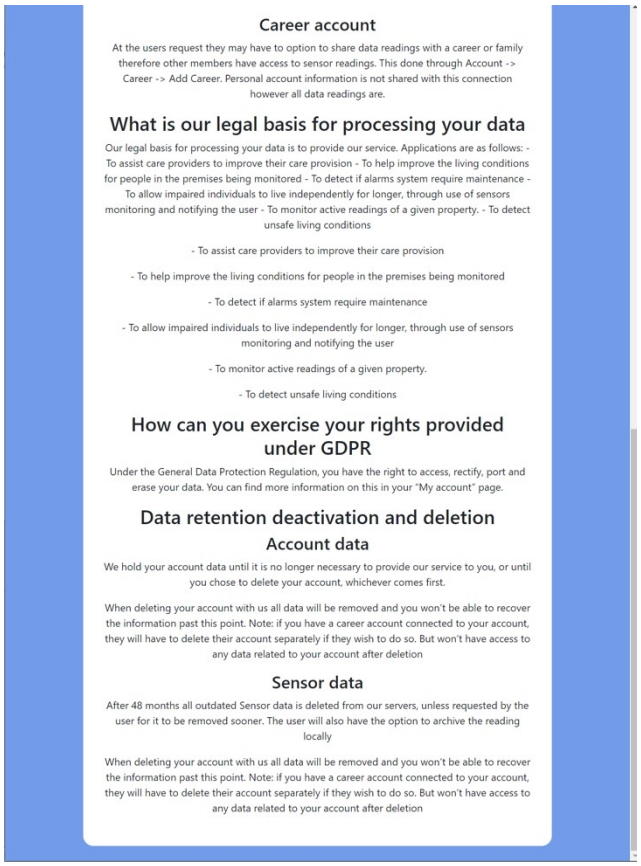
```
    return render_template("account.html", user=user)
```

Additionally, to the account page the user is also able to update and view any personal information stored regarding them and update it accordingly. This is following UK GDPR guidelines that the user has the right to access, rectify, port and erase their data.

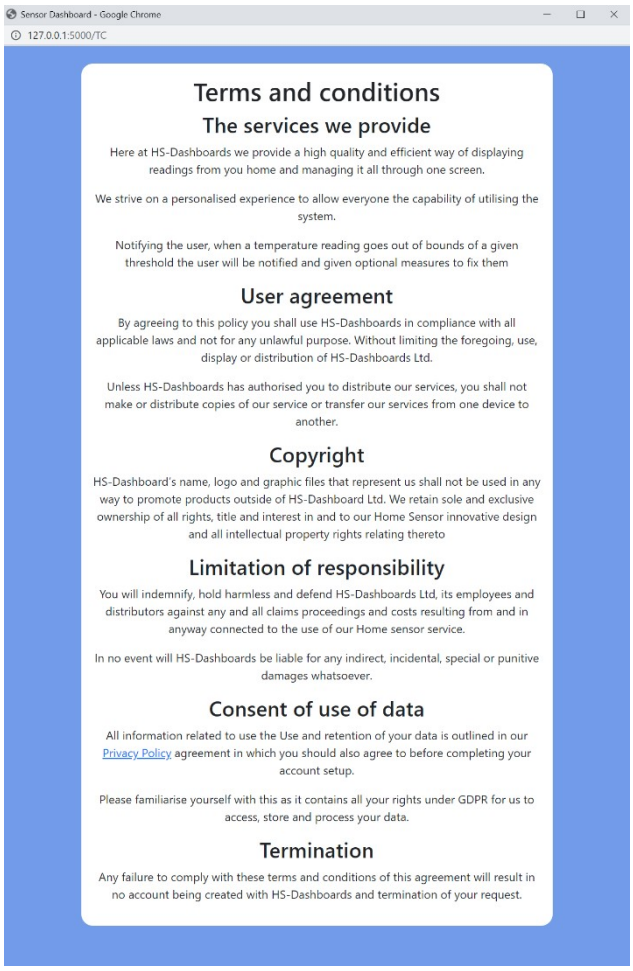
Extra functionality on this page includes repeated Logout feature, and the ability to copy your unique API key to your clipboard for further use while sorting sensor data.

Privacy

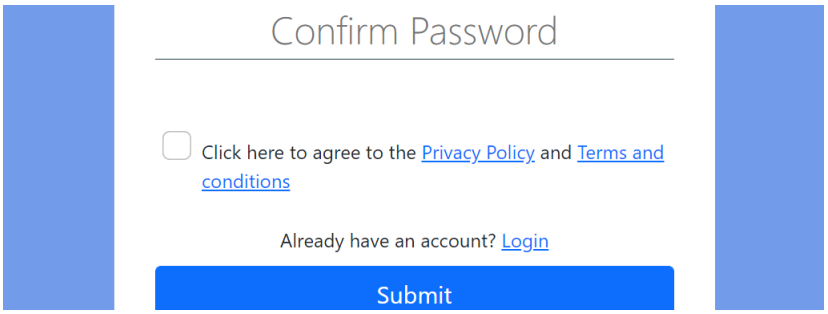

Sensor Dashboard - Google Chrome
127.0.0.1:5000/PP
Privacy Policy
What data we collect
Sensor Data
We collect the content received straight from your sensors. We then combine this data across all devices you use. Information we obtain from these sensors are
1) Device attributes (Information such are operating system and hardware, battery level, signal strength and storage).
2) Device identifiers (This tells us what kind of device is connected with unique identifiers, device ID's, and information of the data collected)
3) Network connections (Information such as name of network, IP address, connection speed, language, IS and whether is an open network) and
4) Sensor reading (This is the sensor readings that repeatedly get upload to our servers
Account data you provide
This is all the data you provide to us when creating your account with us. For example data we collect are your email address, security log in, contact numbers, name, DOB, your consent, images and payment details.
What we do with this data
Provide, personalise and improve our service
We use the information given to us to provide our service and allow you to access all your sensor data in a given space
Displaying data
We will use the information to display your sensor readings in a graphical form within our user interface. You can view these by adding a sensor and selecting your sensor from the home page
Alerts
At your request we also observe trends in your data and notify you when readings are
A) Not providing data,
B) out of threshold bounds decided by the user or
C) Where air quality levels are at a danger to life
Personalisation
Within your user interface you will have the option to personalise your graphical readings by viewing them in different forms, for example changing reading sizes, colours, or graph type
How this information is shared
Third party companies
As of February 2020 no data provided to us is shared with a third party and is kept secure only with us.


Career account
At the users request they may have to option to share data readings with a career or family therefore other members have access to sensor readings. This done through Account -> Career -> Add Career. Personal account information is not shared with this connection however all data readings are.
What is our legal basis for processing your data
Our legal basis for processing your data is to provide our service. Applications are as follows: - To assist care providers to improve their care provision - To help improve the living conditions for people in the premises being monitored - To detect if alarms system require maintenance - To allow impaired individuals to live independently for longer, through use of sensors monitoring and notifying the user - To monitor active readings of a given property. - To detect unsafe living conditions
- To assist care providers to improve their care provision
- To help improve the living conditions for people in the premises being monitored
- To detect if alarms system require maintenance
- To allow impaired individuals to live independently for longer, through use of sensors monitoring and notifying the user
- To monitor active readings of a given property.
- To detect unsafe living conditions
How can you exercise your rights provided under GDPR
Under the General Data Protection Regulation, you have the right to access, rectify, port and erase your data. You can find more information on this in your "My account" page.
Data retention deactivation and deletion
Account data
We hold your account data until it is no longer necessary to provide our service to you, or until you chose to delete your account, whichever comes first.
When deleting your account with us all data will be removed and you won't be able to recover the information past this point. Note: if you have a career account connected to your account, they will have to delete their account separately if they wish to do so. But won't have access to any data related to your account after deletion
Sensor data
After 48 months all outdated Sensor data is deleted from our servers, unless requested by the user for it to be removed sooner. The user will also have the option to archive the reading locally
When deleting your account with us all data will be removed and you won't be able to recover the information past this point. Note: if you have a career account connected to your account, they will have to delete their account separately if they wish to do so. But won't have access to any data related to your account after deletion

Above is the privacy policy for the entire system. This was created with GDPR guidelines in mind and is informational and covers all privacy concerns for the system. This includes what data we collect, what we do with the data, our legal basis, how it is shared and our data retention information.



Above are the Terms and conditions for the system. This document contains all relevant information required to protect our legal position when it comes to the service we provide to our customers. This contains limitations of responsibility, copyright information and the legally binding user agreement the user needs to accept before making use of our services.



Both the Privacy policy and terms and conditions pages are located within the signup process of the system. And is a required field in the sign-up process. This

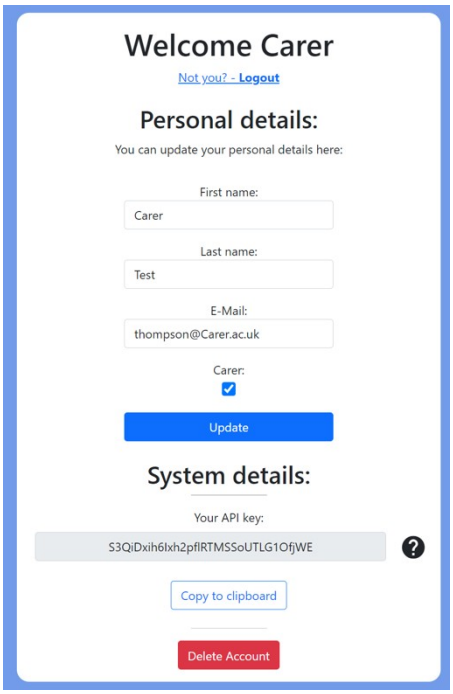
is where the customer agrees and accepts are policy for collecting and holding their data as well as not holding us liable for any damages whatsoever.

‘Your carers’ and ‘Your Patients’ page

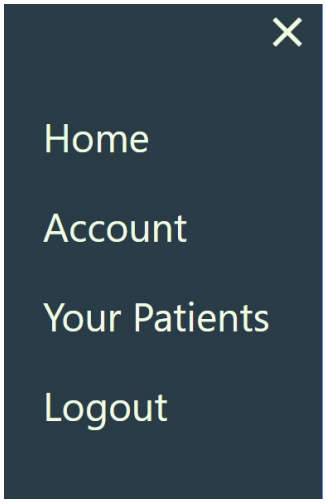
This section of the implementation covers the “Users could add a carer to the account to look over data from any geographic location” Functional requirement.

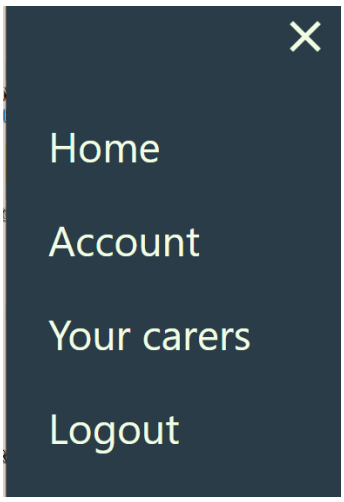
If the user wishes to convert their account to one of a carer’s they have enable the carer tick box located in the Account page. This Is linked to the AccountUpdateFrom which also allows the user to edit and correct any of their details. Below are images of the form's python file where both the tick box and add carer forms are located.

```
21
22  class AccountUpdateForm(FlaskForm):
23      first_name = StringField("First Name", validators=[InputRequired(), Length(max=30, message="First name cannot be longer than 30 characters.")])
24      last_name = StringField("Last Name", validators=[InputRequired(), Length(max=30, message="Last name cannot be longer than 30 characters.")])
25      email = StringField('Email', validators=[InputRequired(), Email()])
26      is_carer = BooleanField('Iscarer')
27
28  class AddCarerForm(FlaskForm):
29      carer_email = StringField('Email', validators=[InputRequired(), Email()], render_kw={'class':"form-control", 'placeholder':"Carer E-mail"})
```



Depending on whether the current account is set to a ‘carers account’ or not, depends on what you will see in your navigation bar. If the account is a carers account the user will see the option “Your Patients” Show up, which will allow you to manage and view your patient's sensor data. Or if the account is regular users they will see a “Your carers” option, where the user will be able to input a carers email address and link the two accounts together.

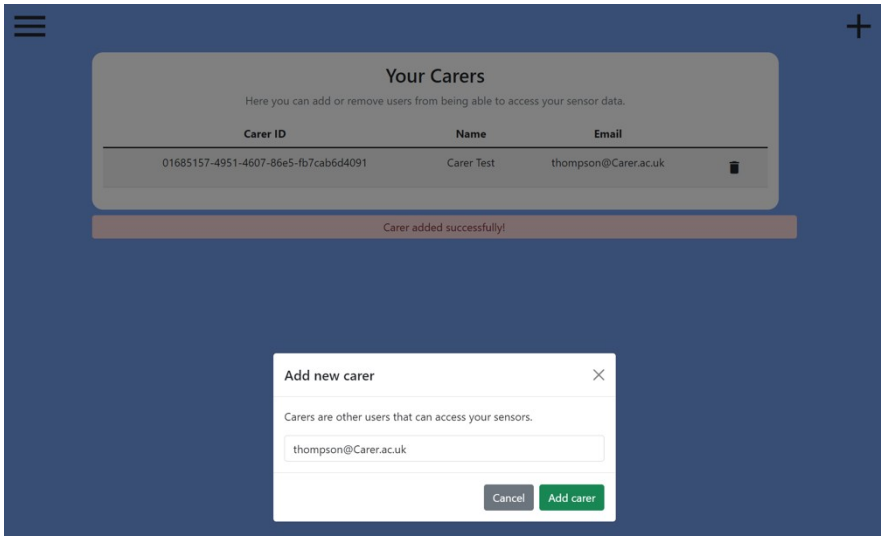




As both user and carer accounts operate from the same Database their different codes are combined in the same User model as seen below, however a user does not access to carers functionality unless their account is converted into one.

```
    def add_carer(self, carer_id):
        new_carer = UserCarer(User_ID=self.id, Carer_ID=carer_id)
        db.session.add(new_carer)
        db.session.commit()
        return new_carer


    def get_carers(self):
        carer_ids = UserCarer.query.filter_by(User_ID=self.id).all()
        carers = []
        for carer in carer_ids:
            carers.append(User.query.filter_by(id=carer.Carer_ID).first())

        return carers

    def get_patients(self):
        patient_ids = UserCarer.query.filter_by(Carer_ID=self.id).all()
        patients = []
        for patient in patient_ids:
            patients.append(User.query.filter_by(id=patient.User_ID).first())

        return patients
```

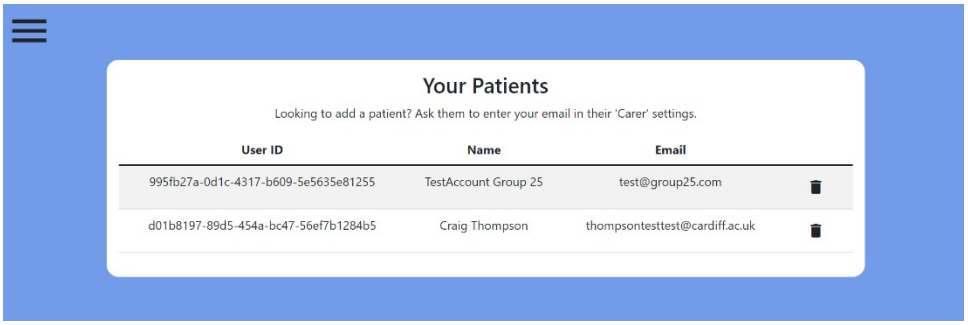
User perspective

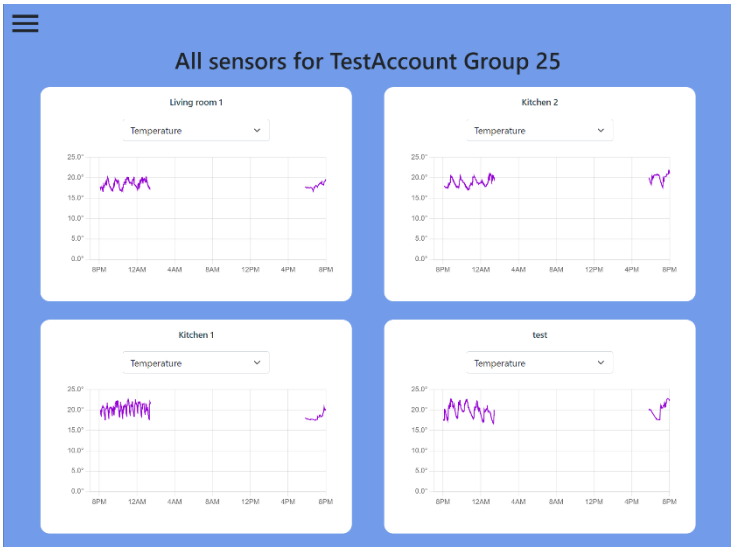


As a user in the system, they have access to adding a carer and deleting the carer. The user perspective can be found above as well as the route code can be found below. This functionality has been well built and well tested and will only allow you add accounts who have the carer account activated and exist in the system. After the user adds a carer to the system, the user carer will then have access to all active sensors from that user.

```python
@app.route("/addcarer", methods=["POST"])
def addcarer():
    if current_user.is_authenticated:
        CarerForm = AddCarerForm(request.form)
        if not CarerForm.validate():
            for field, error in CarerForm.errors.items():
                print(error[0])
                flash(f"Error - {error[0]}")
        else:
            carer_email = request.form.get('carer_email')
            carer = User.query.filter_by(Email=carer_email).first()
            current_carers = current_user.get_carers()

            if carer is None:
                flash("This user is not a carer.")
                return render_template('carers.html', form=CarerForm)

            if not carer.Is_Carer:
                flash("This user is not a carer.")
                return render_template('carers.html', form=CarerForm)

            if carer in current_carers:
                flash("You already have this user registered as a carer.")
                return render_template('carers.html', form=CarerForm)

            current_user.add_carer(carer.id)
            flash("Carer added successfully!")

        return render_template('carers.html', form=CarerForm)
    return '', 401

@app.route("/deletecarer", methods=["POST"])
def deletecarer():
    if current_user.is_authenticated:
        carer_id = request.form.get('carer_id')
        patient_id = request.form.get('patient_id')

        if current_user.id == patient_id or current_user.id == carer_id:

            carer = db.session.query(UserCarer).filter_by(User_ID=patient_id, Carer_ID=carer_id).first()

            if carer is not None:
                db.session.delete(carer)
                db.session.commit()
                return '', 200 #OK

            return '', 400 #Bad Request

    return '', 401 #Unauthorized
```

Carer Perspective





Above is the carers perspective In-Site, and below is the route code for the functionality as well as a snippet of code from the patients HTML file to show how each sensor is displayed to the system. Just like the user's perspective this has been well built and well tested and the carer cannot access any data that is not allowed to them by the user. Nor can they access data from other users we have not listed them as a carer. Please note that during this demonstration no data has been added to the sensors as they were made for this purpose, however in a real situation each sensor would display data in their respective graph.

```python
@app.route("/your_patients")
def your_patients():
    if current_user.is_authenticated and current_user.Is_Carer:
        return render_template('patients.html')
    return redirect(url_for("home"))


@app.route("/patient/<id>")
def patient(id):
    if current_user.is_authenticated:
        if db.session.query(UserCarer).filter_by(User_ID=id, Carer_ID=current_user.id).first():
            patient = db.session.query(User).filter_by(id=id).first()

            return render_template('patient.html',patient=patient, sensors= patient.get_sensors())
    return redirect(url_for("login"))
```

```
<script>
    {% for sensor in sensors %}

        $.get("/get_sensor_data?sensor_id="+ '{{ sensor.get_id() }}' +"&data_type="+ "0" +"&time_option="+ "0", function(data){
                    //{'Sensor_ID': 'ID', 'Summary':{'Avg': 0, 'Min': 0, 'Max': 0}, 'Data': [[timestamp, value], ...]}

                make_chart(data.Sensor_ID, data.Data, 0);
        });

        //Create drop down menu
        var options = ["Temperature", "Outside humidity", "Outside air pressure", "Outside temperature", "Reported humidity", "Light", "Motion", "Noise", "Air pressure", "V
        var dropdown = document.getElementById("dropdown {{ sensor.get_id()}}");

        for (var option in options) {
            var el = document.createElement("option");
            el.textContent = options[option];
            el.value = option;
            dropdown.appendChild(el);
        }

        dropdown.onchange = function() {
            $.get("/get_sensor_data?sensor_id="+ '{{ sensor.get_id() }}' +"&data_type="+ this.value +"&time_option="+ "0", function(data){
                make_chart(data.Sensor_ID, data.Data, this.value);
            });
        };
    {% endfor %}
```

All the codes seen above can be found in the zip file provided with the submission.

Testing

Approaches used to ensure good quality code

To measure if the quality of our code meets standards, we can measure it using the following aspects.

Reliability:

In order to determine the reliability of our code, we can test how many times our system can run without a system failure. In this case, we will add 10 sensors to our sensor dashboard and pass them data separately. By looking at the result, all sensors can successfully read data from the database and display them correctly. We have confidence that our system is reliable enough to handle a large amount of sensor data and display them correctly.

Correctness:

Data accuracy is important to our systems, it's our job to provide correct data summary to users, giving them a proper understanding of the environment they are in. In order to determine the correctness of our code, we have selected five sensors and tested whether their data summaries (e.g., maximum, minimum, average, latest) for different environmental data (e.g., temperature, humidity, air pressure, light, VOC, ECO2, etc.) are correct. By looking at the result, all sensors have provided the correct summary of different environmental data. All data points that out of the boundaries set by users will be marked red for warning purpose.

Portability:

We have tested our web-based system on multiple web browsers such as Chrome, Edge and Safari, and different systems such as Windows 10, macOS, Android and iOS. The test result shows that our system can fully operate on different systems and browsers. However, as our web-based system has not been deployed. We can just run our system in localhost for now.

Maintainability and reusability:

Our code has a tight structure and good readability. All UIs are written in the .html files under templates, database models are written in models.py, routes of the system are written in routes.py, Flask forms are written in forms.py, and the configs are written in config.py. The clear structure allows users to quickly locate and maintain or reuse the relevant code in the future.

Overall, we believe our code is of a good quality. As we follow a consistent code specification during development and test every time we write a new feature to ensure that errors are identified and corrected in a timely manner. We will also add comments to the code to ensure that we can understand the code written by other members of the group.

How the code has been tested from both users' and developers' perspectives

When using our web applications, there can be a difference in perspective between our developers and the typical user. Because we developers know every detail of the code and how the code works, the user does not know the inner workings of our code. This gap may be reflected in the fact that users don't know how to import data to the sensors because they don't have the basic programming knowledge and they don't know how to run a python program (we don't have actual access to the sensors yet). So before our system is deployed online and connected to the actual sensors, users may need the help of a developer to successfully import virtual data to the sensors in order to test if the system is working properly.

Test Cases

All test cases can be found in the Appendix

What our software does well

Some of the things our system does well Is that it always returns and displays the most recent sensor readings. Displaying a snippet of each data sensor on the "Sensor" screen with a detailed graph, where the user can view the past 24 hours and then enlarge each sensor if they wish to view the data further. In the enlarged view the user can also view different time settings like "Last 24 hours", "Last Week" or even "All time". As well as this the user can change the sensor reading selected, for example "Temperate",' HVOC" and so on.

Another thing our system does well is that it covers all basis for data protection and data privacy regulations. The system heavily follows GDPR regulations and contains both a detailed privacy policy and terms and conditions document which the user must agree to before continuing with making an account with us. This covers our legal basis, the collection and storing of the user's data, as well as protecting our legal liability.

What our software doesn't do well and future improvements.

Although the system does what it is supposed to do well, there are still some weaknesses. One of the weaknesses of the system is that it does not filter erroneous data from sensor readings. Although after some study and research on these types of errors we concluded that this area of error is very small and removed this feature from the system. Although it was removed from the system, the system would benefit from collecting erroneous data for future development of sensor prototyping as well as improving the accuracy of our system. A future improvement for this could be to add a button which the user can select whether they want filtered data in their readings or not, as well as stating that we would collect erroneous data for future improvements in the privacy policy

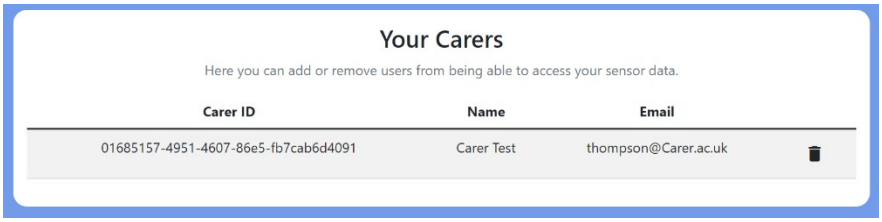
Another weakness of the system is the lack of instructions when it comes to adding new sensors/carers. As an advanced user the system the functionality is efficient and easy to use. However, for some new users who may need extra

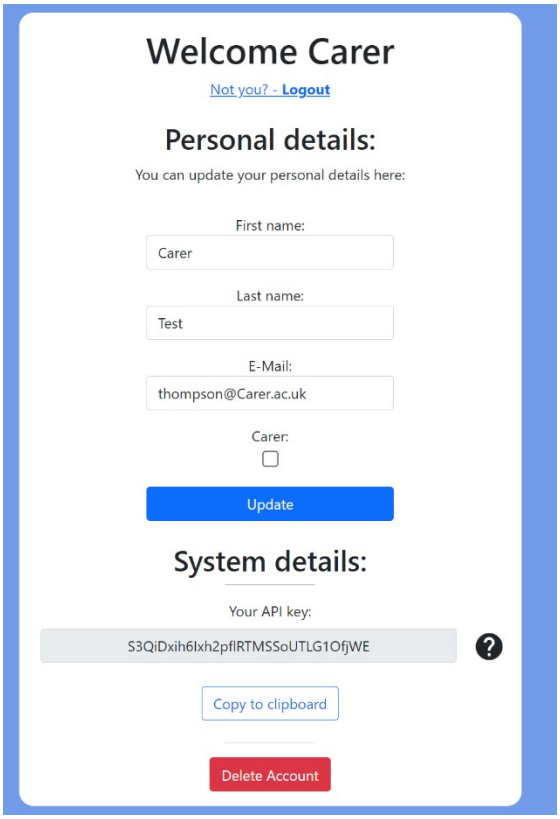
support navigating the In-Site documentation is lacking. For example, understanding that a carer needs to first create an account and switch their profile type in the account section of the navigation bar can be tricky to understand at first but once the user has done it once, they will not forget. An easy solution to solve this problem would be to simply add an F&Q section on the home page therefore the users can familiarize themselves with any issues they are having.

The final area to discuss is the notification system, currently there is nothing wrong with the current system for pushing notification to the user when a sensor is out of bounds. However, there are areas for improvement. As of right now the user will only get notified if they have the web application open on their device, therefore users cannot be made aware if the browser isn’t open. One solution is to create an app for the sensor dashboard; therefore, the notification system could push notifications straight to the user's phone instead of just through the web app. This would improve the system's effectiveness and design and create a more friendly experience

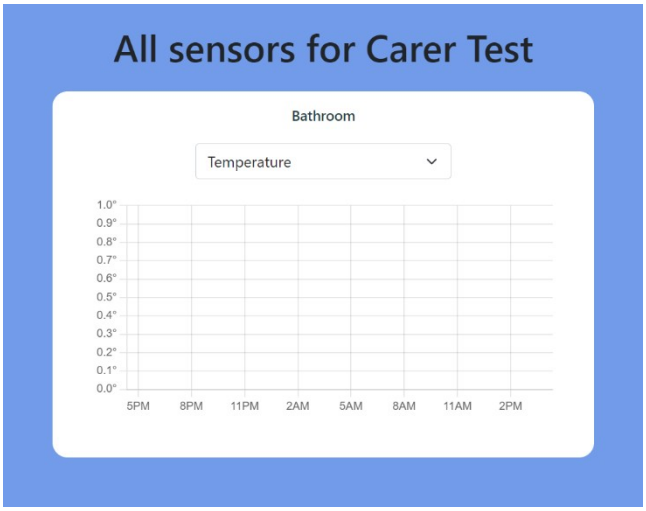
Demonstrating failure

During our thorough testing of the system, only 1 major issue occurred. This was with the carer functionality. When adding a carer to the system there is nothing stopping the user from adding themselves as their own carer as seen below.





This causes many inconsistency issues within the system and also opens up vulnerabilities in the system potentially causing bigger issues further down the line in the system's lifetime. This also uses up more unnecessary storage space within the database and therefore is something that needs to be addressed in the next iteration



As we can see the user has access to its own sensor dashboard twice. Once on the main dashboard and once through the carer menu if they make themselves a carer.

Note: sensor data is not being shown in the example above however displaying sensor data is fully functional in the system

Justification and evaluation of the presented software system

Justify design and implementation choices

For this project, we, as a group, decided to implement this system as a web application. We used Flask and python to implement this decision because we believed that it would be the best way to implement a system for elderly people as they can access it on any device that connects to a network. The use of a website also meant that it would be quicker and easier to implement than an IOS or Android app for example. We used Flask for the web application because it was easier to connect to a database in which we could add users and sensor data for each user. We used a MySQL database to store this information which was connected to the website using Flask. Flask also allowed us to use HTML and CSS to design the layout of the system meaning it was simple to layout each of the sensors on the home page as well as a login and register page. We also used JavaScript to create the graphs which display the data as well as calculate the average, minimum and maximum data for a certain sensor. The decision to use a website for the system allowed us to use JavaScript on the web pages to implement features from our requirements, such as Users will be able to store and manage account details in the system, easily and quickly.

Evaluation of system software after testing

After we tested our system with the tests written in the above section, it shows that our system meets most of the requirements we set out to complete. In total, the tests that we did show that we completed 6 out of 9 functional requirements and 8 out of 8 non-functional requirements. This means we met 66% of functional requirements and 100% of non-functional requirements after all the tests were finished.

Assumptions made

Some of the assumptions that we had to make during the creation of our system influenced some of the decisions we made when implementing our website. We assumed that the user will only want to see data from 24 hours, 7 days, 30 days, 6 months, last year and all time.

We also assumed that the user would have a valid email address to sign up with when registering. They can then use this email to log in and out to see their data. There is no alternative way of creating an account meaning the user must have an email when creating an account.

Another assumption we made was that the user will not need to change their password for their login. There is no way of changing the user's password on the login page as well as the account page if the user is already logged in.

Strengths of the system

When using our system, we specifically designed it to make it easy to navigate. Elderly people won’t have a problem finding the information they need when adding, deleting, and viewing sensor information as well as changing profile

information. The home screen isn’t cramped with information which might overwhelm the user as well.

Although the system is simple and easy to navigate, it does display all the necessary information for each room's sensor. It also allows the user to change what sensor information they are viewing and what time they want the information from with a drop-down box. The user can also see their maximum and minimum readings as well as an average from the time specified using the dropdown box. These features provide extensive information to the user.

Limitations of the system

Overall, the system is great, but it can be better if the user can access it through a portable device such as a mobile application because the notification system only alerts the user when the user has opened the browser. Therefore, the user can also use the system remotely if they have an internet connection.

Also, the system is easy to use so we ignore adding guidance to the system. We should have added guidance to avoid confusion for those who interact with our system for the first time. This can also be clearer to the user what features the system provided and the system are more trustworthy as it enhances the user experience and shows we had considered for the user.

The system is not flexible enough to filter erroneous data, the user cannot adjust the data collected by the sensor apart from deleting a sensor or adding a new sensor. If we improve the filter feature, the user will have more control over the data collected and be more accurate compared to the actual reading as the error data is customizable.

Fulfilled requirements and intended functionality

These are the user requirements that we implement successfully:

Fulfilled requirement 1: Users could view the sensor reading for a specific time range

Intended functionality: Users can choose to view sensor data within a specific time range and the data will be formatted in a graph

Fulfilled requirement 2: Users could view different summaries of the data provided by the sensor

Intended Functionality: Summaries of that sensor will be displayed (e.g., maximum, minimum, average readings). Beyond that, users are now able to review the time that the data is being recorded as an improvement feature on requirement 2.

Fulfilled requirement 3: The system can alert the user when the readings are outside the threshold defined by the user

Intended Functionality: Alert the user when the sensor readings are outside the threshold with a coloured indicator display in the graph

Fulfilled requirement 4: Users could define the threshold.

Intended Functionality: An alert will be triggered if readings are outside the threshold.

Fulfilled requirement 5: Users will be able to store and manage account details in the system

Intended Functionality: Users can be able to register for an account, manage and store their account details. So that they can add sensors to their account, view all the sensor readings, store the threshold they defined, and store all the preferences they have made.

Fulfilled requirement 6: Users will be able to log into the system

Intended Functionality: Users can register for an account and log in via the login page

Fulfilled requirement 7: Users must be displayed a dashboard of all sensors upon logging in

Intended Functionality: All the sensors are displayed on the dashboard once the user logs in successfully.

Fulfilled requirement 8: Users could add a carer to the account to look over data from any geographic location

Intended Functionality: Users can add a carer to their account after they log in to the system, and carers can view users' sensor data too after they log in to the carer account from any geographic location.

Fulfilled requirement 9: Users can delete all personal and sensor data relevant to them if they choose to

Intended Functionality: The users can choose to delete one or more sensors using the delete button and all personal data relevant to that sensor will also be removed from the databases.

Business case

For our system to be considered a proper business case, there are a few features that we need to improve.

Firstly, we need to test our sensor with actual data input. In the development stage, we rely on sensor.py to import sensor data to simulate our system. To achieve this, we need to create hardware, a demo sensor or an actual sensor that can connect with the system. The demo sensor needs to sense different types of data in a decided environment such as the reading of the temperature, humidity, air pressure, noise level, light, motion, VOC (Volatile Organic

Compound), and ECO2(Estimated CO2 level). Potentially, there will be a data transmission issue we need to solve, such as the speed of the data transfer and we also need to identify a method for data to transfer, e.g., we might transfer the data via Wi-Fi or Bluetooth but there are lots of factors we need to consider too, for example, the maximum range of the data transfer, is there an embedded memory location for saving the data if the sensor lost connection with the system, or the battery life of the sensor.

Secondly, the accessibility of the system. Now the user requires a browser to access the system. If the system can be assessed on application with iOS/Android devices, it could broaden user usage and potentially attract more customers to purchase the system. More importantly, the notification function only works while the user has their browser opened. Therefore, an application on the devices will let users receive notifications when they have not opened their browser and the user can also use the system remotely if they have an internet connection. This will be useful when the er would like to check the measure while they are not home or get alerted when they are not sitting in front of the computer to ensure the elderly live in a comfortable environment all the time.

Thirdly, we need to consider how we stored the data. To make a trustworthy system that contains a lot of personal data, we need to store the data safely and encrypted. It is because there is a risk that some criminals get access to the data server and know the record of the elderly home so that the criminals can infer whether the elderly are home alone or not from the noise level or the motion measurement. This might put the elderly at risk. Therefore, how we handle data needs thoughtful consideration. This also led to our next point, the maintenance of our system.

The system requires consistent maintenance. To understand what users desire, we will need to introduce a rating system or list our contact method for the users to provide feedback. Therefore, we will need a maintenance team to respond to feedback. Moreover, regular maintenance can ensure the safety of the system and update the user with a better interface or feature to enhance their user experience. For example, there will be an up-to-date version of the security system or a new feature such as users can download their sensor history from the data server.

If we can improve the aspects mentioned above, we will be one step closer to transforming our project into an actual business and potentially available on the market.

Influence of the potential issues

We have considered the potential impact on the following aspects:

Social
The carer needs to respect the decision of the elderly. Before the carer installs any sensors into the elderly home, the elderly needs to agree on the action first. It is because the elderly might not understand how the sensor works but the carer has the responsibility to explain to the elderly the purpose of the sensor instead of installing the sensor straight away. There is a chance that the elderly will be against the sensor technology as they may think if the carer applied the sensor the carer will rely on the sensor instead of visiting the elderly face to face. Therefore, we recommend the carer and the elderly communicate first before using the system.

Ethical
There is a human privacy issue related to our sensors because the sensors collect personal data and analysis them for healthcare purposes, such as the behavior of the user as there is a motion sensor embedded. Also, informed consent is important for the user because the user should have fully understood what data will be collected and used by the sensor. Therefore, we have introduced a privacy policy for user consent.

Legal
Due to the user's consent, there will be personal data stored on the data server. We will promise the user we will not use it for marketing purposes. It is because there is an increased number of people concerned about their data usage nowadays. This can comfort the user that their data will not be leaked, and the user is confident in using our system.

Professional
As this is a healthcare system aimed at improving the quality of life for the elderly. Our measurement needs to be correct so that it will meet our purpose of developing the system. Also, the user should be informed of all the benefits and risks when they are using the system. We will need to make sure all our group members are honest with the functionalities of the product while we present it to the clients.

Final Statement
To summarise this project, it’s a great system that could be used within lots of different scenarios which have been defined and discussed throughout this report. The system is robust, easy to use and is a great tool to have, so we believe it can be deemed as a success.

If we had more time, or more resources, we think this project could be developed in further ways, for example:

- An official app. This would provide similar functionality to the web app, but perhaps extensions. An example advancement would perhaps be utilising

the devices Bluetooth connection to allow for easy sensor linking and creation. Maybe even utilising biometrics to prevent users having to always enter their password etc.
- ‘Remember me’. We never got the opportunity to fully implement a ‘remember me’ option. For usability, it would be a lot easier if users didn’t always have to log in, so this would definitely be on the list.
- ‘Download my data’ – This would be a nice feature allowing users to access their data offline, or even externally for use with other systems.
- Integration with IoT devices – Finally, and most ambitious but perfectly doable with time & resources would be to be able to cool down a room when seeing its too hot using our solution, or shutting the curtains if the light level is too high.
- Finally, a nicer looking, easier to use graph. Whether it takes making our own framework or just finding a better system, a new way of plotting the data may be desirable if this project was going to be extended at all. The current Chart.js solution did the job, but can be frustrating to develop with.

Whilst there is room for improvement, our HomeSensors system meets the majority of our initial requirements. There’s always ways to make things better, but with a limited amount of time and resources, we believe what weve produced is a great system.

Bibliography

Flask-Login — Flask-Login 0.6.1 Documentation. https://flasklogin.readthedocs.io/en/latest/. Accessed 02 May 2022

Ulrich, Connie M., et al. ‘The Ethics of Sensor Technology Use in Clinical Research’. *Nursing Outlook*, vol. 68, no. 6, Dec. 2020, pp. 720–26. *PubMed*, https://doi.org/10.1016/j.outlook. Accessed 21 Apr 2022.

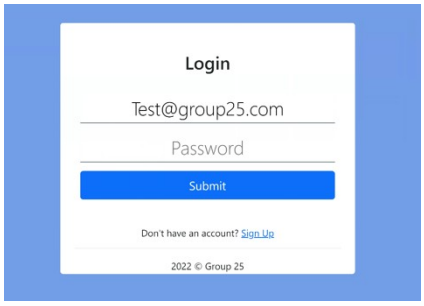
‘Staffing Shortages in the UK’s Care Sector: A Sign of Things to Come’. *Center for Global Development | Ideas to Action*, https://www.cgdev.org/blog/staffing-shortages-uks-care-sector-sign-things-come. Accessed 06 May 2022.

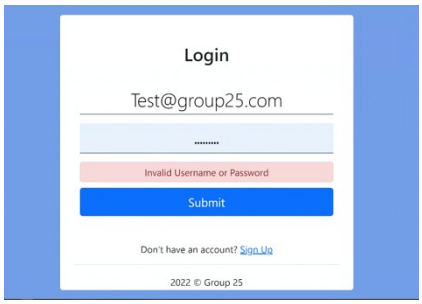
‘Technology for Older People: A Simple Guide’. *Lifeline24*, 20 Apr. 2022, https://www.lifeline24.co.uk/technology-for-older-people/. Accessed 06 May 2022.

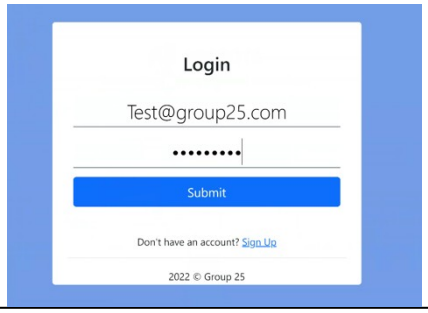
Appendix

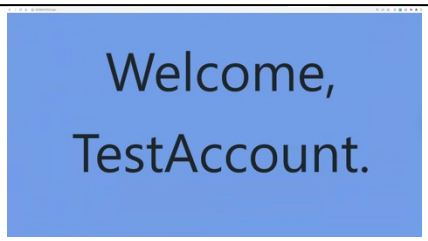
Test cases here

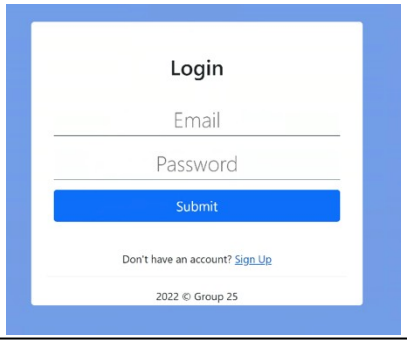
Test cases

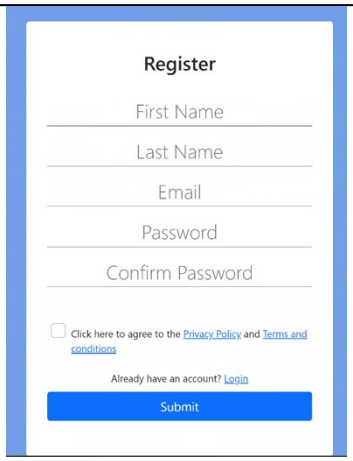
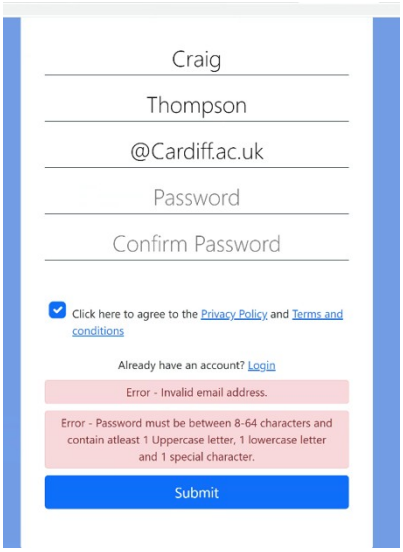
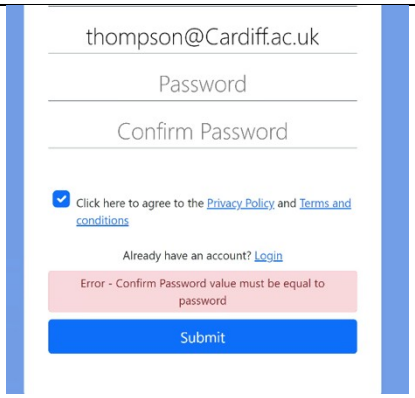
<table>
<tr><td colspan="2">Test Case ID: 1</td><td colspan="2">Test Purpose: Ensure the login system functions properly</td></tr>
<tr><td colspan="4">Preconditions: User has already registered an account</td></tr>
<tr><td colspan="4">Test Case Steps:</td></tr>
<tr><td>Step No.</td><td>Procedure</td><td>Result</td><td>Pass/Fail</td></tr>
<tr><td>1</td><td>Enter email</td><td> 

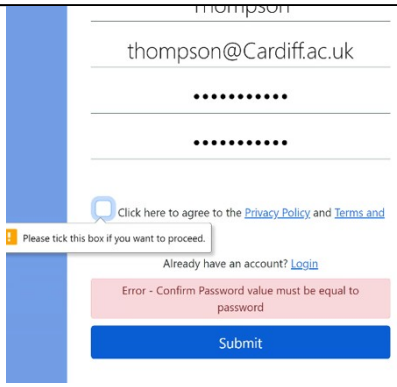
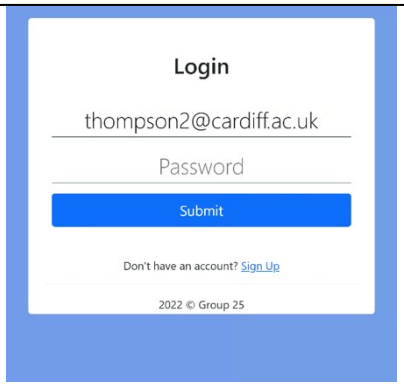
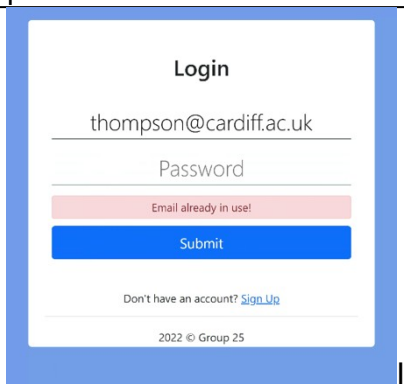
Email is entered into email slot</td><td>TEST PASSED</td></tr>
<tr><td>2</td><td>Enter wrong password (Different from the one that user has set)</td><td> 

The password is incorrect therefore the system does not allow the user to proceed. As well as displaying an error message” Invalid Username or Password”</td><td>TEST PASSED</td></tr>
<tr><td>3</td><td>Enter the correct password</td><td> 
</td><td>TEST PASSED</td></tr>
</table>

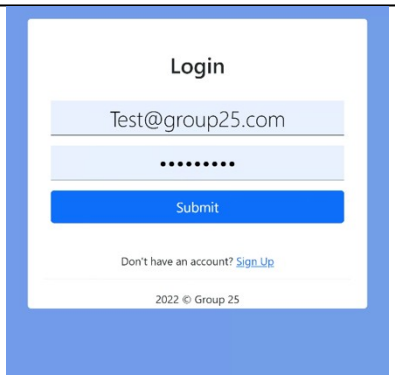
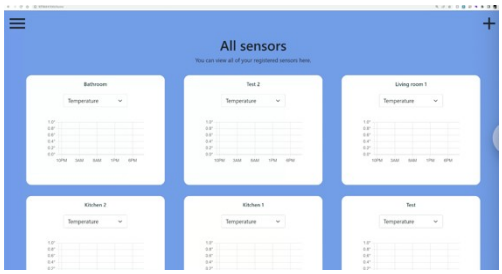
		 When the user inputs the correct details for a pre-registered account, the system welcomes the user and allows access to the sensor dashboard. Therefore, proving the log-in system is fully functional	
Comments: As an alternative flow, if the user does not have an account in the system, they can create one by clicking on the "Sign Up" link. This will be covered in the next test case			

Test Case ID: 2		**Test Purpose**: User/ Career can sign up for an account	
Preconditions: None			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	On log-in page, the user clicks "Sign up" and gets redirected to the sign-up page.		TEST PASSED

		 User successfully gets redirected to the Sign-up/ register page	
2	Users must input email address of email format. And password must contain a special character and be at least 8 characters in length	 When input doesn't meet the requirements multiple error messages appear.	TEST PASSED
3	Demonstrating the "Confirm password" must be the same as "Password"	 System doesn't allow the user to continue unless passwords match	TEST PASSED

4	Demonstrating the user cannot continue without agreeing to the privacy policy and terms and conditions.	 Message pops up asking user to complete the task before signing up	TEST PASSED
5	Users can click to view both Privacy Policy and Terms and Conditions	**<Show images here when updates>** **User can view and easily read the privacy policy and Terms of condition document**	TEST PASSED
6	User creates account successfully and gets taken to log in page.	 After creating an account, the user gets redirected to the login screen where the user can now log in, with the email address already preloaded.	TEST PASSED
7	Does not allow another account to be created if one with that email already exists	 f the user inputs an email address that has already been used in the system, it will redirect the user to the	TEST PASSED

		log in screen with the email pre-loaded. Along with a message saying" Email already in use"	
Comments:			

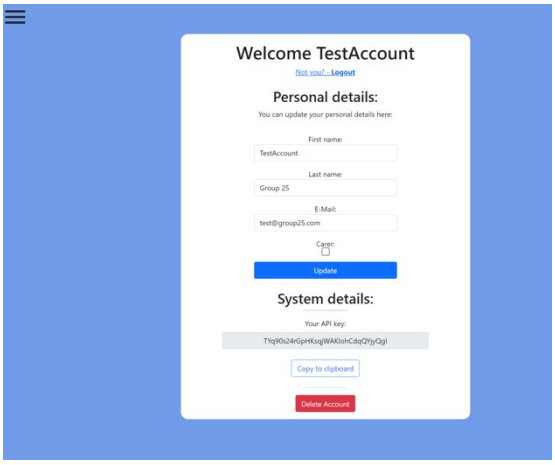
Test Case ID: 3	**Test Purpose**: Are all of a user's sensors displayed to them on the main page after logging in		
Preconditions: User has an existing account and has at least one sensor linked to it			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Go to login page, enter login details and press enter	 User logging into the system with an existing account	TEST PASSED
2	Check if all sensors linked to the account are displayed on the main page	 Upon entering the correct details after a welcome message, the first page the user is visible to is the sensor page with all active sensors. <MAYBE UPDATE PICTURE HERE TO HAVE DATA>	TEST PASSED

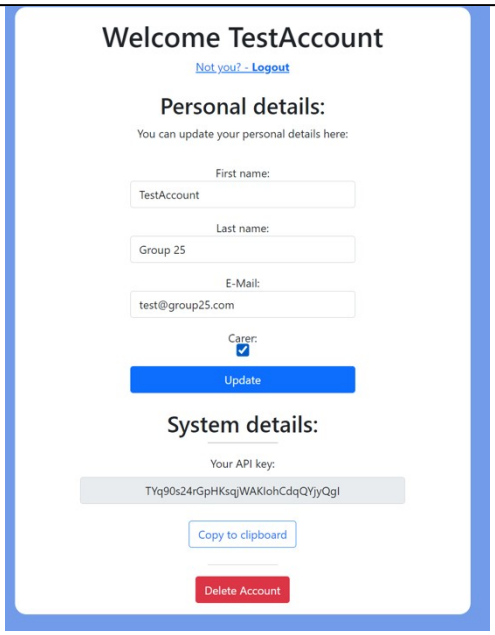
Comments:

Test Case ID: 4	**Test Purpose:** Can users view the sensor reading from a specific time range.
Preconditions: User is logged in and has at least one sensor linked to their account.	
Test Case Steps:	

Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Select a sensor to view by clicking on it from the main page		TEST PASSED
2	Selecting the time range to show data from	)e1e6a495 :e: Last 24 hours Last 24 hours Last 7 days Last 30 days Last 6 months Last year All time  After clicking” All time” for the time frame. It shows all data available for that sensor. In this example it shows all 2 years' worth of data given to us by our client as test data.	TEST PASSED
Comments:			

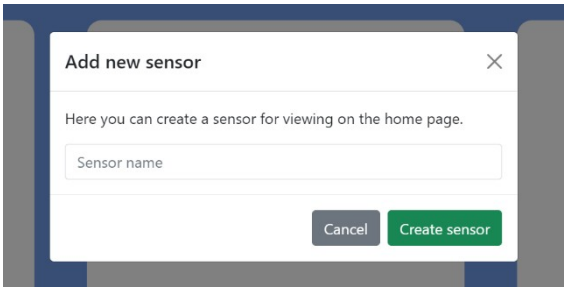
Test Case ID: 5	**Test Purpose:** Testing if the application is easy to navigate		
Preconditions: User is logged in			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Starting from the main page each time try to navigate to every page of the application in 3 clicks or less	Test 1   	TEST PASSED TEST PASSED

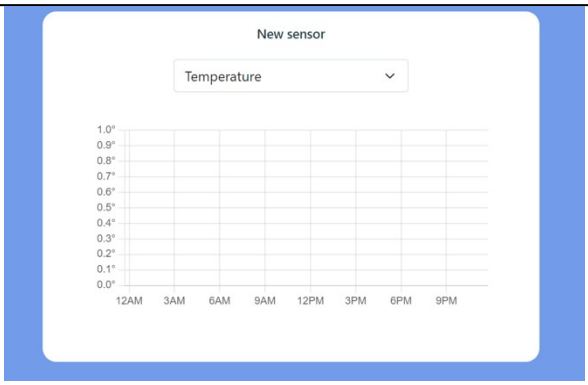
		 Viewing full sensor data accessible within 3 clicks Test 2  	TEST PASSED

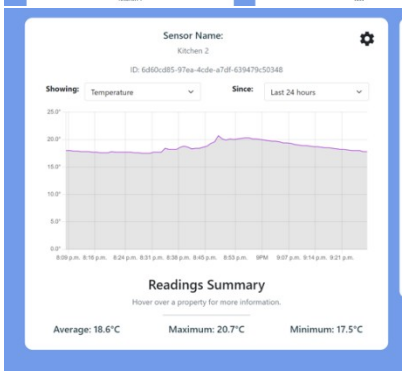


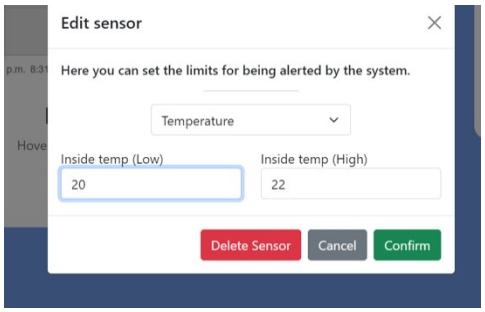
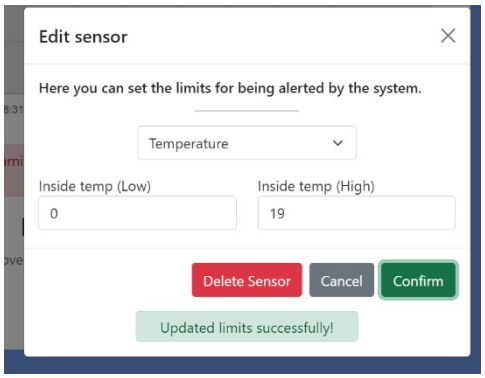
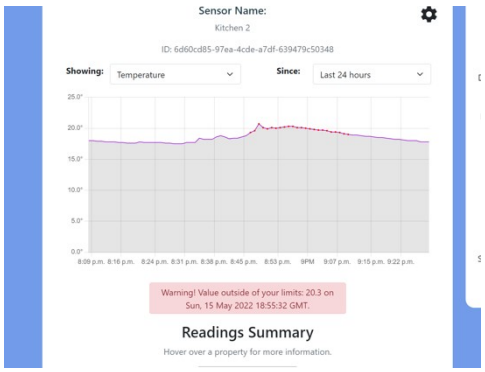
Making an account a carer account accessible within 3 clicks

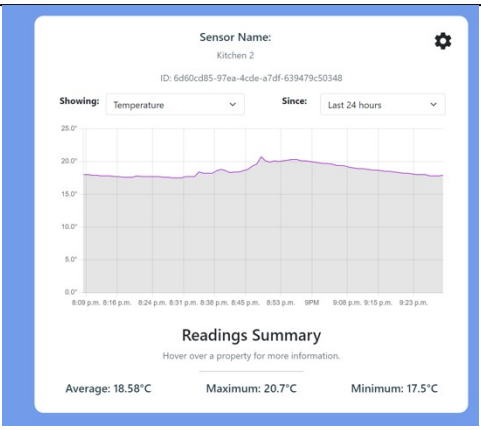
Test 3

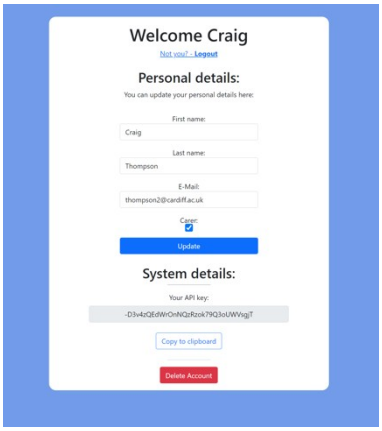


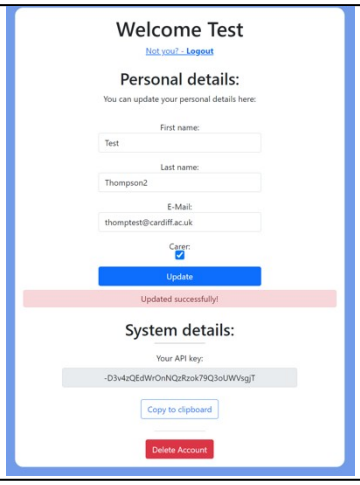
		 Adding a new sensor accessible within 3 clicks	
Comments:			

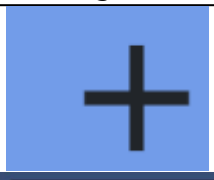
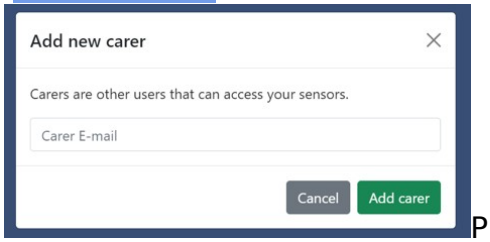
Test Case ID: 6	**Test Purpose:** Testing functional requirement 3 “The system can alert the user when the readings are outside the threshold defined by the user”	
Preconditions: User is logged in and has at least one sensor linked to their account.		
Test Case Steps:		
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Select a sensor from the dashboard screen	  User opens a sensor from the dashboard screen	TEST PASSED

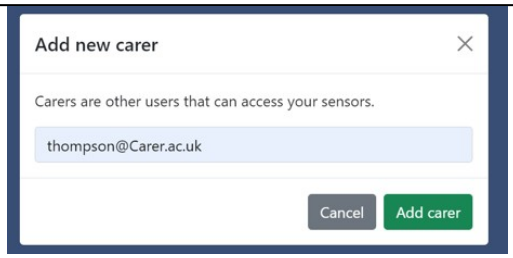
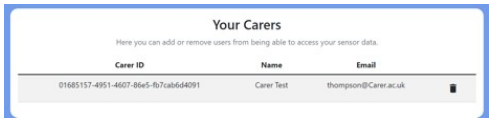
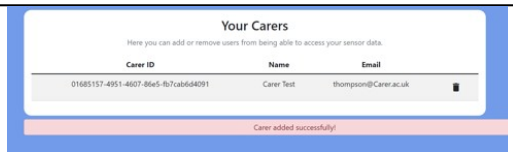
2	Click on the settings icon	 User selects the settings icon	TEST PASSED
3	Set the temperature minimum boundary higher than the current temperature	  User gets alerted if readings are below current threshold	TEST PASSED
4	Set the temperature maximum boundary lower than the current temperature	  User gets alerted if sensor data	TEST PASSED

		is above current threshold	
5	Set temperature boundaries to accept current temperature	 User doesn't get alerted if none of the data is out of bounds	TEST PASSED
Comments: Functionality works and alerts the user of out of bound data. It can be improved to send a notification to the users' phone or even email if the threshold is out of bounds for a long time			

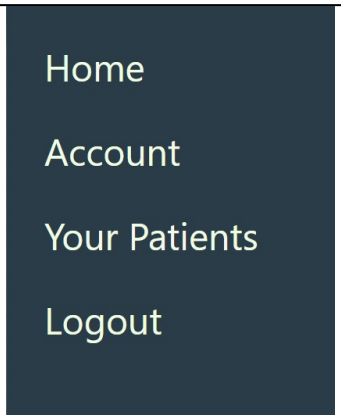
Test Case ID: 7		**Test Purpose:** Users will be able to store and manage account details in the system	
Preconditions: User has an account and is logged in			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Navigate to account settings from the navigation bar		TEST PASSED

2	Choose a new name and email to check if the name and email has been updated successfully		TEST PASSED
Comments: Ability to change password currently isn't available			

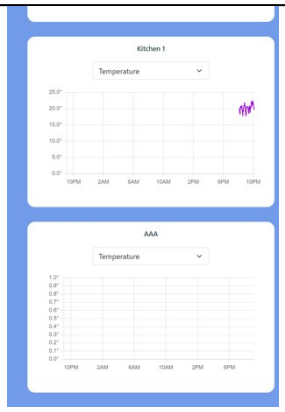
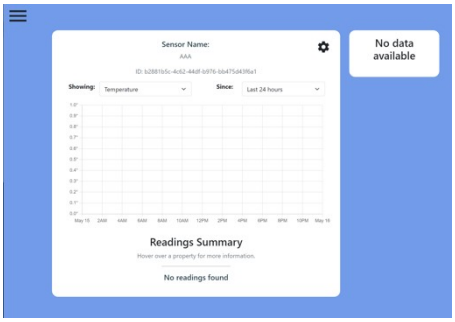
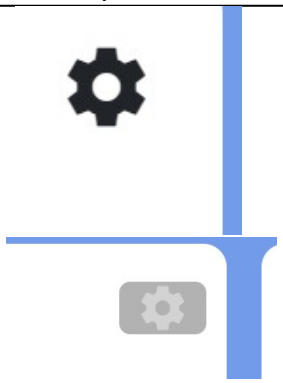
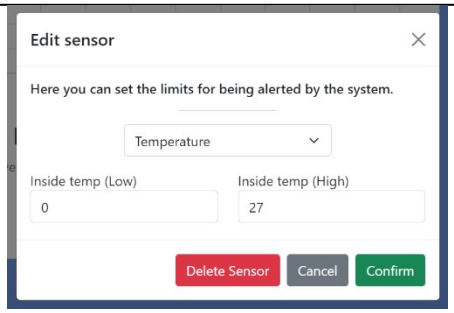
Test Case ID: 8	**Test Purpose:** Testing functionality, "Can the user assign a carer to their account"		
Preconditions: User has an account and knows the username of their carer			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	Click on the "Your Carers" button from the navigation bar	 Viewing carers page	TEST PASSED
2	Click the plus icon and open the add new carer menu	  Plus, icon opens add carer menu	TEST PASSED
3	Enter the carers E-mail,		

	and if the email exists it will add your account to the carers list	  Entering a carer email stores the carers details	TEST PASSED
4	Viewing the addition from the carers perspective	  Carer then has access to all of the user's active sensors	TEST PASSED
Failure	Proof you cannot add yourself as your own carer		TEST FAILED
Comments: Further development is required to limit who the user can add as a carer within the system			

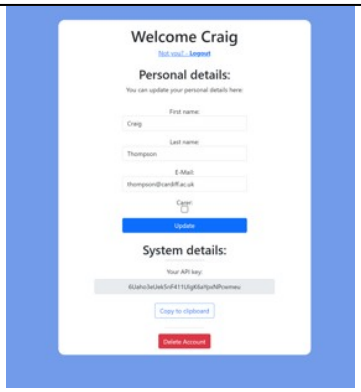
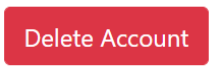
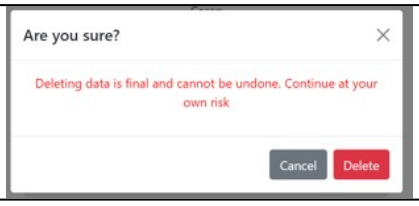
Test Case ID: 9		**Test Purpose:** Can the carer view the sensors of their users	
Preconditions: Logged into a carer account which has at least one user assigned to them			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**

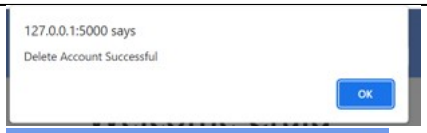
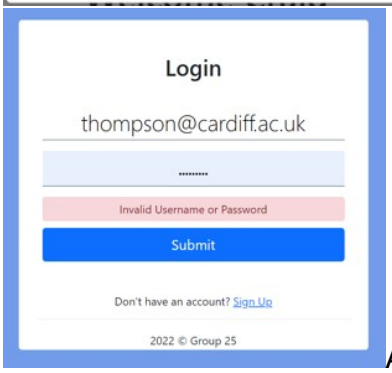
1	Click on ‘Your Patients’ button from the navigation bar	 Carer selects “Your Patients” option	TEST PASSED
2	Select a user	 Carer selects which user to view	TEST PASSED
3	Check if you can view all of the users' sensors	 Carer then has access to all data from that user	TEST PASSED
Comments:			

Test Case ID: 10		**Test Purpose:** Can the user delete sensors/ sensor data	
Preconditions: User is logged in and has at least one sensor linked to their account			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**

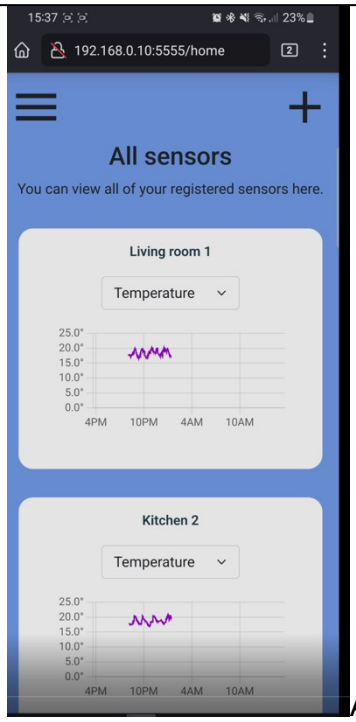
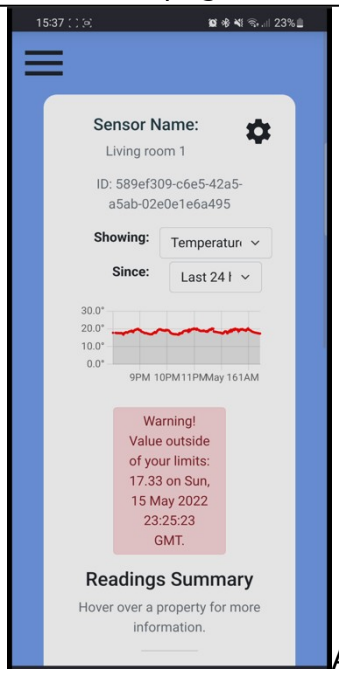
1	Click on a sensor from the main page	  User opens sensor menu	TEST PASSED
2	Click on settings icon on the sensor page	 User selects settings icon	TEST PASSED
3	Select “Delete sensor” button	 User then has access to the delete senser button	TEST PASSED
4	Confirmation		TEST PASSED

		 The user gets confirmation that the sensor has been deleted and dashboard is updated	
Comments: This can be improved by letting users delete specific ranges of data, for example delete all data from 2019, instead of just deleting everything			

Test Case ID: 11		**Test purpose:** User can delete their account if they wish to	
Preconditions: User has an existing account			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**
1	User opens “Account” page from navigation bar	 User opens account menu	TEST PASSED
2	User clicks “Delete account” button	 Click delete account	TEST PASSED
3	User confirms to delete account		TEST PASSED

		Confirms deletion and warned of risks	
4	Confirm deletion is successful and user cannot log in again	  Account permanently deleted and user cannot relog with same credentials	TEST PASSED
Comments:			

Test Case ID: 12		**Test purpose:** User can access the dashboard through a mobile phone	
Preconditions: User has an existing account			
Test Case Steps:			
Step No.	**Procedure**	**Result**	**Pass/Fail**

1	User can view dashboard on mobile device	 All sensors are visible on main dashboard page	TEST PASSED
2	User can view detailed sensor data	 All data, warnings and summaries are visible and available to the user	TEST PASSED
Comments: In the future the size of the graph on a mobile device can be bigger. As well as with further development the sensor dashboard can be made into an application			